

Draft Study Material

Web Developer

(QUALIFICATION PACK: Ref. Id. SSC/Q0503)

SECTOR: IT-ITeS

(Grade XI)



PSS CENTRAL INSTITUTE OF VOCATIONAL EDUCATION
(a constituent unit of NCERT, under Ministry of Education,
Government of India) Shyamla Hills, Bhopal- 462 002, M.P., India
<http://www.psscive.ac.in>

© PSS Central Institute of Vocational Education, Bhopal 2024

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior permission of the publisher.

PSSCIVE Draft Study Material ©Not to be Published

Preface

Vocational Education is a dynamic and evolving field, and ensuring that every student has access to quality learning materials is of paramount importance. The journey of the PSS Central Institute of Vocational Education (PSSCIVE) toward producing comprehensive and inclusive study material is rigorous and time-consuming, requiring thorough research, expert consultation, and publication by the National Council of Educational Research and Training (NCERT). However, the absence of finalized study material should not impede the educational progress of our students. In response to this necessity, we present the draft study material, a provisional yet comprehensive guide, designed to bridge the gap between teaching and learning, until the official version of the study material is made available by the NCERT. The draft study material provides a structured and accessible set of materials for teachers and students to utilize in the interim period. The content is aligned with the prescribed curriculum to ensure that students remain on track with their learning objectives.

The contents of the modules are curated to provide continuity in education and maintain the momentum of teaching-learning in vocational education. It encompasses essential concepts and skills aligned with the curriculum and educational standards. We extend our gratitude to the academicians, vocational educators, subject matter experts, industry experts, academic consultants, and all other people who contributed their expertise and insights to the creation of the draft study material.

Teachers are encouraged to use the draft modules of the study material as a guide and supplement their teaching with additional resources and activities that cater to their students' unique learning styles and needs. Collaboration and feedback are vital; therefore, we welcome suggestions for improvement, especially by the teachers, in improving upon the content of the study material.

This material is copyrighted and should not be printed without the permission of the NCERT-PSSCIVE.

Deepak Paliwal
(Joint Director)
PSSCIVE, Bhopal

Date: 28 September 2024

STUDY MATERIAL DEVELOPMENT COMMITTEE

MEMBER

Prof. K.V. Arya, ABV-Indian Institute of Information Technology & Management, Gwalior, M. P.

Prof. Vishal Goyal, Director, IQAC, GLA University, Mathura

Mr. Vijay Goswami, Founder & Director, Attrix Technologies, Agra, U.P.

Mr. Ankit Srivastav, NIT Agartala

MEMBER-COORDINATOR

Dr. Munesh Chandra, Professor (CSE), Department of Engineering and Technology, PSSCIVE, NCERT, Bhopal

TABLE OF CONTENTS

S.No.	Title	Page No
1.	Module 1. WEB DEVELOPMENT ESSENTIALS	1-79
	Module Overview	1
	Learning Outcomes	1
	Module Structure	1
	Session 1. Information Technology (IT) and Information Technology Enabled Services (ITeS)	2
	Check Your Progress	10
	Session 2. Web design and development	12
	Check Your Progress	19
	Session 3. Web Design and Development Tools	21
	Check Your Progress	27
	Session 4. Application Development Models of IT	29
	Check Your Progress	37
	Session 5. Web Designing Specifications	39
	Check Your Progress	51
	Session 6. Low-level and High-level Design for Programming	54
	Check Your Progress	62
	Session 7. Test and identify Design Defects	64
	Check Your Progress	69
	Session 8. Resolve Design Defects	72
	Check Your Progress	77
2.	Module 2. HTML AND CSS	80-152
	Module Overview	80
	Learning Outcomes	80
	Module Structure	80
	Session 1. HTML	81
	Check Your Progress	95
	Session 2. Concept of lists – Unordered lists, Ordered lists, Definition list	97
	Check Your Progress	119

S.No.	Title	Page No
	Session 3. CSS	121
	Check Your Progress	140
	Session 4. CSS box model	142
	Check Your Progress	151
3.	Module 3. WEB DEVELOPMENT USING JAVA SCRIPT	153-254
	Module Overview	153
	Learning Outcomes	153
	Module Structure	153
	Session 1. JavaScript	154
	Check Your Progress	181
	Session 2. Conditional Logic and Flow Control	183
	Check Your Progress	194
	Session 3. Arrays and Functions	196
	Check Your Progress	208
	Session 4. String Manipulation	210
	Check Your Progress	221
	Session 5. Manipulate Images using Javascript	223
	Check Your Progress	241
	Session 6. HTML5 Canvas	243
	Check Your Progress	253
4	Glossary	-
5	Answer Keys	255

Module 1**WEB DEVELOPMENT
ESSENTIALS****Module Overview**

In this Module, you will first learn about " Information Technology (IT) and Information Technology Enabled Services" covering Information Technology (IT) and Information Technology Enabled Services (ITeS) industry play a pivotal role in the global economy, driving innovation, productivity, and connectivity across various sectors. Then, we move to "Web design and development", where you will learn about web design and development, Constraints in Web Design, Key assumptions for creating websites and Interfaces of web designing process. Then, we discuss "Web Design and Development Tools", covering Web development standards, Web Development Tools, Types of websites, and Coding and programming for website.

Then, we discuss " Application Development Models of IT" covering will understand about IT application development, Importance of SDLC models in application development, Software Development Life Cycle (SDLC) models, Waterfall Model, Iterative Model and Agile model. Furthermore, explaining about "Web Designing Specifications", explaining Web design specifications, Business Requirement Specifications, Elements of Business requirement document, Benefits of Business Requirements Document, Steps to create Business Requirements Document and Software requirement specification. Next, we will cover "Low-level and High-level Design for Programming", which covers Low-level Design, High- Level Design, Object-Oriented Design, Database Design, API (Application Programming Interface) design, and Purpose of High-Level Design. After that "Test and identify design defects" which covers Design defects, types of defects, Common Causes of Design Defects and Stages of defect in software development life cycle. Lastly, we discuss "Resolve design Defects", covering Resolve design Defects, Scope of improvement for future design, Coding tools, and Recording and documentation of Design Defects. This Module equips you with essential knowledge and skills in IT/ITeS.

Learning Outcomes**Module Structure**

Session 1. Information Technology (IT) and Information Technology Enabled Services (ITeS)

Session 2. Web design and development

Session 3. Web Design and Development Tools

Session 4. Application Development Models of IT

Session 5. Web Designing Specifications
Session 6. Low-level and High-level Design for Programming
Session 7. Test and identify Design Defects
Session 8. Resolve Design Defects

Session 1. Information Technology (IT) and Information Technology Enabled Services (ITeS)

Maya, a passionate software developer in Techno Ville, thrived in the heart of the IT/ITeS industry. Assigned to a ground-breaking project, she discovered the synergy of IT, working with hardware, networking, and cybersecurity experts. Inspired by an IT consulting seminar, she delved into BPO, Digital Marketing, and Healthcare IT, realizing their societal impact. Maya's journey extended to India's "Digital India" campaign, leaving a digital legacy for the future as shown in Figure 1.1



Fig. 1.1: Maya working as software developer

In this chapter, the brief introduction about IT\ITes along with segment and relevance will be discussed. The Information Technology (IT) and Information Technology Enabled Services (ITeS) industry play a pivotal role in the global economy, driving innovation, productivity, and connectivity across various sectors. This industry encompasses a wide range of activities related to technology, data management, and digital services.

1.1 Information Technology (IT)

IT refers to the use of computers, software, networks, and other technology tools to process, store, transmit, and manipulate data for various purposes. It includes hardware and software development, IT support, and system administration

Information Technology Enabled services (ITeS) - ITeS, also known as Business Process Outsourcing (BPO), involves outsourcing business processes to third-party service providers.

Did you Know?

Information Technology Enabled Services (ITES) is a term that refers to the use of information technology (IT) to deliver a range of services to businesses and organizations.

1.1.1. Key Components

Hardware: This includes computers, servers, storage devices, networking equipment, and peripherals used in IT operations.

Software: IT relies heavily on software applications and operating systems for tasks ranging from data processing to running applications and managing hardware resources.

Networking: The IT industry relies on robust networks and telecommunications infrastructure to connect devices and facilitate data transfer.

Services: ITeS providers offer services like customer support, technical support, data entry, back-office operations, and more.

1.1.2. Importance

Economic Contribution: The IT/ITeS industry is a significant contributor to the global economy, generating substantial revenue and employment opportunities.

Innovation: IT is at the forefront of technological innovation, driving advances in artificial intelligence, cloud computing, cybersecurity, and more.

Global Connectivity: It enables seamless communication and data sharing across the globe, facilitating international trade and collaboration.

Efficiency: Through automation and digitalization, IT improves operational efficiency and productivity across industries.

1.1.3. Key Trends

Cloud Computing: The shift towards cloud-based services allows organizations to scale their IT infrastructure flexibly and cost-effectively.

Artificial Intelligence and Machine Learning: AI and ML are revolutionizing data analysis, automation, and decision-making processes.

Cyber Security: With increasing digitalization, the need for robust cybersecurity measures to protect data and systems is growing.

Online Work (Remote work): The COVID-19 pandemic accelerated the adoption of remote work and digital collaboration tools.

Big Data and Analytics: Analysing large datasets helps organizations make data-driven decisions and gain insights into customer behaviour and market trends.

1.2 Segments of IT/ITeS Industry

The Information Technology (IT) and Information Technology-enabled Services (ITeS) industry is a vast and diverse sector that encompasses various segments, each serving unique purposes and functions. Here are some of the key segments within the IT/ITeS industry:

Software Development: This is one of the most significant segments within the IT industry. It includes companies and professionals who design, develop, and maintain software applications and systems. This segment covers a wide range of technologies and platforms, including web development, mobile app development, enterprise software development, and more.

Hardware Manufacturing: This segment includes companies involved in the design, manufacturing, and distribution of computer hardware components such as processors, memory chips, motherboards, and storage devices. It also encompasses the production of computer peripherals like keyboards, monitors, and printers.

IT consulting and Services: IT consulting firms offer advisory services to businesses, helping them make strategic decisions related to technology adoption and implementation. These services can include IT strategy development, system integration, and technology optimization.

Cloud Computing and Hosting: This segment involves cloud service providers offer infrastructure, platforms, and software as a service (IaaS, PaaS, SaaS). These services enable

businesses to host and manage their applications and data in the cloud, reducing the need for on-premises infrastructure.

Cyber Security: Cyber security companies focus on protecting organizations' digital assets from cyber threats, including viruses, malware, data breaches, and hacking attempts. They provide services such as network security, identity and access management, and security consulting.

Business Process Outsourcing (BPO): BPO companies provide various back-office and front-office services to businesses, including customer support, data entry, human resources, finance, and accounting. BPO can be categorized further into voice-based (call centres) and non-voice-based services.

Digital Marketing and Advertising: This segment involves agencies and professionals who provide digital marketing services, including search engine optimization (SEO), social media marketing, content marketing, and online advertising. They help businesses promote their products and services online.

E-Commerce and Online retail: E-commerce companies operate online marketplaces where consumers can purchase products and services. This segment includes online retailers, payment gateways, logistics providers, and e-commerce technology companies.

Artificial Intelligence (AI) and Machine Learning (ML): AI and ML companies develop algorithms and systems that can analyse data, make predictions, and automate tasks. This segment has applications in various industries, including healthcare, finance, and autonomous vehicles.

Telecommunications: Telecommunication companies provide the infrastructure and services that enable voice and data communication. This includes internet service providers (ISPs), mobile network operators, and companies involved in the development of 5G technology.

Data Analytics and Business Intelligence: This segment involves companies that specialize in collecting, analysing, and interpreting data to help businesses make data-driven decisions. It includes data analytics tools, data warehousing, and data visualization solutions.

Gaming and Entertainment: Companies in this segment develop and publish video games, interactive entertainment, and digital media content. It also includes esports organizations and streaming platforms.

Healthcare IT: Healthcare IT companies provide technology solutions for the healthcare industry, including electronic health records (EHRs), telemedicine platforms, medical imaging software, and healthcare analytics.

Fintech: Fintech (financial technology) companies leverage technology to provide financial services, including digital banking, payment processing, peer-to-peer lending, and blockchain-based solutions like cryptocurrencies.

EdTech: EdTech companies focus on technology solutions for education and training, offering online courses, e-learning platforms, and educational software.

There are many segments within the IT/ITeS industry, and each segment plays a crucial role in shaping the digital landscape and driving innovation in various sectors of the economy.

Assignment 1.1.

- List down the IT/ITeS Key components.
- List down the IT/ITeS key trends.
- List down the key segments within the IT/ITeS industry.

Relevance of IT/ITeS Industry

The Information Technology (IT) and Information Technology-enabled Services (ITeS) industry continues to be highly relevant in today's world, and its importance has only grown in recent years. Here are some key reasons why the IT/ITeS industry remains relevant:

Digital Transformation: The IT/ITeS industry plays a crucial role in helping businesses and organizations undergo digital transformation. This includes adopting technologies like cloud computing, big data analytics, artificial intelligence (AI), the Internet of Things (IoT), and more. These technologies enable companies to become more efficient, competitive, and customer-centric.

Global Connectivity: IT/ITeS facilitates global connectivity and collaboration. Through the development of communication tools, software, and infrastructure, it has become easier for businesses to operate on a global scale, tapping into new markets and reaching a broader customer base.

Job Creation: The industry is a significant source of employment worldwide. It creates jobs not only in software development and IT support but also in areas like digital marketing, data analysis, cybersecurity, and project management. This employment spans across various skill levels, from entry-level positions to highly specialized roles.

Innovation: IT/ITeS is at the forefront of innovation. Tech companies constantly push the boundaries of what's possible, driving advancements in hardware, software, and services. Innovations like 5G technology, quantum computing, and autonomous vehicles all have roots in the IT sector.

Economic Growth: The industry contributes significantly to the global economy. It fuels economic growth by attracting investments, generating revenue, and fostering entrepreneurship. Many countries actively promote IT/ITeS as a strategic sector for economic development.

E- Government: Governments around the world have embraced IT/ITeS to improve the delivery of public services and enhance governance. This has led to greater transparency, efficiency, and citizen engagement through e-government initiatives.

Education and Training: Governments around the world have embraced IT/ITeS to improve the delivery of public services and enhance governance. This has led to greater transparency, efficiency, and citizen engagement through e-government initiatives.

Cyber Security: With the increasing reliance on digital technologies, cybersecurity has become paramount. The IT/ITeS industry plays a critical role in developing and implementing security measures to protect businesses, individuals, and governments from cyber threats.

Health care and Life Sciences: The industry has made significant contributions to healthcare and life sciences through the development of healthcare information systems, telemedicine solutions, genomics research, and drug discovery technologies.

Environment Sustainability: IT/ITeS can also contribute to environmental sustainability by enabling more efficient resource management, promoting remote work, and developing green technologies.

The IT/ITeS industry remains highly relevant due to its role in driving digital transformation, fostering innovation, creating jobs, contributing to economic growth, and addressing various societal challenges. As technology continues to advance, the industry's importance is likely to increase, shaping the future in profound ways.

1.2.IT application development Industry in India

IT application development is a thriving industry in India and has been a significant driver of the country's economic growth for several decades. India's IT sector, often referred to as the Indian IT industry, is known for its software development, IT services, and business process outsourcing (BPO) capabilities. Some of the key aspects of the IT application development industry in India:

Software services: Indian IT companies are renowned for their expertise in software development, including web and mobile application development. They offer a wide range of services, from custom software development to application maintenance and support.

Global Presence: Indian IT firms have a strong global presence and serve clients from various industries across the world. They have established development centres in multiple countries and work on projects that require diverse technology stacks.

Technology Stack: India's IT industry is well-versed in a variety of programming languages and technologies, including Java, Python, JavaScript, .NET, and more. They also have expertise in emerging technologies like AI, blockchain, and IoT.

Outsourcing and Offshoring: Many international companies choose to outsource their IT application development to Indian firms due to cost-effectiveness, skilled talent, and quality output. India has a large pool of software engineers and developers who are trained to meet global standards.

Start-ups and Innovation: India has a growing start-up ecosystem, with many new companies focused on application development and innovative technologies. Cities like Bengaluru (Bangalore), Hyderabad, and Pune have become hubs for tech start-ups.

Education and Training: India has a robust education system that produces a vast number of engineering and computer science graduates each year. Many of these graduates enter the IT industry after receiving training in software development.

Government Initiatives: The Indian government has introduced various initiatives to promote the IT industry, including the "Digital India" campaign, which aims to digitize government services and promote technology adoption across the country.

Challenges: Despite its success, the Indian IT industry faces challenges such as increasing competition, rising labour costs, data security concerns, and the need for continued upskilling to keep up with technological advancements.

Quality Standards: To maintain and enhance their global reputation, Indian IT firms often adhere to international quality standards and certifications like ISO and CMMI (Capability Maturity Model Integration).

Global Impact: Indian IT companies have played a crucial role in helping businesses worldwide adapt to digital transformation. They have been instrumental in developing and maintaining applications that drive various industries, including finance, healthcare, e-commerce, and manufacturing.

IT application development is a dynamic and influential sector in India, contributing significantly to the country's economy and its reputation as a global technology hub. It continues to evolve as technology advances and international demand for software development services remains strong.

1.3.Subsectors of IT application development:

IT application development is a broad field that encompasses various subsectors, each focusing on specific aspects of creating software applications. Some of the common subsectors of IT application development:

Web Development: Web developers specialize in creating websites and web applications. They work with technologies like HTML, CSS, JavaScript, and various web development frameworks to build interactive and responsive web applications.

Mobile app Development: Mobile app developers create applications for smartphones and tablets. This includes iOS app development (using Swift or Objective-C) and Android app development (using Java or Kotlin).

Desktop Application Development: Developers in this sub sector focus on building software applications that run on desktop operating systems like Windows, macOS, or Linux. They often use programming languages such as C++, C#, or Java.

Game Development: Game developers create video games for various platforms, including consoles, PCs, and mobile devices. This sub sector involves aspects like game design, graphics, sound, and physics simulations.

Database Development: Database developers design, implement, and maintain databases. They work with database management systems (DBMS) like MySQL, PostgreSQL, Oracle, or Microsoft SQL Server to ensure data storage and retrieval efficiency.

Enterprise Application Development: Enterprise application developers build software solutions tailored for businesses. These applications can include customer relationship management (CRM) systems, enterprise resource planning (ERP) software, and other tools that help organizations manage their operations.

Cloud Application Development: Developers in this subsector create applications that leverage cloud computing resources and services. This includes using platforms like Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP) to build scalable and reliable cloud-based applications.

Embedded System Development: Embedded systems developers work on applications that run on embedded hardware, such as microcontrollers and IoT devices. These applications can range from firmware for consumer electronics to industrial automation systems.

AI and Machine Learning Development: AI and machine learning developers build applications that use artificial intelligence and machine learning algorithms. These applications can include chatbots, recommendation systems, image recognition software, and more.

DevOps and Continuous Integration/ Continuous deployment (CI/CD): DevOps engineers and CI/CD specialists focus on automating and streamlining the development, testing, and deployment processes. They work with tools like Docker, Kubernetes, Jenkins, and Git to ensure smooth application delivery.

Blockchain Development: Blockchain developers create decentralized applications (DApps) and smart contracts on blockchain platforms like Ethereum, Hyperledger, or Binance Smart Chain.

Security Application Development: Security-focused developers specialize in creating applications and tools to protect systems and data from cyber threats. They may work on antivirus software, firewalls, intrusion detection systems, and encryption tools.

These sub sectors often overlap, and many developers acquire skills in multiple areas. The choice of sub sector depends on the specific goals and interests of the developer, as well as the demands of the projects they work on.

1.4. Different types of application development in IT industry:

Application development in the IT industry encompasses a wide range of software solutions designed to address various requirements and challenges. Some of the application development

in the IT industry are web application development, mobile application development, game development, cloud application development, healthcare and medical software, financial software development, educational software development and E commerce software development. The field is continuing to evolve with emerging technologies and changing business needs.

Assignment 1.2.

List down any five Subsectors of IT application development.

List down different types of application development in IT industry.

List down any three key aspects of the IT application development industry in India.

Career path and growth opportunities for web developer:

A career as a web developer offers numerous growth opportunities and can take various paths, depending on your interests, skills, and career goals. An overview of potential career paths and growth opportunities for web developers, as shown in Figure 1.2.

The career of a Web developer starts as a junior web and moves up to chief technology officer. Here freedom is that, you can work as a freelance web developer, taking on projects for clients and building your portfolio. You can start your web development agency if you have entrepreneurial aspirations. The field of web development often allows for remote work, giving you the flexibility to work with clients and companies worldwide. Remember that personal development and growth in a web development career require continuous learning, adaptability, and a willingness to embrace new technologies and challenges. Building a strong portfolio, mastering best practices, and staying connected to industry trends will help you excel in this field.



Fig. 1.2: Career paths and growth opportunities for web developers

1.5. Roles and Responsibilities of Web Developer:

The roles and responsibilities of a web developer can vary depending on the specific job, organization, and the technology stack they work with. However, here are some common roles and responsibilities associated with web developers:

Front-End Development

- **User Interface (UI) Design:** Creating visually appealing and user-friendly web interfaces.
- **HTML/CSS:** Writing and maintaining HTML and CSS code for website layout and styling.
- **JavaScript:** Developing client-side scripting to enhance interactivity and user experience.
- **Responsive Design:** Ensuring websites are accessible and perform well on various devices and screen sizes.

Back-End Development

- **Server-Side Scripting:** Writing code that runs on the server to handle data processing and business logic.
- **Database Management:** Designing, implementing, and maintaining databases for web applications.
- **API Development:** Building and integrating APIs for data exchange between the front-end and back-end systems.
- **Security:** Implementing security measures to protect web applications from threats like hacking and data breaches.

Full-Stack Development

Combining front-end and back-end development skills to work on the entire web application development process.

SUMMARY

- The IT/ITeS industry encompasses technology, data management, and digital services.
- IT includes hardware, software, networking, and IT services.
- ITeS involves outsourcing business processes using technology.
- Key components include computers, software, and networking equipment.
- The industry drives economic growth, innovation, and global connectivity.
- Key trends include cloud computing, AI, cyber security, and remote work.
- IT application development has diverse sub sectors like web and mobile development.
- The Indian IT industry is known for software development, IT services, and BPO.
- Web developers have roles in front-end, back-end, and full-stack development.
- Staying updated with industry trends is essential for web developers.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS (MCQ)

1. What does IT stand for? (a) Internet Technology (b) Information Technology (c) Innovative Technology (d) International Technology
2. Which of the following is a key component of the IT/ITeS industry? (a) Agriculture (b) Manufacturing (c) Hardware (d) Construction
3. What is the primary role of IT consulting firms in the IT/ITeS industry? (a) Software development (b) Customer support (c) Advisory services (d) Data entry
4. Which trend has accelerated the adoption of remote work in the IT industry? (a) Cloud computing (b) Artificial Intelligence (c) Cybersecurity (d) Online work
5. What does BPO stand for in the context of IT/ITeS? (a) Business Process Optimization (b) Business Process Outsourcing (c) Business Process Organization (d) Business Process Oversight
6. What sector does the IT/ITeS industry play a crucial role in promoting environmental sustainability? (a) Healthcare (b) Telecommunications (c) Gaming and Entertainment (d) Fintech
7. Which Indian city is known as a hub for tech start-ups in the IT application development industry? (a) New Delhi (b) Mumbai (c) Bengaluru (Bangalore) (d) Chennai
8. Which subsector of IT application development focuses on creating decentralized applications and smart contracts on blockchain platforms? (a) Game Development (b) Database Development (c) Blockchain Development (d) Mobile App Development
9. What are some common types of application development in the IT industry? (a) Game development, healthcare software development, and cloud application development (b) E-commerce software development, financial software development, and educational software development (c) Web application development, mobile application development, and database development (d) All of the above
10. Which role in web development focuses on the user interface and user experience of a website? (a) Front-end developer (b) Back-end developer (c) Full-stack developer (d) Database developer

B. Fill in the blanks

1. IT refers to _____ Technology.
2. ITeS, also known as _____ Outsourcing (BPO).
3. Hardware includes computers, servers, _____, networking equipment, and peripherals used in IT operations.
4. The IT industry relies on robust networks and telecommunications infrastructure to connect _____ and facilitate _____ transfer.
5. Cloud _____ allows organizations to scale their IT infrastructure flexibly and cost-effectively.
6. AI and ML are revolutionizing data analysis, _____, and decision-making processes.
7. The shift towards _____ services enables businesses to reduce the need for on-premises infrastructure.

8. Web developers specialize in creating websites and _____ applications.
9. Game developers create _____ for various platforms, including consoles, PCs, and mobile devices.
10. Blockchain developers create decentralized applications (DApps) and smart contracts on _____ platforms like Ethereum.

C. True or False

1. The IT/ITeS industry encompasses a wide range of activities related to technology, data management, and digital services.
2. Hardware components include computers, servers, storage devices, networking equipment, and peripherals used in IT operations.
3. Cloud Computing allows organizations to scale their IT infrastructure inflexibly and expensively.
4. AI and ML are not used in data analysis and automation processes.
5. The Indian IT industry is known for software development, IT services, and business process outsourcing.
6. Game developers focus only on PC and console games, not mobile games.
7. Indian IT firms often adhere to international quality standards and certifications like ISO and CMMI.
8. Web developers are only responsible for designing websites, not coding them.
9. Web developers are not required to stay up-to-date with the latest industry trends and technologies.
10. The IT/ITeS industry has no relevance in promoting environmental sustainability.

D. Short Question Answers

1. What is the primary focus of the Information Technology (IT) industry?
2. Define Information Technology Enabled Services (ITeS) and provide an example.
3. Name one key component each from Hardware, Software, and Networking in the IT industry.
4. How does the IT/ITeS industry contribute to global connectivity and collaboration?
5. What key trend in the IT industry accelerated the adoption of remote work?
6. What are the primary responsibilities of web developers in the IT industry?
7. How does the IT/ITeS industry promote economic growth?
8. Name two subsectors of IT application development mentioned in the chapter.
9. Why is staying up-to-date with industry trends crucial for web developers?
10. How does the IT/ITeS industry contribute to environmental sustainability?

Session 2. Web Design and Development

Once there was a guy named Anil. He was curious about the internet and how websites worked. So, he took a class to learn about building websites from his teacher, Mr. Vijay. Anil made a simple website called "The Web Wiz" to help people learn about making websites. It got popular in school because it had easy lessons, fun quizzes, and a place to ask questions. His website helped lots of students, and even a community center asked Anil to talk about web design. In the end, Anil learned that being curious and trying new things could lead to great achievements. He was happy he could help others along the way. As shown in Figure 2.1.



Fig. 2.1: Anil making website

In this chapter, you will learn about web design and development, Constraints in Web Design, Key assumptions for creating websites and Interfaces of web designing process.

2.1. Web Design and Development

Web design and development is a multifaceted process that involves creating and maintaining websites.

Did You Know?

Web design and development is an umbrella term that describes the process of creating a website.

It encompasses various stages, from planning and design to coding and testing.

1. Planning and Analysis:

- First, Identify the purpose and goals of the website, its main function, and the target audience.
- Conduct market research and competitor analysis to gather insights.
- Plan the content structure and decide what information will be presented on the website.

2. Information Architecture:

- Create a visual representation of the website's structure, outlining all pages and their relationships.
- Design rough sketches or wireframes of key pages to establish layout and functionality.

3. Design:

- Develop the website's visual style, including colors, typography, and imagery.
- Create detailed mockups or prototypes of key pages to visualize the design.

4. **Development:**

- **Front End Development:** Build the user interface using HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript. This is what the user interacts with directly.
- **Back End Development:** Create the server-side logic, databases, and application functionality. Common technologies include PHP, Python, Ruby on Rails, Node.js, etc.
- **Content Management System:** If needed, integrate a CMS like WordPress or Drupal to make content management easier for non-technical users.
- **Responsive Design-** Ensure the website is accessible and functions well on various devices and screen sizes. Typically, we use a CSS framework to achieve that.

5. **Content Creation:**

- Add text, images, videos, and other media to the website.
- Optimize content for search engines by using proper keywords and metadata.

6. **Testing:**

- Test all features and functionality to ensure they work as intended.
- Verify that the website functions correctly on different browsers and devices.
- Check load times and optimize performance.
- Ensure the website is secure against common vulnerabilities.

7. **Deployment:**

- Choose a domain name and set up web hosting.
- Deploy all website files to the hosting server.
- Configure DNS settings to point the domain to the hosting server.

8. **Launch:**

- Do one last round of testing to ensure everything is working on the live server.
- Create backups of the website and server configurations.
- Inform stakeholders and users about the website's launch.

9. **Maintenance and Updates:**

- Regularly update content, security patches, and software to keep the website current and secure.
- Monitor website performance and user feedback for ongoing improvements.
- Plan for future enhancements and expansions as needed.

Web design and development is an ongoing process that requires attention to detail and continuous improvement to meet evolving user needs and technological advancements. Collaboration between designers, developers, and content creators is crucial for a successful outcome.

2.2. Concept of web design and development:

Web design and development are two closely related but distinct disciplines that are essential for creating and maintaining websites and web applications.

2.2.1. Web design:

Web design refers to the process of creating the visual and user interface (UI) elements of a website or web application. It focuses on the aesthetics, layout, and overall look and feel of a website.

- (i) **Visual Elements:** This includes choosing color schemes, typography, images, graphics, and other visual elements to create a visually appealing and coherent design.
- (ii) **User Experience (UX):** Web designers strive to create a positive user experience by making websites easy to navigate, ensuring intuitive user interfaces, and optimizing for mobile and other devices.
- (iii) **Layout and Structure:** Designers determine the arrangement of content, buttons, menus, and other elements on web pages to enhance user engagement and accessibility.
- (iv) **Responsive Design:** Ensuring that the design adapts and looks good on various screen sizes and devices is crucial, a concept known as responsive design.
- (v) **Wireframing and Prototyping:** Designers often create wireframes and prototypes to plan and test the website's layout and functionality before development begins.
- (vi) **Graphic Design:** Graphic design skills are often needed to create custom icons, logos, and other visual assets for the website.

Assignment 2.1.

- List down the various stages of Web design and development.
- List down the names of various elements used for web design.

Web Development:

Web development, on the other hand, is the process of building the functionality and interactivity of a website or web application. Web developers use programming languages and frameworks to turn the design into a functional website. Key aspects of web development include:

- (i) **Front-End Development-** Front-end developers work on the client side, writing code (HTML, CSS, JavaScript) that determines how the website will be displayed and how users will interact with it in their web browsers.
- (ii) **Back-End Development-** Back-end developers focus on the server-side of the website. They handle databases, server scripting, and server infrastructure to ensure that websites can process data, handle user accounts, and perform various functions.
- (iii) **Full Stack Development-** Some developers are proficient in both front-end and back-end development, known as full-stack developers. They can work on all aspects of a web project.
- (iv) **Database Management-** Storing and retrieving data efficiently is crucial. Web developers often work with databases (e.g., MySQL, MongoDB) to manage and store information.
- (v) **Security-** Developers must be vigilant about security, implementing measures to protect websites and user data from various threats, including hacking and data breaches.
- (vi) **Testing and Debugging-** Developers test and debug code to ensure the website functions correctly across different browsers and devices.
- (vii) **Deployment and Maintenance-** After development, websites need to be deployed to web servers, and ongoing maintenance and updates are essential to keep the site secure and up-to-date.

Did You Know?

Web design and development are collaborative processes, with designers and developers working together to create websites that are visually appealing, user-friendly, and functional.

Basic elements for web designing:

Web design involves several fundamental elements that work together to create a visually appealing and user-friendly website. The some of the basic elements for web design are

Overall layout-The overall layout of the website may

- Grid Structure: Establish a grid system to organize content and maintain consistency throughout the website.
- Responsive Design: Ensure that the layout adapts to different screen sizes and devices, providing a seamless user experience on desktops, tablets, and smartphones.
- White Space: Use white space (negative space) effectively to enhance readability and reduce clutter.

Colour scheme:

- Choose a color palette that aligns with your brand and conveys the desired mood or emotion.
- Use color consistently for elements like headings, text, links, buttons, and backgrounds.
- Consider accessibility standards to ensure that color choices are legible for all users, including those with visual impairments.

Typography:

- Select appropriate fonts for headings and body text, focusing on readability and aesthetics.
- Maintain font consistency throughout the website to create a cohesive design.
- Pay attention to font sizes and spacing for improved readability, and consider responsive typography for different screen sizes

Navigation content- Design clear and intuitive navigation menus, such as a top menu bar or a hamburger menu for mobile devices. Ensure that navigation labels are descriptive and user-friendly. Implement breadcrumb trails, sitemaps, and a search bar to help users find information easily throughout the website.

Content- Create high-quality, engaging, and relevant content that caters to the needs and interests of your target audience. In order to convey information effectively and easily the Use of multimedia elements, such as images, videos, and infographics is required. Organize content logically into sections and pages, making it easy for users to digest and navigate.

Imagery and Graphics-

- Incorporate images and graphics that align with your brand and enhance the visual appeal of the website.
- Optimize images for web performance by compressing them without sacrificing quality.
- Use responsive images to ensure they load appropriately on various devices and screen sizes.

User Interaction and Feedback-

- Include interactive elements like buttons, forms, and social media integrations.
- Provide feedback to users through hover effects, animations, and notifications to enhance the user experience.
- Ensure that forms and interactive elements are user-friendly and accessible.

Consistency- Maintain visual and functional consistency across all pages of the website. Use style guides and design patterns to establish and adhere to design standards.

Accessibility- Follow accessibility guidelines (such as WCAG) to ensure that your website is usable by individuals with disabilities. Provide alternative text for images, ensure proper contrast, and use semantic HTML elements.

Performance- Optimize website performance by minimizing page load times through techniques like image optimization, lazy loading, and caching. In order to have best performance, test the website on various browsers and devices to ensure cross-compatibility.

The effective web design is not static, it should be continually assessed and updated to meet evolving user expectations and design trends. The gather the user feedback regularly and conducting usability testing can help you refine and improve your website's design.

2.3. Constraints in Web Design

Constraints in web design refer to limitations or restrictions that designers must work within to create effective and functional websites. These constraints can come from various sources and impact different aspects of web design. Some of the common constraints in web design:

Technical Constraints- Designers must ensure that websites work correctly and appear consistently across different web browsers (e.g., Chrome, Firefox, Safari, Internet Explorer). Websites should be responsive and adapt to various devices, including desktops, laptops, tablets, and smartphones.

Accessibility Constraints- Designers must follow accessibility standards to make websites usable for people with disabilities. This includes considerations for screen readers, keyboard navigation, and other assistive technologies. Ensuring sufficient contrast between text and background colors to accommodate users with visual impairments.

Content Constraints- Websites often need to accommodate various types of content, from text and images to videos and interactive elements. Designers must organize and present content effectively. Balancing the amount of content on a page to avoid overwhelming users while providing enough information.

Brand and Style Constraints- Designers must adhere to brand guidelines, which include the use of specific colors, fonts, logos, and other branding elements. Maintaining a consistent design and user experience across the entire website to build brand identity and improve user recognition.

Budget Constraints- Designers often have to work within budget constraints, which can impact the choice of design elements, technology stack, and the scope of the project.

Usability Constraints- Designers need to create intuitive and user-friendly interfaces that allow users to navigate and interact with the website easily. Organizing content and creating effective navigation menus are essential for a positive user experience.

Time Constraints- Designers often work within tight schedules, which can impact the depth of design exploration and testing

Understanding and effectively managing these constraints is crucial for web designers to deliver websites that meet the needs of both clients and users while maintaining high-quality standards. Balancing these constraints requires creativity, problem-solving skills, and a deep understanding of web design principles and best practices.

2.4. Key assumptions for creating websites

Creating a website involves a range of key assumptions and considerations to ensure that the website meets its intended goals and serves its target audience effectively. Some of the key assumptions to consider

1. **Purpose and Goals-** Before creating a website, it's crucial to have a clear understanding of its purpose and goals. Are you building an e-commerce site, a blog, a portfolio, or a corporate website? Define the primary objectives of the website.
2. **Target Audience-** Identify your target audience and their needs. Consider their demographics, preferences, and behaviours to tailor the website's content and design to meet their expectations.
3. **Content Strategy-** Determine what type of content you'll provide on the website. This includes text, images, videos, and other multimedia elements. Plan how often you'll update and maintain the content.
4. **Platform and Technology-** Choose the right platform and technology stack for your website. This might involve deciding between content management systems (CMS) like WordPress, custom development, or website builders like Wix or Squarespace.
5. **User Experience (UX) and Design-** Ensure that the website is user-friendly and visually appealing. Consider factors like navigation, layout, color schemes, and typography to create a positive user experience.
6. **Mobile Responsiveness-** Given the increasing use of mobile devices, assume that your website will be accessed on various screen sizes. Make the site responsive to ensure it functions well on mobile phones, tablets, and desktops.
7. **Performance Optimisation-** Optimize the website's performance to reduce load times. This includes optimizing images, using content delivery networks (CDNs), and minifying code.
8. **Hosting and Domain-** Choose a reliable hosting provider and a suitable domain name. Ensure that your hosting plan can handle your website's requirements and traffic.
9. **Marketing and Promotion-** Develop a marketing strategy to promote your website. This may involve social media, email marketing, content marketing, and other promotional activities.
10. **Budget and Resources-** Estimate the budget required for website development, hosting, marketing, and ongoing maintenance. Allocate resources accordingly.
11. **Backup and Recovery-** Implement regular backup and disaster recovery plans to safeguard your website's data and ensure quick recovery in case of unexpected events.

Assignment 2.2.

- List down any three common constraints used in web design.
- List down any three key assumptions for creating websites.

Interfaces of web designing process

Web design is a multifaceted process that involves various stages and interfaces, both in terms of the design itself and the collaboration between designers, developers, and clients. Here is an overview of the interfaces in the web design process

1. **Client\ Designer Interface-** The process begins with a meeting between the client and the designer to discuss project goals, requirements, and expectations. The designer creates a project proposal outlining the scope, timeline, and cost of the project for the client's approval.
2. **Research and Planning-** Ongoing communication between the client and designer to gather information about the target audience, competitors, and project objectives. The Designers collaborate with their team members to conduct market research and gather insights.

3. **Wireframing and Prototyping-** Designers create wireframes and prototypes of the website's layout and functionality. Clients review and provide feedback on wireframes and prototypes.
4. **Design Development-** Designers work on the visual elements, including color schemes, typography, images, and graphics. Designers collaborate with developers to ensure the design is feasible and optimized for web development.
5. **Content Creation-** The client provides content such as text, images, and videos, which designers incorporate into the design. Designers work with content writers to ensure that the content fits the design and conveys the intended message effectively.
6. **Design Reviews and Revision-** Clients review the design and provide feedback for revisions. Designers communicate feedback to the design team for necessary adjustments.
7. **Testing-** Developers test the functionality of the website, ensuring it works correctly across different browsers and devices. Clients may also perform user testing to provide feedback on usability.
8. **Client Approval and Deployment-** Clients review the final design and provide approval for launch. Developers deploy the website to a web server, making it accessible to the public
9. **Maintenance-**the Designers may provide ongoing support and updates as needed, based on client requests. The Users interact with the website, and provide feedback for further refinements. The Developers may perform maintenance and updates as per client feedback to keep the website secure and up-to-date.

Throughout the web design process, effective communication is essential between all interfaces to ensure that the final product aligns with the client's goals and meets the needs of the target audience. Collaboration between designers, developers, content creators, and clients is critical for a successful web design project.

SUMMARY

- Web design and development involve a multifaceted process that includes planning, analysis, design, coding, testing, deployment, and maintenance.
- The process begins with defining the website's purpose, target audience, and content structure.
- Information architecture involves creating a visual representation of the website's structure and wireframes for key pages.
- Web design focuses on aesthetics, user experience (UX), layout, responsiveness, and visual elements.
- Web development encompasses front-end and back-end development, databases, security, testing, deployment, and maintenance.
- Content creation and optimization for search engines are crucial in web design.
- Testing ensures the website functions correctly, performs well, and is secure.
- Deployment involves choosing a domain, hosting, and DNS configuration.
- Regular maintenance, updates, and user feedback are vital for ongoing improvements.
- Web design and web development are distinct but closely related disciplines, requiring collaboration.
- Basic elements of web design include layout, color schemes, typography, navigation, content, imagery, user interaction, consistency, accessibility, and performance optimization.

- Interfaces in the web design process include client/designer, research/planning, wireframing/prototyping, design development, content creation, design reviews/revision, testing, client approval/deployment, and maintenance.
- Effective communication and collaboration among these interfaces are essential for successful web design projects.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What is the purpose of the Planning and Analysis stage in web design and development? (a) Creating visual elements (b) Coding the website (c) Determining website goals and target audience (d) Managing the website's content
2. Which front-end technologies are commonly used in web development? (a) PHP (b) Python (c) HTML, CSS, JavaScript (d) Ruby on Rails
3. What is the main focus of web design? (a) Building server-side logic (b) Enhancing user experience (c) Managing databases (d) Writing code for web browsers
4. What does UX stand for in web design? (a) User Experience (b) User Extension (c) User XML (d) Universal Experience
5. What is responsive design? (a) Designing with multiple colors (b) Designing for mobile devices (c) Optimizing images (d) Creating database logic
6. Which constraint involves making websites usable for people with disabilities? (a) Technical Constraints (b) Content Constraints (c) Accessibility Constraints (d) Brand Constraints
7. Which element is crucial for creating an effective web design layout? (a) Grid Structure (b) Brand Colors (c) Mobile Apps (d) Marketing Strategy
8. What is one of the key assumptions when creating a website? (a) There's no need to understand the target audience (b) The budget is not important (c) Mobile responsiveness isn't necessary (d) A clear purpose and goals must be defined.
9. Which interface involves the collaboration between the client and designer to discuss project goals? (a) Client/Designer Interface (b) Research and Planning Interface (c) Testing Interface (d) Maintenance Interface
10. What is the main purpose of wireframes and prototypes in web design? (a) Final website launch (b) Client meetings (c) Layout and functionality planning (d) Marketing strategy

B. Fill in the blanks

1. In web development, _____ developers work on the user interface using HTML, CSS, and JavaScript.
2. Content creation includes adding text, images, videos, and other media to the _____.
3. Effective web design should be continually assessed and updated to meet evolving user expectations and _____ trends.
4. Web design is a multifaceted process that involves various _____ and interfaces.

5. Web design refers to the process of creating the visual and _____ of a website or web application.
6. Web developers work with databases to manage and _____ information.
7. Graphic design skills are often needed to create custom icons, logos, and other visual assets for the _____.
8. The effective web design is not _____.
9. Designers must ensure that websites work correctly and appear consistently across different web _____.
10. In _____, the overall layout includes grid structure, responsive design, and effective use of white space.

C. True or False

1. Web design is primarily focused on creating server-side logic.
2. Web developers use programming languages to build the functionality of a website.
3. Web designers are not concerned with user experience (UX).
4. Responsive design ensures that websites look the same on all screen sizes.
5. Designers must work within budget constraints, which can impact design elements.
6. Usability constraints in web design involve creating user-friendly interfaces.
7. Time constraints have no impact on the depth of design exploration and testing.
8. Wireframes and prototypes are used for marketing purposes.
9. Content creation involves only adding text to a website.
10. The client and designer interface are where the project begins with a discussion of project goals.

D. Short Question Answers

1. What are the main stages involved in the web design and development process?
2. Explain the purpose of the Planning and Analysis stage in web design.
3. What is the significance of determining the target audience for a website?
4. Describe the difference between front-end and back-end development in web development.
5. How does responsive design contribute to the success of a website?
6. What role does content creation play in web design, and why is it important?
7. What are the common constraints in web design, and how do they impact the design process?
8. Why is maintaining consistency in web design, such as using a consistent color scheme and typography, essential?
9. What is the importance of following accessibility standards in web design?
10. What are the key assumptions to consider when creating a website?

Session 3. Web Design and Development Tools

Meet Aarti, a curious kid who loves the internet. She wanted to learn how websites were made, so she took a class with Mr. Ajay. Aarti learned about special tools:

Text Editor: It's like a special notepad for writing website stuff.

HTML: It's the skeleton of a web page, telling it how to look.

CSS: It's like a paintbrush, making web pages look cool.

Images and Graphics: They make web pages pretty, like using crayons for drawings.

Web Browser: It's a magic window to see web pages.

With these tools, Aarti created "The Music," a helpful website for learning. Kids everywhere loved it. Aarti learned that these tools were his superpower, helping him create cool things on the internet and teach others. As shown in Figure 3.1.



Fig. 3.1: Aarti making website

In this chapter, you will learn about Web development standards, Web Development Tools, Types of websites, and Coding and programming for websites.

3.1. Web development standards

Web development standards and tools are essential components of building high-quality, reliable, and accessible websites and web applications. These standards help ensure compatibility across different browsers and devices while improving the user experience and

Did You Know?

Web development standards ensure that web pages are consistent, compatible, and easy to maintain across different devices, browsers, and platforms.

The following are some of the most important web development standards and tools:

1. **HTML (Hypertext Markup Language):** HTML is the foundation of web development. It defines the structure and content of web pages. Staying up-to-date with HTML5 standards is crucial for modern web development.
2. **CSS (Cascading Style Sheets):** CSS is used for styling web pages. CSS3 introduces advanced styling features and is commonly used for responsive design.

3. **JavaScript:** JavaScript is a programming language used for adding interactivity and functionality to web pages. ECMAScript 6 (ES6) is the latest standard for JavaScript.
4. **Accessibility:** Web Content Accessibility Guidelines (WCAG) provide standards for creating web content that is accessible to people with disabilities. Ensuring your web projects are accessible is not only a best practice but, in some cases, a legal requirement.
5. **HTTP/HTTPS:** Hypertext Transfer Protocol (HTTP) and its secure version (HTTPS) are fundamental protocols for web communication. Using HTTPS is essential for data security and SEO.
6. **Responsive Design:** Creating websites that adapt to various screen sizes and devices is crucial. Media queries and responsive design principles are standard practice.
7. **Semantic Markup:** Using semantic HTML elements (e.g., `<header>`, `<nav>`, `<section>`, `<article>`) helps convey the meaning and structure of your content to both browsers and developers.
8. **Progressive Enhancement:** Build web experiences that work for all users, regardless of their device or browser capabilities. Start with a basic, functional version and enhance it for modern features.

3.2. Web Development Tools:

1. **Text Editors and IDEs:** Popular text editors for web development include Visual Studio Code, Sublime Text, and Atom. Integrated Development Environments (IDEs) like WebStorm are also used by many developers.
2. **Version Control:** Tools like Git and platforms like GitHub or GitLab help developers track changes, collaborate with others, and manage code versions effectively.
3. **Package Managers:** npm (for JavaScript) and Composer (for PHP) are examples of package managers used to install and manage dependencies in web projects.
4. **Task Runner and Build Tools-** Tools like Gulp, Grunt, and Webpack automate tasks like minification, compilation, and optimization to streamline development workflows.
5. **Browsers and Developer Tools:** Modern browsers (Chrome, Firefox, Edge) come with developer tools that assist in debugging, profiling, and optimizing web applications.
6. **Front End frameworks and Libraries:** Popular frameworks and libraries like React, Angular, and Vue.js simplify the development of complex web applications.
7. **Back End Frameworks:** Frameworks like Express (Node.js), Django (Python), and Ruby on Rails (Ruby) provide structure and utilities for server-side development.
8. **Content Management Systems (CMS)-** CMS platforms like WordPress, Drupal, and Joomla offer pre-built solutions for managing and publishing web content.
9. **Testing and Debugging Tools:** Tools like Jest (for JavaScript testing), Selenium (for browser automation), and Postman (for API testing) help ensure the reliability of your web applications.
10. **Performance Optimisation Tools:** Tools like Google PageSpeed Insights and Lighthouse can help you analyse and optimize the performance of your websites.
11. **Security Tools:** Tools like OWASP ZAP and security scanners help identify and address security vulnerabilities in web applications.
12. **Hosting and Deployment Services:** Services like AWS, Heroku, and Netlify offer hosting and deployment solutions to make it easier to publish web applications.

3.3. Types of Website

Websites can be broadly categorized into two main types based on their functionality and how they deliver content to users: static websites and dynamic websites.

3.3.1 Static Websites:

Content: Static websites consist of web pages with fixed content that does not change unless manually edited by a web developer or administrator.

Technologies: They are typically built using HTML, CSS, and sometimes JavaScript for basic interactivity.

Characteristics:

- **Fixed Content:** The content on static websites remains constant until someone with web development knowledge updates it.
- **Fast Loading:** Static websites are generally faster to load because the server simply serves pre-built HTML pages to users without any processing.
- **Simple Hosting:** They can be hosted on basic web servers, and there is no need for server-side scripting or databases.
- **Limited Interactivity:** Static sites are not designed for extensive user interactions or personalized content delivery.
- **Example Use Cases:** Personal blogs, small business websites, portfolios, and informational websites with content that doesn't change frequently.

Did you know?

Static website is a site with little to no user interaction, and the design is generally consistent on all platforms.

Dynamic Websites:

Content: Dynamic websites generate content on the fly, often pulling data from databases or other sources, allowing for real-time updates and user interactions.

Technologies: Dynamic websites use a combination of server-side scripting languages (e.g., PHP, Python, Ruby) and client-side technologies (HTML, CSS, JavaScript).

Characteristics:

- **Data-Driven:** Content is generated dynamically based on user input, database queries, or external APIs.
- **User Interactivity:** Users can interact with dynamic websites through forms, logins, comments, and other features.
- **Personalization:** Content can be customized for individual users based on their preferences or behaviour.
- **Content Management Systems (CMS):** Many dynamic websites use CMS platforms like WordPress, Joomla, or Drupal to facilitate content creation and management.
- **Example Use Cases:** E-commerce websites, social media platforms, online forums, news websites, and any site where content needs to change frequently or be tailored to user preferences.

Did You Know?

A dynamic website is a website that changes as users interact with it.

It's important to note that many modern websites incorporate elements of both static and dynamic content. For instance, a website might have static pages for its core content (like the homepage and about page) and dynamic elements for user accounts, product listings, or real-time notifications.

Ultimately, the choice between a static and dynamic website depends on the specific goals and requirements of the project. Static websites are simple and easy to maintain but lack interactivity, while dynamic websites offer more functionality and flexibility but require more complex development and hosting infrastructure.

3.3.3 Difference between static and dynamic websites: Static and dynamic websites are two distinct types of websites that serve different purposes and have different characteristics.

3.1 Table for Difference between static and dynamic websites

Aspect	Static Websites	Dynamic Websites
Content	Fixed content/ rarely changes.	Content can change dynamically.
Design	Uniform design and layout across all pages.	Design and layout can vary by page or user.
Interactivity	Limited interactivity and user engagement.	Highly interactive and responsive to user actions.
Data	Data remains constant unless manually updated.	Data can be fetched from databases or external sources.
Maintenance	Easier to maintain with fewer updates.	Requires regular maintenance and updates.
Development	Simple to develop as it involves static files.	More complex development with server-side scripting.

The choice between a static and dynamic website depends on your specific needs. Static websites are suitable for simple, low-maintenance sites, while dynamic websites are better for complex, interactive, and frequently updated web applications.

3.4 Coding and programming for website

Coding and programming for a website involves creating the underlying code that makes a website function and displays content. Websites are typically built using a combination of HTML, CSS, and JavaScript, and may also involve server-side programming languages and databases for dynamic functionality.

3.4.1 Front-end development tools:

Front-end development is an essential part of web development that focuses on creating the user interface and user experience of a website or web application. HTML, CSS, and JavaScript are the core technologies used for front-end development.

1. **HTML (Hypertext Markup Language):** HTML is the foundation of web development. It defines the structure and content of web pages. HTML5, the latest version, introduced many new elements and attributes that enhance the capabilities of web pages, such as audio and video embedding, semantic tags, and improved form handling.
2. **CSS (Cascading Style Sheets):** CSS is used for styling web pages. CSS is used to control the presentation and layout of web pages. It allows you to define styles for HTML elements, such as fonts, colors, margins, padding, and positioning. CSS3 introduced advanced

features like animations, transitions, and media queries, which enable responsive design for different screen sizes and devices

3. **Java Script:** JavaScript is a programming language used for adding interactivity and functionality to web pages. JavaScript is a versatile programming language that adds interactivity and dynamic behaviour to web pages. It can be used to manipulate HTML and CSS, handle user input, and communicate with servers (Ajax). Modern JavaScript, with the help of libraries and frameworks like React, Angular, and Vue.js, simplifies complex tasks and facilitates the creation of single-page applications (SPAs).
4. **Text Editors and IDEs:** Developers use text editors like Visual Studio Code, Sublime Text, or IDEs like WebStorm to write and manage their HTML, CSS, and JavaScript code. These tools often have features like syntax highlighting, code completion, and debugging.
5. **Version Control Systems (e.g. Git):** Version control systems like Git help developers collaborate on projects, track changes, and manage code versions effectively. Popular platforms like GitHub and GitLab host repositories for easy sharing and collaboration.
6. **Package Monitors (e.g. npm and Yarn):** Package managers simplify the process of installing, updating, and managing third-party libraries and frameworks. npm and Yarn are commonly used for JavaScript projects.
7. **Build Tools (e.g. Webpack and Gulp):** Build tools automate tasks like bundling and minifying JavaScript and CSS files, optimizing images, and managing dependencies. They improve performance and maintainability.
8. **Browser Developer Tools:** Browsers come with built-in developer tools that allow you to inspect and debug HTML, CSS, and JavaScript in real-time, making it easier to troubleshoot issues.
9. **Responsive Design and Testing Tools:** Tools like responsive design simulators and browser extensions help ensure that web pages look and function well on various screen sizes and devices.
10. **Performance Optimisation Tools:** Tools like Lighthouse and PageSpeed Insights analyze web page performance and suggest optimizations for faster loading times.

Front-end development is a dynamic field, and staying up-to-date with the latest technologies, tools, and best practices is essential for creating modern and user-friendly web experiences.

3.4.2 Back-end development tools: Python, PHP, and SQL are commonly used tools and languages in the field of back-end web development.

1. **Python:** Python is a versatile, high-level programming language known for its simplicity and readability. It is often used in web development for building back-end applications using various frameworks and libraries, such as Django, Flask, and FastAPI. Python offers excellent support for handling web requests, managing databases, and implementing business logic. It is known for its strong community.
2. **PHP:** PHP (Hypertext Pre-processor) is a server-side scripting language specifically designed for web development. It is one of the most widely used languages for creating dynamic web pages and web applications. PHP is often used with popular web development frameworks like Laravel, Symfony, and CodeIgniter. It has built-in support for connecting to databases, making it a suitable choice for back-end development when paired with a web server like Apache or Nginx.
3. **SQL (Structured Query Language):** SQL is not a programming language but a domain-specific language used for managing and querying relational databases. In back-end

development, SQL is essential for creating, updating, and querying databases, which store data for web applications. Common relational database management systems (RDBMS) that use SQL include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. Developers use SQL queries to interact with databases, retrieve data, update records, and perform various database-related operations.

When developing the back end of a web application, these tools and languages can be used together to create a robust and functional system. Python or PHP can be used to handle the application's logic, while SQL is used to manage and query the database to store and retrieve data. The choice between Python and PHP often depends on the project requirements, team expertise, and personal preferences of the developers involved.

Assignment 3.1.

- List down the names of commonly used tools and languages in the field of back-end web development.
- List down the names of commonly used technologies used for front-end development.

SUMMARY

- Web development standards and tools are crucial for creating high-quality, reliable, and accessible websites and web applications.
- HTML, CSS, JavaScript, and accessibility standards like WCAG are fundamental to modern web development.
- Front-end tools include text editors, version control systems, and responsive design for creating user interfaces.
- Back-end development tools like Python, PHP, and SQL are used to manage server-side logic and databases.
- Static and dynamic websites serve different purposes, and the choice depends on project goals.
- Coding for websites involves using HTML, CSS, JavaScript, and back-end languages to create functional web applications.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What is the primary function of HTML in web development? (a) Styling web pages (b) Creating server-side logic (c) Defining the structure and content of web pages (d) Adding interactivity to web pages
2. Which of the following CSS versions is commonly used for responsive design? (a) CSS1 (b) CSS2 (c) CSS3 (d) CSS4
3. ECMAScript 6 (ES6) is associated with which web development technology? (a) HTML (b) CSS (c) JavaScript (d) HTTP
4. Which set of guidelines provides standards for creating web content that is accessible to people with disabilities? (a) HTTP/HTTPS (b) WCAG (c) HTML5 (d) CSS3
5. Why is using HTTPS important in web development? (a) It enhances web page styling. (b) It improves website interactivity. (c) It is crucial for data security and SEO. (d) It simplifies content management.
6. What is the standard practice for creating websites that adapt to various screen sizes and devices? (a) Semantic markup (b) Progressive enhancement (c) Accessible design (d) Dynamic content generation
7. Which tool is commonly used for tracking changes, collaborating with others, and managing code versions in web development? (a) npm (b) HTTP (c) Git (d) CSS
8. Which of the following is an example of a front-end JavaScript library or framework? (a) Git (b) MySQL (c) React (d) PHP
9. What is the primary purpose of content management systems (CMS) in web development? (a) Managing databases (b) Creating responsive designs (c) Handling server-side logic (d) Simplifying content creation and management
10. Which tool is commonly used for web application testing and automation? (a) HTML (b) CSS (c) Selenium (d) HTTP

B. Fill in the blanks

1. HTML is the foundation of web development, defining the _____ and content of web pages.
2. CSS3 introduces advanced styling features and is commonly used for _____ design.
3. JavaScript is a programming language used for adding _____ and functionality to web pages.
4. Using HTTPS is essential for _____ and SEO in web development.
5. Responsive design involves creating _____ that adapt to various screen sizes and devices through the use of media queries and responsive design principles, making it a standard practice.
6. Using semantic _____ elements, such as <header>, <nav>, <section>, and <article>, helps convey the meaning and structure of content to both browsers and developers.
7. HTML is the foundation of _____ development.

8. ECMAScript 6 (ES6) is the latest standard for _____.
9. Python is a versatile, _____ programming language known for its simplicity and readability.
10. _____ is essential for creating, updating, and querying databases, which store data for web applications.

C. True/False

1. CSS3 is primarily used for creating static web pages.
2. JavaScript is a programming language that adds interactivity and functionality to web pages.
3. Responsive design principles are not considered a standard practice in modern web development.
4. Using semantic HTML elements does not impact the accessibility of a website.
5. Progressive enhancement involves building web experiences for all users, regardless of their device or browser capabilities.
6. Web development standards and tools play a minimal role in ensuring the quality of websites and web applications.
7. Text editors and IDEs are not commonly used for web development.
8. Version control systems like Git are primarily used for project management, not for tracking changes in code.
9. JavaScript is a versatile programming language that adds interactivity and dynamic behavior to web pages.
10. Package managers like npm are used to install and manage dependencies in web projects.

D. Short Question Answers

1. What is the primary purpose of HTML in web development?
2. Explain the role of CSS in web design and development.
3. What are some of the key principles of responsive design?
4. How do semantic HTML elements contribute to web accessibility?
5. Why is it important to use HTTPS for web security and SEO?
6. What are the benefits of using a content management system (CMS) for web projects?
7. What is the significance of progressive enhancement in web development?
8. Name two popular text editors used by web developers.
9. How do version control systems like Git benefit web development projects?
10. In web development, what is the purpose of a package manager like npm?

Session 4. Application Development Models of IT

Meet Atul, a curious kid who loved using tablets and smartphones. He wanted to know how the cool apps on his devices were made, so he learned about "Application Development Models of IT." He found three models:

Waterfall Model: It's like following a recipe step by step, but you can't go back. It's simple but doesn't allow many changes.

Agile Model: It's like building with blocks, making changes as you go. It's flexible and fun.

Iterative Model: It's like drawing a picture, adding details over time. It's creative.

With these models, Atul created a fun app for his friends to play, and they loved it. Atul learned that making apps was like a fun adventure. He could use different models to create cool things on his gadgets, just like playing with toys, and share them with friends for a great time. As shown in figure 4.1.



Fig. 4.1: Atul creating apps

In this chapter, you will understand about IT application development, Importance of SDLC models in application development, Software Development Life Cycle (SDLC) models, and Waterfall Model, Iterative Model and Agile model.

4.1 IT application development

IT application development models, also known as software development methodologies or frameworks, are structured approaches that guide the process of creating and managing software applications. These models help development teams to streamline their work, improve collaboration, and deliver high-quality software products. There are several established application development models, each with its own principles, practices, and methodologies.

4.1.1 Importance of SDLC models in application development

Software Development Life Cycle (SDLC) models are essential in application development for several key reasons:

1. **Structured Approach:** SDLC models provide a structured and systematic approach to software development. They break down the entire development process into phases and activities, ensuring that each step is well-defined and well-documented. This structured approach helps in better project management and control.
2. **Risk Management:** SDLC models help in identifying and managing risks throughout the development process. By following a predefined set of steps and incorporating risk

assessment at each phase, teams can proactively address issues and mitigate potential problems.

3. **Quality Assurance:** SDLC models emphasize the importance of quality throughout the development lifecycle. Quality checks and testing are integrated into each phase, ensuring that the final product meets the required standards and is free from critical defects.
4. **Clear Communication:** SDLC models facilitate clear communication among team members, stakeholders, and clients. With well-defined phases and milestones, everyone involved in the project has a common understanding of the development progress and can effectively communicate expectations and requirements.
5. **Predictable Timelines and Budgets:** SDLC models help in estimating project timelines and budgets more accurately. By breaking the project into manageable phases and tasks, it becomes easier to estimate the resources and time required for each phase, leading to more realistic project planning.
6. **Scalability:** SDLC models can be adapted to suit the specific needs of different projects. Whether you are working on a small-scale application or a large enterprise-level system, you can choose an appropriate SDLC model or customize one to fit the project's size and complexity.
7. **Documentation and Traceability:** SDLC models encourage the creation of comprehensive documentation at each stage of development. This documentation is invaluable for future reference, maintenance, and troubleshooting. It also ensures that the development process is well-documented for compliance and audit purposes.
8. **Client Satisfaction:** SDLC models prioritize client involvement and feedback throughout the development process. This client-centric approach ensures that the final product aligns with the client's vision and requirements, ultimately leading to higher client satisfaction.
9. **Change Management:** SDLC models provide mechanisms for handling changes and scope creep effectively. Changes are evaluated, documented, and implemented in a controlled manner to minimize disruption and maintain project focus.
10. **Continuous Improvement:** SDLC models promote a culture of continuous improvement. After each project, teams can assess what worked well and what didn't, and use these insights to refine their processes for future projects.

In summary, SDLC models play a crucial role in application development by providing a structured framework for planning, executing, and managing software projects. They ensure that development efforts are efficient, controlled, and result in high-quality software that meets user expectations. Choosing the right SDLC model for a specific project is key to its success.

4.2 Software Development Life Cycle (SDLC) models

Software Development Life Cycle (SDLC) models are frameworks that guide the process of developing software applications or systems. These models define the stages, tasks, and activities involved in the software development process and provide a structured approach to managing and delivering software projects. There are several SDLC models, each with its own set of principles, advantages, and disadvantages. Some of the most commonly used SDLC models:

1. **Waterfall Model:** The Waterfall model is a linear and sequential approach to software development. It divides the project into discrete phases, such as requirements gathering, design, implementation, testing, deployment, and maintenance. Each phase must be

completed before moving on to the next, and changes are difficult to implement once a phase is completed. It is suitable for projects with well-defined and stable requirements.

2. **Prototype Model:** The Prototype Model is a type of software development methodology that emphasizes the creation of prototypes to understand customer requirements better, test ideas, and refine the system before the final product is developed. Unlike traditional SDLC models that usually follow a linear and sequential approach (like the Waterfall model), the Prototype Model is iterative and flexible, allowing developers to make changes more easily based on feedback.
3. **Agile Model:** Agile is an iterative and flexible approach to software development that emphasizes collaboration, customer feedback, and delivering working software incrementally. Agile methodologies include Scrum, Kanban, and Extreme Programming (XP). Agile teams work in short development cycles called sprints or iterations, making it easier to adapt to changing requirements.
4. **Scrum:** Scrum is a specific Agile framework that organizes work into time-boxed iterations called sprints. It emphasizes teamwork, regular inspection, and adaptation. Scrum roles include the Product Owner, Scrum Master, and Development Team, and it relies on artifacts like the product backlog and burndown charts.
5. **Kanban:** Kanban is an Agile methodology focused on visualizing work and managing flow. Work items are represented on a Kanban board, and the emphasis is on limiting work in progress (WIP) to optimize throughput. It's particularly suitable for continuous improvement and managing support or maintenance tasks.
6. **DevOps:** DevOps is a cultural and technical approach that emphasizes collaboration between development and operations teams. It aims to automate and streamline the software delivery pipeline, making it possible to release software faster and more reliably. Continuous integration (CI) and continuous deployment (CD) are common practices in DevOps.
7. **Lean Development:** Lean development is inspired by manufacturing principles and seeks to eliminate waste in the software development process. It emphasizes delivering value to customers, minimizing defects, and optimizing workflow. Lean practices can be integrated into other development models, such as Agile and Kanban.
8. **Spiral Model:** The Spiral model combines iterative development with elements of risk management. It involves multiple cycles of planning, designing, building, and testing, with each cycle representing a "spiral." It's particularly useful for complex projects where risks need to be addressed incrementally.
9. **Rapid Application Development (RAD):** RAD is a model that focuses on quickly building prototypes and iteratively refining them based on user feedback. It's suitable for projects where speed of development is critical and requirements are likely to change.
10. **V-Model (Verification and Validation Model):** The V-Model is an extension of the Waterfall model that emphasizes the relationship between each development phase and its corresponding testing phase. It highlights the importance of verification and validation at each step to ensure that the software meets the specified requirements.
11. **Incremental and Iterative Development:** This approach involves breaking a project into smaller, manageable parts and developing them incrementally. It allows for regular testing and integration of new features, making it easier to adapt to changing requirements.

The software teams often choose these development models based on the nature of the project, organizational constraints, and customer needs. Additionally, hybrid approaches that combine elements of different models are becoming increasingly common to address specific project requirements effectively.

Assignment 4.1.

- List down any five names of SDLC models.
- List down any three key reasons for the Software Development Life Cycle (SDLC) models.

4.3 Waterfall Model

The Waterfall Model is a traditional software development methodology that follows a linear and sequential approach to software development. It was first introduced by Dr. Winston W. Royce in a paper published in 1970. The model is named "Waterfall" because it envisions the software development process as flowing steadily downwards through several phases, like a waterfall as shown in figure 4.2.

The typical phases of the Waterfall Model:

- 1. Requirement Gathering and Analysis:** In this initial phase, the project team works closely with stakeholders to gather and document all the project's requirements. This phase aims to define what the software needs to do and how it should behave.
- 2. System Design:** Once the requirements are established, the system design phase begins. Here, architects and designers create detailed system architecture and design specifications. The focus is on defining the system's structure and components.
- 3. Implementation:** In this phase, developers start coding the software based on the design specifications. Each component or module of the software is developed independently.
- 4. Testing:** After implementation, the software is thoroughly tested to ensure that it meets the specified requirements. Testing includes unit testing, integration testing, system testing, and acceptance testing.
- 5. Deployment:** Once testing is complete and the software is deemed ready, it is deployed to the production environment or made available to users.
- 6. Maintenance and Support:** This phase involves ongoing maintenance, bug fixes, and support for the software throughout its lifecycle.

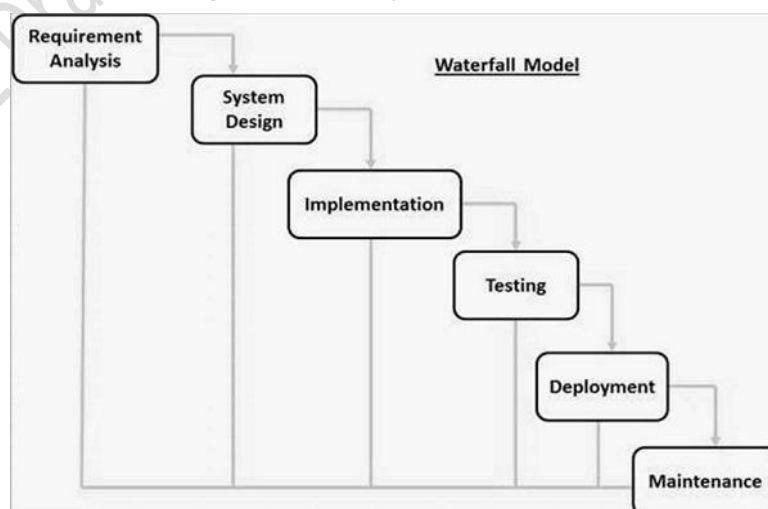


Fig. 4.2: Waterfall Model

The Waterfall Model has several advantages:

1. It provides a structured and well-documented approach.
2. It is easy to manage and understand.
3. It works well for projects with stable and well-defined requirements.

Therefore, it also has significant limitations:

1. It assumes that all requirements are known upfront, which is often not the case.
2. Changes to requirements can be costly and difficult to implement once the project is in the later stages.
3. It does not accommodate flexibility well, making it less suitable for projects where requirements are likely to change.

In practice, the Waterfall Model has been largely replaced by more iterative and flexible development methodologies like Agile, which better accommodate changing requirements and promote collaboration among team members and stakeholders throughout the development process.

4.4 Iterative Model

The term "Iterative Model" can refer to various iterative approaches used in different fields and contexts, including software development, project management, and problem-solving. In general, an iterative model is a process or methodology that involves repeating a series of steps or activities with the goal of refining and improving a product, project, or solution over time. Each iteration typically builds upon the previous one, incorporating feedback and lessons learned to make incremental progress.

One common example of an iterative model is the "Iterative and Incremental Development" approach in software development, which includes methodologies like Agile and Scrum. In this context, the iterative model involves breaking down a project into smaller increments or iterations, with each iteration producing a potentially shippable product increment. Teams work on a subset of features or requirements during each iteration, gather feedback from stakeholders, and then use that feedback to make improvements in the next iteration. This process continues until the project is completed. The Key characteristics of iterative models include:

1. **Repetition:** The process involves repeating a set of steps or activities in cycles or iterations.
2. **Feedback:** Each iteration incorporates feedback from users, customers, or stakeholders to make improvements.
3. **Incremental Progress:** The product or project evolves incrementally with each iteration, adding new features or refining existing ones.
4. **Flexibility:** Iterative models are often more flexible and adaptable to changing requirements or circumstances compared to traditional linear models.
5. **Continuous Improvement:** The focus is on continuous improvement, with each iteration aimed at delivering a better result than the previous one.
6. **Risk Reduction:** By addressing potential issues and risks early in the process, iterative models can help mitigate project risks.

It's worth noting that the specific details of iterative models can vary depending on the field and context in which they are applied. Different industries and domains have adopted iterative approaches to suit their unique needs and challenges.

Overall, the iterative model is a valuable approach when dealing with complex projects, uncertain requirements, or situations where the end goal may not be entirely clear from the outset. It allows for flexibility, adaptability, and the incorporation of feedback to achieve better outcomes.

Assignment 4.2.

- List down the name of phases of the Waterfall Model.
- Draw a waterfall model diagram.
- List down the different characteristics of iterative models.

4.5 Agile Model

The Agile model, often referred to as agile methodology or agile framework, is a set of principles and practices used in software development and project management. It emphasizes flexibility, collaboration, customer feedback, and iterative progress. Agile methodologies are designed to respond to changing requirements and deliver valuable software quickly. There are several agile methodologies, with Scrum and Kanban being two of the most widely used. Agile Model as shown in Figure 4.3.



Fig. 4.3: Agile Model

The some of the key concepts and principles associated with Agile:

1. **Iterative and Incremental Development:** Agile projects are divided into small increments or iterations, usually two to four weeks in length. Each iteration results in a potentially shippable product increment, allowing for continuous improvement.
2. **Customer Collaboration:** Agile teams work closely with customers, stakeholders, and end-users throughout the project to ensure that the product aligns with their needs and expectations.
3. **Flexibility:** Agile prioritizes responding to change over following a rigid plan. It acknowledges that requirements can change, and it adapts to those changes as they occur.
4. **Cross-Functional Teams:** Agile teams are typically cross-functional, meaning they include members with a variety of skills (developers, testers, designers, etc.). This promotes collaboration and shared responsibility for the project's success.
5. **Product Backlog:** In Scrum, one of the most popular Agile methodologies, there is a product backlog—a prioritized list of features and user stories that guide the development team's work.

6. **Sprints:** In Scrum, work is organized into time-bound iterations called sprints. During each sprint, the team commits to completing a set of backlog items.
7. **Daily Standup (Scrum):** Teams hold daily stand-up meetings to discuss progress, challenges, and plans. These meetings are typically brief and help keep the team aligned.
8. **Retrospectives:** At the end of each iteration, Agile teams conduct retrospectives to reflect on what went well, what could be improved, and what changes to make in the next iteration.
9. **Continuous Integration and Testing:** Agile practices often include continuous integration, where code changes are integrated and tested frequently to detect issues early.
10. **Working Software as a Measure of Progress:** Agile prioritizes delivering working software as the primary measure of progress. This contrasts with traditional project management, which may prioritize documentation or meeting milestones.
11. **Minimal Viable Product (MVP):** Agile encourages the development of an MVP, which is the smallest set of features that deliver value to users. This allows for faster delivery of a basic product that can be improved upon iteratively.
12. **Customer Feedback:** Agile teams actively seek feedback from customers and users, incorporating it into the development process to ensure the product meets their needs.

Popular Agile methodologies include:

- **Scrum:** A framework that organizes work into fixed-length sprints, with roles like Scrum Master, Product Owner, and Development Team.
- **Kanban:** A visual method that uses a Kanban board to manage and optimize workflow, focusing on minimizing work in progress.
- **Extreme Programming (XP):** An Agile approach with a strong emphasis on technical excellence, including practices like pair programming and test-driven development.

The Agile model has gained popularity in software development and has also been adapted for use in other industries due to its ability to promote adaptability and customer-centricity.

4.6 DevOps

DevOps is a set of practices, principles, and cultural philosophies aimed at improving collaboration and communication between software development (Dev) and IT operations (Ops) teams. The primary goal of DevOps is to streamline the software delivery process, reduce development cycle times, and enhance the overall quality of software products. It bridges the gap between development and operations by fostering a culture of shared responsibility and continuous improvement. Here are some key aspects of DevOps:

1. **Collaboration:** DevOps encourages cross-functional teams to work closely together throughout the entire software development lifecycle, from planning and coding to testing, deployment, and monitoring. This collaboration helps identify and resolve issues early in the process.
2. **Automation:** Automation is a fundamental principle of DevOps. By automating repetitive tasks such as building, testing, and deployment, teams can reduce errors, increase efficiency, and ensure consistency in their processes.
3. **Continuous Integration:** CI involves automatically integrating code changes from multiple developers into a shared repository. This practice ensures that new code is regularly tested, reducing integration problems and making it easier to catch and fix issues early.

4. **Continuous Delivery:** CD extends CI by automatically deploying code changes to production or staging environments as soon as they pass automated tests. This enables rapid and reliable software releases.
5. **Infrastructure as Code (IaC):** IaC involves defining and provisioning infrastructure resources (servers, databases, networks, etc.) using code. This allows for consistent and repeatable infrastructure deployments and simplifies scaling and management.
6. **Monitoring and Feedback:** DevOps places a strong emphasis on monitoring applications and infrastructure in production. Teams use monitoring tools to gain insights into system performance and user behavior, enabling them to make data-driven improvements.
7. **Microservices and Containers:** DevOps often goes hand-in-hand with the adoption of microservices architecture and containerization technologies like Docker. These technologies make it easier to build, deploy, and manage complex, scalable applications.
8. **Security:** DevOps includes a focus on integrating security practices into the development and deployment process (DevSecOps). This ensures that security is not an afterthought but is considered from the beginning.
9. **Culture:** DevOps is not just about tools and practices; it's also a cultural shift. It promotes a culture of shared responsibility, continuous learning, and a willingness to experiment and iterate.
10. **Feedback Loops:** DevOps teams encourage feedback loops at all stages of the development lifecycle. Feedback helps identify areas for improvement and drives the evolution of processes and tools.

DevOps has become increasingly popular in the software industry as it enables organizations to deliver software more rapidly, reliably, and securely. It breaks down silos between development and operations teams, leading to faster innovation and improved customer experiences. However, implementing DevOps practices effectively requires a commitment to cultural change, skill development, and the right tooling to support automation and collaboration.

SUMMARY

- IT application development models guide software development with structured approaches.
- Software Development Life Cycle (SDLC) models provide structured project management.
- Waterfall is a linear model suitable for stable requirements.
- Agile emphasizes flexibility, customer collaboration, and iterative progress.
- DevOps streamlines software delivery by bridging development and operations.
- Continuous Integration (CI) automates code integration and testing.
- Iterative models allow for incremental progress and adaptation to changing requirements.
- Lean Development focuses on delivering value, minimizing defects, and optimizing workflow.
- Agile encourages customer feedback and collaboration throughout the development process.
- SDLC models play a crucial role in ensuring efficient, controlled, and high-quality software development.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. The Waterfall model is suitable for projects with: (a) Changing and dynamic requirements (b) Stable and well-defined requirements (c) Frequent customer feedback (d) Continuous integration
2. What is the primary focus of DevOps? (a) Automation (b) Collaboration between development and operations (c) Documentation (d) Risk management
3. The Iterative model is suitable for projects with: (a) Uncertain requirements (b) Well-defined and stable requirements (c) No customer involvement (d) No room for change
4. What is the primary goal of DevOps? (a) Enhancing the documentation process (b) Reducing development cycle times (c) Minimizing collaboration between teams (d) Increasing project complexity
5. In Agile, what is a Product Backlog? (a) A list of completed features (b) A prioritized list of features and user stories (c) A list of development team members (d) A list of customer feedback
6. The Spiral model is particularly useful for: (a) Projects with well-defined requirements (b) Projects with minimal risks (c) Complex projects where risks need to be addressed incrementally (d) Rapid application development
7. What is the main emphasis of Lean Development? (a) Maximizing defects (b) Delivering value to customers (c) Slowing down the workflow (d) Frequent changes in requirements
8. In Agile, what is a "Sprint"? (a) A long-distance run (b) A time-boxed development iteration (c) A design phase (d) A client meeting
9. Which SDLC model is known for its emphasis on the relationship between development and testing phases? (a) V-Model (b) Scrum (c) Spiral Model (d) RAD
10. What does the term "MVP" stand for in Agile? (a) Most Valuable Product (b) Maximum Viable Product (c) Minimum Viable Product (d) Major Visual Prototype

B. Fill in the blanks

1. The key characteristic of iterative models is that they involve repeating a set of steps or activities in _____.
2. DevOps bridges the gap between development and operations by fostering a culture of _____ responsibility and continuous improvement.
3. In Agile, teams hold daily stand-up meetings to discuss progress, challenges, and _____.
4. The Spiral model combines iterative development with elements of _____ management and is useful for complex projects.
5. RAD is suitable for projects where speed of development is critical and requirements are likely to _____.
6. In the V-Model, the emphasis is on _____ and validation at each step to ensure that the software meets the specified requirements.
7. Continuous Integration (CI) involves automatically integrating code changes from multiple developers into a shared _____.

8. Kanban is an Agile methodology focused on visualizing work and managing _____ to optimize throughput.
9. The Iterative model allows for flexibility, adaptability, and the incorporation of _____ to achieve better outcomes.
10. DevOps encourages cross-functional teams to work closely together throughout the entire software development _____.

C. True or False

1. SDLC models are frameworks that guide the process of developing hardware applications.
2. The Waterfall model is well-suited for projects with frequently changing requirements.
3. The Agile model emphasizes delivering valuable software incrementally through short development cycles.
4. Scrum is a specific Agile framework that does not include roles like Scrum Master and Product Owner.
5. DevOps focuses on collaboration between development and marketing teams.
6. The Spiral model is best suited for simple and straightforward projects.
7. RAD is suitable for projects where speed of development is not critical.
8. The V-Model highlights the importance of verification and validation at each step of development.
9. Continuous Integration (CI) involves manual integration of code changes.
10. In Kanban, the emphasis is on maximizing work in progress (WIP).

D. Short Question Answers

1. What are IT application development models, and why are they important in software development?
2. What is the main purpose of Software Development Life Cycle (SDLC) models in application development?
3. Name one advantage and one limitation of the Waterfall model in software development.
4. In Agile, what is the significance of customer collaboration during the development process?
5. How does DevOps aim to improve collaboration between development and operations teams?
6. What is the primary focus of Continuous Integration (CI) in DevOps?
7. How do iterative models differ from traditional linear models in software development?
8. Explain the key principle of "working software as a measure of progress" in Agile.
9. Name one Agile methodology other than Scrum and Kanban and briefly describe its characteristics.
10. What is the main goal of Lean Development, and how does it achieve it in the software development process?

Session 5. Web Designing Specifications

Meet Ajay, a kid who loves the internet. He wanted to make his own website, so he learned about "Web Designing Specifications." He found out that it was like creating a treehouse. Here are the simple rules he learned:

Colors: Like picking paint for a treehouse.

Fonts: Choosing how the words look, like a sign on the treehouse.

Images: Decorating the treehouse with cool posters.

Layout: Arranging the furniture in the treehouse.

Links: Creating secret passages in the treehouse.

With these rules, Ajay made an amazing website that people from everywhere loved to visit. Ajay learned that these rules were like a treasure map for creating a fantastic website. Just like decorating a treehouse, he could make an awesome online place for everyone to enjoy. As shown in Figure 5.1.



Fig. 5.1: Ajay designing website

In this chapter, you will understand about Web design specifications, Business Requirement Specifications, Elements of Business requirement document, Benefits of Business Requirements Document, Steps to create Business Requirements Document and Software requirement specification.

5.1 Web design specifications

Web design specifications can vary depending on the specific project and its goals. However, here are some general specifications and guidelines to consider when designing a website:

- 1. Purpose and Goals-** Define the purpose of the website (e.g., informational, e-commerce, portfolio) and Set clear goals for the website (e.g., increasing sales, providing information, building brand awareness).
- 2. Target audience:** Identify the primary audience for the website (e.g., age group, interests, location) and Consider user personas to better understand the audience's needs.
- 3. Design Concept:** Create a design concept that aligns with the brand identity and goals. Choose colour schemes, typography, and visual elements that resonate with the target audience.

- 4. Layout and Navigation:** Decide on the layout of web pages (e.g., single-column, multi-column) and plan a clear and user-friendly navigation structure with intuitive menus and links.
- 5. Content Strategy:** Determine what content will be on the website (e.g., text, images, videos) and develop a content plan and ensure it aligns with the overall design.
- 6. Responsive Design:** Ensure the website is responsive and adapts to various screen sizes and devices (desktop, tablet, mobile).
- 7. Page Load Speed:** Optimize images and code to ensure fast loading times and use content delivery networks (CDNs) to improve performance.
- 8. Accessibility:** Design with accessibility in mind to ensure the website is usable by individuals with disabilities and Follow WCAG (Web Content Accessibility Guidelines) standards.
- 9. Browser Compatibility:** Test the website on multiple web browsers to ensure it works consistently across different platforms.
- 10. Search Engine Optimisation (SEO)** Implement on-page SEO best practices (e.g., meta tags, keyword optimization) to improve search engine rankings.
- 11. Security:** Incorporate security measures to protect against common web vulnerabilities (e.g., SSL certificates, data encryption).
- 12. Content Management System (CMS):** Choose a CMS (e.g., WordPress, Drupal) if necessary and customize it to meet project requirements.
- 13. Testing and Quality Assurance:** Conduct thorough testing to identify and fix any bugs or issues and ensure compatibility with various devices, browsers, and screen sizes.
- 14. Performance monitoring:** Set up tools to monitor website performance and user behaviour (e.g., Google Analytics).
- 15. Backup and recovery:** Establish regular backup procedures and a disaster recovery plan.
- 16. Legal and Compliance:** Ensure compliance with legal requirements (e.g., GDPR for privacy) and intellectual property rights.
- 17. Maintenance and Updates:** Plan for ongoing maintenance, updates, and content management.
- 18. User Training:** If required, provide training for website administrators and content creators.
- 19. Launch Plan:** Develop a launch plan to ensure a smooth transition from development to production.
- 20. Feedback and Iteration:** Collect user feedback after launch and be prepared to make improvements based on user input.

Web design is a dynamic field, and best practices can change over time. Staying updated with the latest trends and technologies is essential for creating modern and effective websites. Additionally, collaborating with web developers, content creators, and other stakeholders is crucial for successful web design projects.

5.2 Business Requirement Specifications (BRS)

A Business Requirements Specification (BRS) is a document that outlines the business needs and objectives of a project or system. It serves as a foundation for the development of a new product, service, or software application. The BRS provides a clear and detailed description of what the business expects to achieve through the project, which helps the development and

implementation process. Some of the key components typically included in a Business Requirements Specification:

1. **Objective:** Begin with a concise statement of the overall goal or objective that the project aims to achieve. This should provide a high-level view of what the business hopes to accomplish.
2. **Scope:** Define the boundaries of the project by specifying what is included and what is not included. This helps in managing expectations and avoiding scope creep.
3. **Stakeholders:** Identify and list all the stakeholders involved in the project, including their roles and responsibilities. This may include business owners, project managers, users, and external partners.
4. **Functional Requirements:** Detail the specific functions and features that the project or system must have to meet the business objectives. This section often includes use cases, user stories, and workflow diagrams.
5. **Non-Functional Requirements:** Describe the quality attributes or constraints that the project must adhere to. This can include performance, security, scalability, and regulatory compliance.
6. **Constraints:** Identify any limitations or constraints that could impact the project, such as budget restrictions, resource limitations, or time constraints.
7. **Assumptions:** Document any assumptions that have been made during the requirement gathering process. This is important for clarity and transparency.
8. **Dependencies:** Highlight any dependencies on other projects, systems, or external factors that could affect the project's success.
9. **Risk and Mitigations:** Identify potential risks that could impact the project and outline strategies for mitigating these risks.
10. **Acceptance Criteria:** Define the criteria that will be used to determine whether the project has been successfully completed. These criteria should be specific, measurable, and achievable.
11. **Prioritisation:** If there are conflicting requirements or limited resources, prioritize the requirements to indicate which ones are most critical for the project's success.
12. **Sign-Off:** Include a section where stakeholders can review and approve the BRS. Sign-off indicates their agreement with the documented requirements.
13. **Change Control:** Describe the process for handling changes to the requirements once the project is underway. This helps prevent scope creep and ensures that changes are properly documented and approved.

Assignment 5.1.

- List down any five web design specifications.
- List down the key components of Business Requirements Specification (BRS).

Creating a well-defined Business Requirements Specification is essential for ensuring that a project aligns with the business's needs and objectives. It serves as a communication tool that helps bridge the gap between business stakeholders and the development team, leading to a more successful project outcome.

5.3 Elements of Business requirement document (BRD)

A Business Requirement Document (BRD) is a critical document in the early stages of a project or initiative, especially in software development or system implementation. It serves as a

comprehensive guide that outlines the objectives, scope, and specifications of a project from a business perspective. The Essential elements typically included in a BRD:

1. Title page:

- Project title
- Date of creation
- Version number
- Author(s)
- Contact information

2. Table of Contents: A list of sections and subsections with page numbers for easy navigation.

3. Executive Summary: A high-level overview of the project, including its purpose, goals, and key deliverables. This section should provide a concise summary of the entire BRD.

4. Project Background: A description of the business problem or opportunity that the project aims to address. Any relevant historical information or context is also included here.

5. Objectives and Goals: Clear and specific statements of what the project intends to achieve with measurable success criteria.

6. Scope and Boundaries: A definition of what is included in the project (in-scope) and what is not (out-of-scope). The constraints or limitations of the project.

7. Stakeholder Analysis: Identification of key stakeholders, including their roles and responsibilities with an overview of their interests and concerns.

8. Functional Requirements: Detailed descriptions of the functionality the system or project must deliver. The Use cases, user stories, or scenarios that illustrate how the system will be used with Business rules and logic.

9. Non-Functional Requirements:

- Performance expectations (e.g., response times, throughput).
- Security requirements.
- Scalability and reliability requirements.
- Compliance and regulatory requirements.

10. Data Requirements:

- Description of data sources.
- Data formats and structures.
- Data storage and handling requirements.

11. User Interface (UI) Requirements:

- Design and layout specifications.
- Navigation and user interaction guidelines.
- Prototypes or wireframes if available.

12. System Interfaces:

- Integration points with other systems or components.
- Communication protocols and data formats.

13. Testing and Quality Assurance:

- Test scenarios, cases, and criteria.

- Quality standards and testing methodologies.

14. Project Timeline: A high-level project schedule or timeline with milestones and dependencies.

15. Budget and Resource Requirements: Estimated project costs, resource allocation, including personnel and equipment.

16. Risk Analysis: Identification of potential risks and mitigation strategies with contingency plans.

17. Change Management and Training: Plans for implementing and managing changes resulting from the project. The training requirements for end-users and support staff.

The Business Requirement Document should be a living document, subject to updates and revisions as the project progresses and as new information becomes available. It plays a crucial role in ensuring that all stakeholders have a shared understanding of the project's objectives and requirements.

5.4 Benefits of Business Requirements Document:

The Business Requirements Document is a crucial document in the field of project management and software development, outlining the requirements and expectations of a project. Some of the benefits of creating and using a BRD:

1. It helps in clearly defining the business objectives and requirements of a project. It ensures that all stakeholders are on the same page regarding the project's purpose and goals.
2. It helps in defining the scope of the project, which is critical for project managers to plan resources, timelines, and budgets effectively.
3. By documenting requirements comprehensively, a BRD helps identify potential risks and challenges early in the project. This enables proactive risk management.
4. The BRD serves as a communication tool between business stakeholders and the project team. It bridges the gap between technical jargon and business language, making it easier for both sides to understand each other's needs and constraints.
5. Throughout the project lifecycle, the BRD serves as a reference point for all decision-making. It can help resolve disputes or disagreements by referring back to the documented requirements.
6. When changes to project requirements arise, the BRD can serve as a baseline against which proposed changes can be evaluated. This helps in maintaining control over scope creep.
7. A well-defined BRD acts as a benchmark for quality assurance and testing efforts. Testers can use it to ensure that the final product meets the specified requirements.
8. With a clear understanding of requirements, project managers can estimate costs more accurately, reducing the risk of cost overruns.
9. The BRD helps in better planning and scheduling of tasks and activities, leading to improved time management and project delivery.
10. Ultimately, by meeting the documented business requirements, the project is more likely to meet customer expectations and achieve a higher level of satisfaction.
11. It provides a structured and documented record of the project's business requirements. This documentation can be useful for audits, compliance, and future reference.
12. A BRD can be a valuable resource for training new team members or stakeholders who join the project after its initiation. It offers a comprehensive overview of the project's goals and requirements.

In summary, a Business Requirements Document (BRD) plays a pivotal role in ensuring project success by facilitating effective communication, managing scope, mitigating risks, and providing a foundation for decision-making and quality assurance throughout the project's lifecycle.

Assignment 5.2.

- List down any five Benefits of Business Requirements Document.
- List down any five essential elements typically included in a BRD.

5.5 Steps to create Business Requirements Document

A Business Requirements Document (BRD) is a formal document that outlines the business requirements for a particular project or initiative. It serves as a foundation for project planning and ensures that everyone involved has a clear understanding of what needs to be accomplished. The steps to create a Business Requirements Document:

1. Begin by clearly stating the purpose of the Business Requirements Document. What problem or opportunity does the project aim to address?
2. Define the scope of the project. What is included and excluded from the project? What are the boundaries?
3. Identify all stakeholders who will be involved or impacted by the project. This includes internal and external parties.
4. Work with stakeholders to gather and document their requirements. These requirements should be specific, measurable, achievable, relevant, and time-bound (SMART).
5. Not all requirements are equally important. Prioritize them based on their criticality to the project's success.
6. Organize the requirements in a structured manner. You can use a table or a spreadsheet to create a matrix that includes the requirement ID, description, priority, source (who provided the requirement), and status.
7. For each high-level requirement, break it down into detailed functional requirements. These should describe what the system or solution needs to do.
8. Non-functional requirements describe the quality attributes of the solution, such as performance, security, scalability, and usability. Document these clearly.
9. Review the BRD with stakeholders to ensure that it accurately represents their needs and expectations. Validate that the requirements are complete, consistent, and feasible.
10. If applicable, create use cases or user stories that illustrate how the system will fulfil specific requirements from the user's perspective.
11. Use diagrams, flowcharts, or other visuals to enhance the understanding of complex requirements or processes.
12. For each requirement, specify the acceptance criteria that must be met for it to be considered successfully implemented.
13. Outline the process for managing changes to the requirements during the project. This helps maintain the document's integrity.
14. Once the BRD is complete and all stakeholders agree, obtain formal sign-off from key stakeholders to indicate their approval.
15. Distribute the approved BRD to all relevant project team members and stakeholders. Ensure that everyone is aware of its contents.

16. The BRD serves as a foundation for project planning, including the creation of a project plan, budget, and timeline. Throughout the project, keep the BRD up to date as requirements change or evolve.

The format and structure of a BRD can vary depending on organizational standards and the specific needs of the project. The key is to ensure that it effectively communicates the business requirements to all relevant parties involved in the project.

5.6 Steps to create User Requirement Specification

A User Requirement Specification (URS) is a document that outlines the functional and non-functional requirements of a system or product from the perspective of the end-user. It serves as a blueprint for developers and stakeholders to understand what the user needs and expects from the system. The Steps to create a URS:

1. Clearly state the purpose of the User Requirement Specification document. Define the boundaries and scope of the project or system to be documented. What is the intended outcome, and what are the primary objectives?
2. Identify all the stakeholders involved in the project. These could include end-users, customers, project managers, developers, quality assurance teams, and regulatory bodies.
3. Collect user requirements through various methods such as interviews, surveys, observations, and feedback from existing systems (if applicable). Ensure that you involve a diverse group of users to capture a wide range of perspectives.
4. Organize the gathered requirements into categories for better clarity. Common categories include functional (what the system should do) and non-functional (performance, security, usability, etc.) requirements.
5. For each requirement, provide the following information:
 - a) A unique identifier or code for easy reference.
 - b) A clear and concise description of the requirement.
 - c) The source of the requirement (e.g., user interviews, regulatory guidelines).
 - d) The priority or importance of the requirement (e.g., critical, high, medium, low).
 - e) Any dependencies on other requirements.
6. Define acceptance criteria for each requirement. These are the conditions that must be met for a requirement to be considered fulfilled. Acceptance criteria make requirements testable and measurable.
7. Share the draft URS document with stakeholders and subject matter experts for review and validation. Ensure that all requirements are accurate, complete, and aligned with the project's goals.
8. If the system involves data storage, processing, or reporting, specify the data requirements and reporting formats in detail.
9. Create a traceability matrix to link each requirement to its source, acceptance criteria, and any associated test cases. This helps ensure that all requirements are properly addressed.
10. Conduct a final review of the URS with all relevant stakeholders. Once everyone agrees, obtain formal approval or sign-off from key stakeholders.
11. Keep track of changes to the URS over time. Maintain a version control system to document revisions and updates.
12. Throughout the project's lifecycle, use the URS as a reference to guide design, development, testing, and validation efforts.

As the project progresses and new insights emerge, be prepared to update the URS to reflect any changes in requirements or priorities. Creating a well-documented URS is crucial for the success of a project, as it helps ensure that the final product or system meets the needs and expectations of its users.

5.7 Software requirement specification (SRS)

A Software Requirement Specification (SRS) is a document that outlines the detailed functional and non-functional requirements of a software system. It serves as a contract between the software development team and the client or stakeholders, ensuring that both parties have a clear understanding of what the software is expected to do.

Did You Know?

The SRS is developed based on the agreement between customer and contractors.

An outline of what an SRS typically includes:

1. Introduction:

Purpose: Describe the purpose of the document and the software project.

Scope: Define the scope of the software, including what it will and will not do.

Document Conventions: Explain any conventions or standards used in the document.

2. Overall Description:

Product Perspective: Describe how the software fits into the larger system or ecosystem.

Product Functions: List and describe the major functions and features of the software.

User Classes and Characteristics: Identify the different types of users and their characteristics.

Operating Environment: Describe the hardware, software, and network environments where the software will operate.

Design and Implementation Constraints: Explain any constraints that may impact the design or implementation.

3. Specific Requirements:

Functional Requirements: Detail the specific functions and capabilities the software must provide.

Non-Functional Requirements: Specify non-functional aspects like performance, reliability, security, usability, and scalability.

External Interface Requirements: Describe how the software will interact with external systems or interfaces.

User Interface Requirements: Explain the design and behaviour of the user interface.

System Features: Describe additional features or components of the software.

4. Performance Requirements:

Response Time: Specify the acceptable response times for various functions.

Throughput: Define the expected processing capacity of the system.

Scalability: Describe how the system should handle increased loads.

5. Design Constraints:

Standards Compliance: Identify any industry or regulatory standards that must be followed.

Hardware Limitations: Specify any hardware limitations or requirements.

Software Limitations: Specify any software or platform constraints.

6. **Quality Attributes:**

Reliability: Describe how the software should behave under normal and exceptional conditions.

Security: Define security requirements and access controls.

Usability: Specify usability and user experience requirements.

Maintainability: Describe how the software should be maintained and updated.

7. **Other Requirements:**

Legal and Regulatory Requirements: Detail any legal or regulatory constraints.

Documentation Requirements: Specify documentation that must accompany the software.

Training Requirements: Describe any training that users or administrators will need.

Include any additional information, such as diagrams, mock-ups, or data models.

Define any technical terms or acronyms used in the document.

Maintain a record of changes made to the SRS over time.

The SRS should be a comprehensive and well-structured document that provides a clear and complete understanding of the software's requirements. It serves as a reference throughout the development process and helps ensure that the final product meets the client's expectations.

5.8 Importance of SRS document

A Software Requirements Specification (SRS) document is a crucial artefact in the software development process. It serves as a blueprint for the entire project and plays a pivotal role in ensuring the successful development of a software system. Some of the key reasons why the SRS document is important:

1. The SRS document acts as a bridge between the client or stakeholders and the development team. It helps in clearly defining and communicating what the software is supposed to do and how it should behave.
2. It defines the scope of the project by outlining the features, functionalities, and constraints of the software system. This helps in avoiding scope creep, where additional requirements are introduced during development.
3. The SRS provides a foundation for estimating the time, effort, and resources required for the project. This is essential for project planning and budgeting.
4. By specifying detailed requirements, the SRS serves as a benchmark for quality assurance and testing. Testers can use it to create test cases and ensure that the software meets the specified criteria.
5. It helps in identifying potential risks and challenges early in the project lifecycle. By documenting requirements and constraints, the SRS enables teams to address these issues proactively.
6. Developers and designers use the SRS as a reference to design and implement the software. It provides them with a clear understanding of the desired functionality and behaviour.
7. If changes are requested during the development process, the SRS can be used as a baseline to evaluate the impact of those changes. This helps in managing change requests effectively.
8. A well-defined SRS document helps in managing client expectations. Clients can review the document to ensure that their requirements are accurately captured, reducing the likelihood of misunderstandings or dissatisfaction.

9. In certain industries, software must adhere to specific legal and regulatory requirements. The SRS can be used to ensure that the software complies with these standards.
10. It serves as a critical piece of documentation throughout the software development lifecycle. It provides a historical record of the project's requirements, which can be valuable for maintenance, updates, and future reference.
11. The SRS document encourages collaboration among various stakeholders, including business analysts, developers, testers, and project managers. It helps in aligning their efforts toward a common goal.
12. By clearly documenting requirements, the SRS reduces ambiguity and minimizes the chances of misunderstandings and misinterpretations, which can lead to costly errors.

In summary, the Software Requirements Specification document is essential for setting the direction of a software project, ensuring that it meets the needs of stakeholders, and facilitating effective communication and collaboration among project teams. It serves as a fundamental document that guides the entire software development process from inception to delivery.

5.9 Guidelines for writing SRS

Writing a Software Requirements Specification (SRS) is a critical step in the software development process. An SRS document serves as a blueprint for the entire project, detailing what the software should do, how it should behave, and what it should not do. The guidelines for writing an effective SRS:

1. **Introduction:** Begin with an overview of the document, including its purpose, scope, and any relevant background information about the project.
2. **Purpose:** Clearly state the purpose of the software and what problem it aims to solve.
3. **Scope:** Define the boundaries of the software. What is included and what is not included in the project? Be specific to avoid misunderstandings later.
4. **Functional Requirements:**
 - a) List all the functions and features that the software should provide. Use clear and concise language.
 - b) Organize requirements into categories or sections for better readability.
 - c) Specify the input and output for each function or feature.
 - d) Include any user roles and their associated permissions if applicable.
5. **Non-Functional Requirements:**
 - a) Address non-functional aspects like performance, security, scalability, and usability.
 - b) Use measurable criteria whenever possible. For example, specify response times, error rates, or security standards that must be met.
6. **User Interfaces:**
 - a) Describe the user interfaces, including mock-ups, wireframes, or design guidelines if available.
 - b) Specify how users will interact with the system, including navigation and user flows.
7. **Data Requirements:**
 - a) Describe the data the system will handle, including data types, data sources, and data storage requirements.
 - b) Specify any data validation and data processing rules.

8. Assumptions and Dependencies:

- a) Clearly state any assumptions made during the requirements gathering process.
- b) Identify any external systems, libraries, or dependencies that the software relies on.

9. Constraints: List any constraints that may impact the development or operation of the software, such as budget, time, or technology limitations.

10. Quality and Testing Requirements: Specify quality assurance and testing criteria. Describe the testing approach, including test cases, scenarios, and acceptance criteria.

11. Documentation Requirements: Specify the documentation that should accompany the software, such as user manuals, system documentation, or API documentation.

12. Change Control: Define a process for handling changes to the requirements during the development process. This should include change request procedures and approval processes.

13. Traceability: Establish traceability between requirements, design documents, and test cases. This helps ensure that all requirements are met in the final product.

14. Review and Approval: Specify who will review and approve the SRS document, including stakeholders and subject matter experts. Before finalizing the SRS, conduct reviews and validation with stakeholders to ensure that all requirements are accurate, complete, and well-defined.

Write the SRS in a clear and concise manner, avoiding ambiguous or overly technical language that may lead to misunderstandings. SRS is a living document that may evolve throughout the project, so it's important to maintain good communication and documentation practices as the software development progresses.

Assignment 5.3.

- List down any five guidelines for writing SRS.
- List down any five key reasons for showing the importance of SRS document.

5.10 Creating SRS document

Creating a Software Requirements Specification (SRS) document is an essential step in software development. The SRS document outlines the functional and non-functional requirements of the software to be developed. It serves as a reference for both the development team and stakeholders, ensuring a clear understanding of what the software is expected to do. Here are the typical sections and information to include in an SRS document:

1. Introduction:

Purpose: Explain why the software is being developed and what problem it aims to solve.

Scope: Define the boundaries of the project, including what is and isn't included.

Document Conventions: Specify any naming conventions or formatting guidelines used in the document.

2. Overall Description:

Product Perspective: Describe how the software fits into the larger system, if applicable.

Product Functions: List and describe the main functions or features of the software.

User Classes and Characteristics: Identify the types of users and their roles.

Operating Requirement: Specify the hardware, software, and network requirements.

Design and Implementation Constraints: List any limitations or constraints, such as technology choices or regulatory requirements.

Assumptions and Dependencies: Document any assumptions made during the requirements gathering process and dependencies on other systems or components.

3. Specific Requirements:

Functional Requirements: Detail the specific functions or features of the software, typically using use cases, user stories, or a functional requirements list.

Non-Functional Requirements: Include performance, security, usability, and other non-functional requirements.

External Interface Requirements: Describe how the software will interact with external systems, including APIs, data formats, and protocols.

User Interface Requirements: Explain the user interface design, including wireframes or mock-ups if available.

System Features: Describe additional system-level features such as error handling, logging, and reporting.

Data Requirements: Specify data storage, data processing, and data flow requirements.

Quality Attributes: Define the quality attributes like reliability, scalability, and maintainability.

Constraints: List any constraints that impact the development, such as budget or time limitations.

Use Cases: Provide detailed use case scenarios that illustrate how the system will be used.

4. **Business Rules:** Document any business rules that the software must adhere to.

5. **Test and Validation Criteria:** Define criteria for testing and validating that the software meets the specified requirements.

6. **Acceptance Criteria:** List criteria that must be met for the software to be accepted by the client or stakeholders.

7. Include a glossary of terms and acronyms used in the document to ensure common understanding. Provide references to any external documents or sources used during the requirements gathering process. Include any supplementary information, such as diagrams, charts, or additional documentation.

When creating an SRS document, it's crucial to involve stakeholders, including end-users and developers, to ensure that the requirements accurately represent their needs and expectations. Additionally, the document should be reviewed and updated as the project progresses to reflect any changes or new insights.

SUMMARY

- Web design specifications encompass various documents like BRS, BRD, and SRS, which define project objectives, scope, and software requirements.
- Designing a website involves considering elements like target audience, responsive design, accessibility, and SEO for effective user engagement.
- Browser compatibility, security, and performance monitoring are essential aspects of web design.
- BRS and BRD serve as communication tools between stakeholders, while SRS outlines software functionality and non-functional requirements.

- Effective SRS writing involves sections like introduction, product description, requirements, constraints, and validation criteria.
- Clear documentation ensures that software projects stay on track, meet user needs, and adhere to industry standards.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What is the primary purpose of an SRS (Software Requirements Specification) document?
(a) To showcase the design concept (b) To outline business requirements (c) To define the web design layout (d) To specify the content strategy
2. Which section of the SRS document describes the hardware and software environment in which the software will operate? (a) Functional Requirements (b) Non-Functional Requirements (c) Operating Environment (d) System Features
3. What does URS stand for in the context of software development? (a) User Requirement Specification (b) Universal Resource Schema (c) User Registration System (d) Uniform Reporting Standard
4. Which section of the Business Requirements Specification (BRS) document identifies potential risks and strategies for mitigating those risks? (a) Functional Requirements (b) Risk and Mitigations (c) Scope and Boundaries (d) Acceptance Criteria
5. What is the main goal of establishing clear goals and purposes for a website in the web design process? (a) To make the website visually appealing (b) To guide the design and functionality decisions (c) To increase website loading speed (d) To choose the perfect color scheme
6. Which section of the Business Requirements Specification (BRS) document outlines the specific functions and features that the project or system must have to meet the business objectives? (a) Constraints (b) Functional Requirements (c) Scope (d) Assumptions
7. In the context of web design, what does "responsive design" refer to? (a) Designing a website with bold and vibrant colors (b) Ensuring the website responds quickly to user interactions (c) Designing a website that adapts to various screen sizes and devices (d) Creating a website with complex animations
8. What does SEO stand for in web design? (a) Search Engine Order (b) Site Enhancement Optimization (c) Security and Encryption Operations (d) Search Engine Optimization
9. Which section of the Business Requirements Specification (BRS) typically outlines the key stakeholders and their roles in the project? (a) Scope (b) Stakeholders (c) Constraints (d) Assumptions
10. What does "WCAG" stand for in the context of web design accessibility? (a) Web Compatibility and Graphics (b) Web Content Accessibility Guidelines (c) Website and Content Accessibility Group (d) Web Coding and Graphics

B. Fill in the blanks

1. An SRS document helps in identifying potential _____ and challenges early in the project.
2. The URS (User Requirement Specification) is a document that outlines the functional and non-functional requirements from the perspective of the_____.
3. The Performance Requirements section of the SRS document defines the acceptable _____ times and processing capacity of the system.
4. A well-defined User Requirement Specification (URS) document serves as a _____ for developers and stakeholders.
5. The _____ Environment section of an SRS document provides a description of the hardware, software, and network environments in which the software will operate.
6. The _____ document outlines the requirements from the perspective of the end-user.
7. Non-functional requirements in an SRS document may include aspects like performance, _____, and usability.
8. The _____ section of the Business Requirements Specification (BRS) document outlines the primary goals and objectives of the project.
9. The _____ requirements describe aspects like performance, security, and scalability in an SRS document.
10. In web design, responsive design ensures that a website adapts to various _____ and _____.

C. True or False

1. An SRS document is not necessary when developing small-scale software applications.
2. The URS document is primarily focused on technical specifications and implementation details.
3. Change control procedures are not required in the SRS document, as changes should be discouraged.
4. The Business Requirements Specification (BRS) document is a living document subject to updates and revisions as the project progresses.
5. Constraints, assumptions, and dependencies are essential elements to consider when defining business requirements.
6. Business Requirement Specification (BRS) document remains static and never undergoes revisions throughout a project's lifecycle.
7. An SRS document should specify the data requirements, including data formats, data sources, and data storage requirements.
8. The primary purpose of responsive design is to make a website attractive with a wide range of colors and animations.
9. "Scope creep" refers to the act of carefully defining the boundaries of a project.
10. User feedback collection and website improvement based on that feedback are not significant aspects of web design.

D. Short Question Answers

1. What is the primary purpose of a Business Requirements Specification (BRS) document in a web design project?
2. Why is it important to identify the target audience when designing a website?
3. How does responsive design contribute to a better user experience on websites?
4. What are some examples of non-functional requirements in a Software Requirements Specification (SRS) document?
5. What role does usability testing play in the web design process?
6. How does a traceability matrix help in managing software requirements?
7. What is the significance of setting "Acceptance Criteria" for requirements in an SRS document?
8. Why is it crucial to conduct compatibility testing for different web browsers in web design?
9. What is "scope creep," and why is it important to prevent it in a software development project?
10. What are some key elements included in the "Overall Description" section of an SRS document?

Session 6. Low-level and High-level Design for Programming

Meet Ankit, a kid who loved computer games. He wanted to learn about "Low-level and High-level Design for Programming." He found out it's like building with LEGO bricks.

Low-level Design: It's about making the tiny LEGO pieces - like how characters move and the game keeps score.

High-level Design: It's like planning the whole LEGO castle - the story, goals, and how everything fits together.

With these designs, Ankit made a fantastic game. It had a great story, and the characters moved just right. Ankit learned that making games is like being a LEGO master. Low-level design is for the small parts, and high-level design is for the big picture. Just like building a LEGO castle, he made amazing games that everyone loved as shown in Figure 6.1.



Fig. 6.1: Ankit designing LEGO castle

In this chapter, you will understand about Low-level Design, High-Level Design, Object-Oriented Design, Database Design, API (Application Programming Interface) design, and Purpose of High-Level Design.

Low-level and high-level design are two critical phases in the software development process, and they serve different purposes. Let's delve into each of them:

6.1 Low-level Design (LLD):

Low-level design refers to the process of converting the high-level design into a more detailed and concrete blueprint for software implementation. It involves specifying how each component/module of the software will function and interact with others. LLD focuses on the internal details of each module, including data structures, algorithms, class structures, and functions. It's concerned with the nitty-gritty technical aspects of the software.

LLD is highly detailed and closer to the actual code. It involves breaking down high-level components into smaller, manageable units. In LLD, developers typically decide on data structures, database schema, coding standards, and design patterns to be used in the implementation phase. UML diagrams like class diagrams, sequence diagrams, and activity diagrams are often used in low-level design. Pseudocode or code snippets may also be used to illustrate the implementation logic.

The primary output of LLD is a comprehensive design document that describes how each component will be implemented. This document serves as a guide for developers during the

coding phase. LLD helps ensure that the software is implemented correctly and efficiently. It minimizes ambiguities and reduces the risk of errors during coding.

Did You Know?

LLD is a step-by-step refinement process that can be used for designing data structures, required software architecture, source code, and ultimately, performance algorithms.

6.2 High- Level Design (HLD): High-level design is the initial phase of design that focuses on defining the overall system architecture and structure. It outlines the major components, their interactions, and the flow of data and control between them. HLD focuses on the system as a whole and abstracts away the finer implementation details. It answers questions like "What should the system do?" and "How should it be organized?"

HLD is less detailed compared to LLD. It deals with the big picture and system-level concepts. During HLD, architects and designers decide on the system's architecture, major modules, interfaces, and technologies to be used. It often involves defining non-functional requirements like scalability, performance, and security.

UML diagrams like use case diagrams, system architecture diagrams, and flowcharts are commonly used in high-level design. Textual descriptions and documentation also play a crucial role. The primary output of HLD is a high-level design document that serves as a roadmap for the development team. It guides the subsequent phases of the software development life cycle. HLD ensures that the system's overall structure aligns with the project's goals and requirements. It helps in making important architectural decisions early in the project, which can save time and resources in the long run.

Did You Know?

Low-level design deals with the technical details of how individual components/modules will be implemented, while high-level design focuses on defining the system's architecture and overall structure.

6.3 Object-Oriented Design:

Object-oriented programming is a popular paradigm in computer science and software development that is based on the concept of "objects." Objects are instances of classes, which are user-defined data types that encapsulate both data (attributes) and the methods (functions) that operate on that data. It is one of the most popular and widely used approaches for designing and organizing software systems. Object-oriented design is a key aspect of object-oriented programming (OOP), which is a programming paradigm that encourages the organization of code into reusable and self-contained objects. Some fundamental concepts and principles of object-oriented design:

Objects: Objects are instances of classes. They represent real-world entities, concepts, or things within the software system. For example, in a banking application, you might have objects like "Account," "Customer," and "Transaction."

Class: A class is a blueprint or template for creating objects. It defines the structure and behaviour of objects. In OOD, classes encapsulate both data (attributes or properties) and behaviour (methods or functions). For instance, a "Car" class might have attributes like "color" and "make," and methods like "start" and "stop."

Encapsulation: Encapsulation is the concept of bundling data (attributes) and methods (behaviour) that operate on that data into a single unit, which is the class. This unit is often

referred to as an "object." Encapsulation helps in hiding the internal details of how an object works, exposing only what is necessary for its interaction with other objects.

Inheritance: Inheritance is a mechanism that allows one class to inherit the properties and behaviours of another class. It promotes code reusability by creating a hierarchy of classes. For example, you can have a base class "Vehicle" and derived classes like "Car" and "Motorcycle" that inherit from it.

Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common base class. This concept enables dynamic method dispatch, where the appropriate method is invoked based on the actual type of the object. Polymorphism is often achieved through method overriding and interfaces/abstract classes.

Abstraction: Abstraction is the process of simplifying complex reality by modelling classes based on their essential properties and behaviours, while ignoring irrelevant details. Abstraction helps in managing the complexity of large software systems by focusing on what's essential.

Association: Association represents the relationship between two or more objects. It can be a simple connection, or it can involve more complex relationships like aggregation (a "whole-part" relationship) or composition (a strong form of aggregation).

UML: UML is a visual language used for modelling and representing object-oriented systems. It includes diagrams like class diagrams, use case diagrams, and sequence diagrams, which help in visualizing and documenting the design of a software system.

Did You Know?

Object-oriented programming provides a structured and modular way to design and organize code. It promotes modularity, reusability, and maintainability, making it a widely useful in the field of software engineering.

Did You Know?

Database design is a collection of processes that help create, implement, and maintain a business's data management systems.

6.4 Database Design

Designing a database for a programming project is a critical step in building a robust and efficient application. The database design process involves defining the structure of your database, including tables, relationships, and constraints.

The key steps and considerations for designing a database for programming:

- 1. Define Requirements:** Start by understanding the requirements of your application. What data needs to be stored? What are the relationships between different pieces of data? What are the expected data access patterns (read-heavy, write-heavy, or balanced)?
- 2. Entity-Relationship Diagram (ER Diagram):** Create an ER Diagram to visually represent the entities (tables) and their relationships in your database. Identify the primary keys (unique identifiers) for each table and the foreign keys that establish relationships.
- 3. Normalisation:** Apply the principles of database normalization to minimize data redundancy and improve data integrity. Break down your data into separate tables to eliminate duplicate information.
- 4. Choose a Database Management System (DBMS):** Select a DBMS that fits your project's needs. Popular choices include MySQL, PostgreSQL, SQLite, Microsoft SQL Server, and NoSQL databases like MongoDB.

5. Table Design:

- a) Design tables based on the entities identified in your ERD.
- b) Define data types for each column (e.g., integer, varchar, date).
- c) Set primary keys for each table, typically using auto-incrementing integers or unique identifiers.
- d) Define foreign keys to establish relationships between tables.

6. Indexing: Identify columns that will be frequently used in search and retrieval operations and create indexes for those columns. Indexes can significantly improve query performance.

7. Constraints: Define constraints to enforce data integrity, such as unique constraints, check constraints, and foreign key constraints.

8. Data Access Patterns: Consider how your application will access data. Optimize your table design and indexing to support the most common and performance-critical queries.

9. Security: Implement security measures to protect your database, including authentication, authorization, and encryption.

10. Data Migration and Seeding: Plan how you will migrate existing data into the database and how you will seed initial data if necessary.

11. Testing and Optimization: Test your database design thoroughly to ensure it meets your application's requirements and performs efficiently. Monitor and optimize database performance as your application scales and evolves.

12. Backup and Recovery: Implement regular backup and recovery procedures to safeguard your data in case of failures or data corruption.

13. Documentation: Document your database schema, including table definitions, relationships, and constraints. This documentation is crucial for future development and maintenance.

14. Scalability: Consider how your database design will scale as your application grows. Choose appropriate hardware and scaling strategies.

15. Maintenance: Plan for ongoing database maintenance tasks, such as schema updates, performance tuning, and data archiving.

16. Version Control: If applicable, use version control systems to manage changes to your database schema over time.

17. Compliance and Regulations: Ensure that your database design complies with any relevant industry regulations or data protection laws.

18. Monitoring and Logging: Implement monitoring and logging to track database performance and diagnose issues.

It is essential to continuously refine and adapt your design as your application evolves and your understanding of its requirements deepens. Collaboration with other team members, such as database administrators and developers, is also crucial to create a successful database design for your programming project.

Did You Know?

Database design is an iterative process.

6.5 API (Application Programming Interface) design

Designing an API (Application Programming Interface) is a critical aspect of building software systems, whether it's for web applications, mobile apps, or other software components. A well-designed API can make it easier for developers to interact with your system, promote consistency, and ensure future scalability.

Did You Know?

APIs are software intermediaries that allow two applications to communicate with each other using requests and responses.

There are some of key principles and best practices for API design:

- 1. Clear Purpose and Scope:** Define the purpose of your API. What problem does it solve or what functionality does it provide? Clearly define the scope of your API to avoid it becoming too large or too complex.
- 2. Consistency:** Maintain consistent naming conventions, URI structures, and request/response formats. Use HTTP status codes and error messages consistently to indicate the outcome of API requests.
- 3. RESTful Principles:** Follow REST (Representational State Transfer) principles whenever possible, using HTTP methods (GET, POST, PUT, DELETE) for CRUD (Create, Read, Update, Delete) operations. Use resource URIs that represent entities and use HTTP verbs to operate on them.
- 4. Versioning:** Include a version number in your API's URI (e.g., **/v1/resource**) to allow for future updates without breaking existing clients. Use proper versioning strategies like URL versioning or header versioning.
- 5. Authentication and Authorisation:** Implement secure authentication and authorization mechanisms, such as OAuth2 or API keys. Clearly document how to authenticate and obtain necessary permissions.
- 6. Request and Response Format:** Use standard data formats like JSON or XML for request and response payloads. Ensure that response payloads are well-structured and contain all necessary information.
- 7. Error Handling:** Define clear and consistent error responses with meaningful error codes and messages. Include error details, such as the field causing the error, to aid developers in debugging.
- 8. Pagination and Filtering:** Implement pagination for large result sets to improve performance and reduce load on the server. Allow filtering and sorting of results where applicable.
- 9. Rate Limiting:** Implement rate limiting to prevent abuse and ensure fair usage of your API. Clearly document rate limits and how to handle rate limit exceeded responses.
- 10. Documentation:** Provide comprehensive and up-to-date API documentation. Tools like Swagger or OpenAPI can help generate interactive documentation. Include usage examples and code samples to assist developers in using your API.
- 11. Testing:** Thoroughly test your API endpoints, including edge cases and error scenarios. Consider using automated testing tools and continuous integration for ongoing testing.

- 12. Security:** Implement security best practices, such as input validation, parameterized queries, and protection against common security vulnerabilities like SQL injection and cross-site scripting (XSS).
- 13. Feedback and Community Engagement:** Encourage feedback from developers using your API and be responsive to their needs. Foster a community around your API, with forums or support channels.

The API design is an iterative process, and it's essential to gather feedback from developers who use your API to continuously improve it. Additionally, keeping up-to-date with industry best practices and evolving standards is crucial for maintaining a successful API.

6.6. Creating low-level design:

Creating a low-level design (LLD) is an essential step in the software development process. It involves taking the high-level design or architectural blueprint of a software system and breaking it down into smaller, more detailed components. These components typically include modules, classes, functions, data structures, and their relationships. The goal of an LLD is to provide a detailed plan for how each component of the system will be implemented. The steps to create a low-level design:

- 1. Understand Requirements:** Before you can create an LLD, you must have a thorough understanding of the system's requirements. This includes both functional and non-functional requirements. You should also have a clear understanding of high-level design and architecture.
- 2. Identify Modules\Components:** Break down the system into smaller modules or components. Each module should have a well-defined responsibility and should encapsulate a specific piece of functionality.
- 3. Define Interfaces:** For each module or component, define the interfaces or APIs that will allow it to communicate with other modules. Specify the input parameters, output, and expected behaviour of these interfaces.
- 4. Design Data Structure:** Define the data structures that will be used by the modules. This includes defining the structure of data objects, databases, and any other data storage mechanisms.
- 5. Algorithms and Logic:** Outline the algorithms and logic that will be used within each module to perform its functions. This includes flowcharts, pseudocode, or detailed descriptions of how the module will accomplish its tasks.
- 6. Error Handling:** Specify how errors and exceptions will be handled within each module. Define error codes, error messages, and the actions that should be taken when errors occur.
- 7. Concurrency and Multithreading:** If your system needs to support concurrency or multithreading, define how this will be handled within each module. Specify synchronization mechanisms and thread-safe data structures.
- 8. Performance Considerations:** Consider performance optimization strategies, such as caching, indexing, or parallel processing, and incorporate them into the design where necessary.
- 9. Security:** Identify potential security risks and vulnerabilities in your design and outline how they will be addressed. This may include encryption, authentication, and access control mechanisms.

- 10. Testing and Validations:** Define how each module will be tested and validated. This includes unit testing, integration testing, and any other testing strategies relevant to your system.
- 11. Documentations:** Document the low-level design thoroughly. This documentation should include design diagrams, code comments, and any other information that will help developers understand and implement the design.
- 12. Reviews and Feedback:** Conduct a design review with your development team to gather feedback and make improvements to the LLD. Ensure that the design aligns with the high-level design and requirements.
- 13. Refinements:** Iterate on the LLD as needed based on feedback and changes in requirements. Keep the design up to date as the project progresses.
- 14. Implementations:** Once the LLD is finalized, developers can begin implementing the system based on the design.

The level of detail required in an LLD can vary depending on the complexity of the project and the preferences of your development team. It's essential to strike a balance between providing enough detail for implementation and avoiding unnecessary complexity.

6.7 Introduction to high-level design (HLD)

High-Level Design (HLD) is a crucial phase in the software development and system architecture process. It is a fundamental step that occurs after the initial requirements gathering and analysis but before the detailed design and implementation. HLD focuses on creating a conceptual blueprint or architectural overview of a system or software application.

6.7.1. Purpose of High-Level Design (HLD):

HLD serves as a bridge between requirements gathering and low-level design (LLD) or implementation. It provides a bird's-eye view of the system, helping stakeholders understand its structure, components, and interactions. It establishes the architectural foundation for the project, making key design decisions.

6.7.2. Key Components of High-Level Design:

- 1. System Architecture:** Determine the overall structure of the system, including the major components, their relationships, and how they work together.
- 2. Data Design:** Define the data model, including databases, data storage, and data flow within the system.
- 3. Interface Design:** Identify the external interfaces and communication protocols (APIs, web services, etc.) the system will use.
- 4. Technology Stack:** Decide on the programming languages, frameworks, and tools that will be used in the development.
- 5. Security Considerations:** Outline high-level security measures and access controls.
- 6. Scalability and Performance:** Consider how the system will handle future growth and ensure optimal performance.
- 7. Error Handling and Exception Management:** Define how the system will handle errors and exceptions gracefully.

6.7.3. Key Activities in High-Level Design:

- 1. Requirement Analysis:** Review and refine the project requirements, ensuring that they align with the proposed architecture.

2. **System Decomposition:** Break down the system into manageable modules or components, defining their responsibilities.
3. **Data Modelling:** Create entity-relationship diagrams or similar models to represent the data structure and relationships.
4. **Interface Design:** Specify the APIs and external communication methods that the system will use to interact with other systems or users.
5. **Technology Selection:** Choose the appropriate technologies, libraries, and tools that best fit the project's needs.
6. **Prototyping:** In some cases, create prototypes or mock-ups to validate the design concept before moving to detailed implementation.

6.7.4. Deliverables of High-Level Design:

1. **Architectural Diagrams:** Visual representations of the system's architecture, such as block diagrams, flowcharts, or UML diagrams.
2. **Data Models:** Entity-relationship diagrams (ERDs) or data flow diagrams (DFDs) to illustrate data structures and flows.
3. **Interface Specifications:** Detailed documentation of external interfaces, APIs, and communication protocols.
4. **Technology Stack Documentation:** A list of technologies, frameworks, and tools chosen for the project, with justifications.
5. **High Level Security Plan:** An overview of the security measures and access controls planned for the system.
6. **Performance and Scalability Considerations:** Documentation on how the system will handle growth and maintain optimal performance.

High-Level Design is a critical phase in software development that establishes the architectural foundation of a project. It defines the system's structure, data flow, interfaces, and technology choices, providing a clear roadmap for the subsequent phases of detailed design and implementation. Effective HLD ensures that the final system meets the requirements and functions efficiently while allowing for future scalability and maintainability.

Assignment 6.1.

- List down the Key Activities in High- Level Design.
- List down any five steps to create a low-level design.
- List down any five key principles for API design.

SUMMARY

- Low-level design (LLD) involves detailed planning for software implementation.
- High-level design (HLD) focuses on system architecture and structure.
- Object-oriented design (OOD) uses objects, classes, and principles for organizing code.
- Database design involves steps like requirements analysis, ER diagrams, and data normalization.
- API design should have a clear purpose, consistency, RESTful practices, and documentation.
- High-level design serves as an architectural overview with components and technology selection.

Check Your Progress

A. Multiple-Choice Questions (MCQs):

1. What is the primary focus of Low-Level Design (LLD)? (a) High-level architecture (b) Internal details of software components (c) System requirements (d) Project documentation
2. Which of the following UML diagrams is typically used in High-Level Design (HLD)? (a) Sequence diagram (b) Class diagram (c) Activity diagram (d) ER diagram
3. In Object-Oriented Design (OOD), what are objects? (a) Instances of classes (b) Data attributes (c) Methods (d) Variables
4. What is encapsulation in Object-Oriented Design? (a) Hiding data from other objects (b) Combining multiple classes into one (c) Using inheritance to derive new classes (d) Applying polymorphism to methods
5. Which step of Database Design involves minimizing data redundancy? (a) Entity-Relationship Diagram (b) Normalization (c) Table Design (d) Security
6. What is the purpose of rate limiting in API design? (a) Ensuring secure authentication (b) Preventing abuse and ensuring fair usage (c) Optimizing database performance (d) Defining request and response formats
7. Which step of creating low-level design involves defining error codes and messages? (a) Identifying modules/components (b) Defining interfaces (c) Error Handling (d) Documentations
8. High-Level Design (HLD) serves as a bridge between which phases of software development? (a) Requirement analysis and prototyping (b) Detailed design and implementation (c) Low-Level Design (LLD) and testing (d) System architecture and data design
9. What is a key benefit of implementing version control for a database design? (a) Ensuring security (b) Handling rate limits (c) Managing changes to the schema over time (d) Testing and validation
10. Which aspect does High-Level Design (HLD) primarily focus on? (a) Technical implementation details (b) Architectural overview (c) Low-level components (d) Algorithm development

B. Fill in the Blanks

1. High-Level Design (HLD) provides a _____ view of the system's structure and components.
2. Objects in Object-Oriented Design (OOD) encapsulate both data (attributes) and _____.
3. Database normalization involves minimizing data _____ and improving data integrity.
4. In API design, implementing rate limiting helps prevent _____ and ensures fair usage.
5. Creating _____ design involves defining error codes, error messages, and actions.
6. High-Level Design serves as a bridge between _____ gathering and implementation.
7. Rate limiting is implemented in API design to prevent abuse and ensure _____.

8. The primary output of Low-Level Design (LLD) is a comprehensive _____ document.
9. High-Level Design defines the system's architecture, _____, components, and technology choices.
10. Versioning in API design allows for future updates without _____ existing clients.

C. True or False

1. Low-Level Design (LLD) focuses on the nitty-gritty technical aspects of the software.
2. High-Level Design (HLD) is less detailed compared to LLD.
3. Objects in Object-Oriented Design (OOD) encapsulate only data, not methods.
4. Normalization in Database Design involves breaking down data into separate tables.
5. Rate limiting is not relevant in API design.
6. In Object-Oriented Design, inheritance allows one class to inherit the properties and behaviours of another class.
7. High-Level Design includes detailed UML diagrams like class diagrams and sequence diagrams.
8. Database design is an iterative process.
9. An ERD is used to visually represent the entities and their relationships in database design.
10. High-Level Design focuses on the technical implementation details of the system.

D. Short Question Answers

1. What is the primary purpose of Low-Level Design (LLD) in software development?
2. How does High-Level Design (HLD) differ from Low-Level Design (LLD) in terms of focus and detail?
3. Explain the concept of encapsulation in Object-Oriented Design (OOD).
4. What is the significance of database normalization in the Database Design process?
5. Why is rate limiting important in API design, and how does it benefit applications?
6. In Object-Oriented Design (OOD), how does inheritance promote code reusability?
7. How does polymorphism enable dynamic method dispatch in Object-Oriented Design?
8. Describe the role of abstraction in simplifying complex reality in software design.
9. What are the primary outputs of Low-Level Design (LLD) in the software development process?
10. How does version control contribute to the effective management of a database design over time?

Session 7. Test and identify Design Defects

Meet Ankit, a kid who loved creating things like LEGO houses and drawings. But sometimes, there were mistakes. So, he learned about "Test and Identify Design Defects." He found out it's like being a detective:

Testing: It's like making sure your LEGO house stands up and your drawing is nice.

Checking: It's like looking closely to find any mistakes or smudges.

Finding Mistakes: It's like spotting missing pieces in your LEGO set and figuring out how to fix them.

Ankit used these tricks when creating. It made his things better, just like a super detective solving a fun puzzle. He realized that finding and fixing mistakes was part of the exciting adventure of creating as shown in Figure 7.1.



Fig 7.1: Ankit identifying design defects

Did You Know?

A design defect is a flaw in a product's design that makes it unreasonably dangerous to the user.

7.1 Design Defects

Design defects in products or systems can lead to various issues, including safety hazards, inefficiencies, and customer dissatisfaction. These defects can have a wide range of causes, and understanding both the defects and their underlying causes is essential for preventing them.

There are some of the common design defects and their potential causes:

1. **Insufficient Safety Measures:** Lack of safety features or safeguards in products or systems can lead to accidents and injuries.
2. **Poor Ergonomics:** Products that are uncomfortable or challenging to use due to inadequate ergonomics can lead to user discomfort and decreased productivity.
3. **Inefficient Functionality:** Products that do not perform their intended functions efficiently or effectively can lead to customer dissatisfaction and wasted resources.
4. **Capability Issues:** Incompatibility with other systems or components can result in integration problems and reduced functionality.
5. **Inadequate Durability:** Products that wear out quickly or are not built to withstand expected usage conditions can lead to frequent replacements and increased costs.

6. **Complexity:** Overly complex designs can confuse users and increase the likelihood of errors and maintenance challenges.
7. **Lack of User-Friendly Interfaces:** Poorly designed user interfaces can frustrate users and hinder the usability of software applications or devices.
8. **Cost Overruns:** Designs that exceed budgetary constraints can lead to financial issues and project delays.
9. **Environmental Impact:** Designs that have a negative impact on the environment, such as excessive energy consumption or the use of harmful materials, can result in sustainability concerns.

7.2 Types of Defects

There are Following are some of the basic types of defects in the software development:

1. **Arithmetic Defects:** It include the defects made by the developer in some arithmetic expression or mistake in finding solution of such arithmetic expression. This type of defect is basically made by the programmer due to access work or less knowledge. Code congestion may also lead to arithmetic defects as programmers are unable to properly watch the written code.
2. **Logical Defects:** Logical defects are mistakes done regarding the implementation of the code. When the programmer doesn't understand the problem clearly or thinks in a wrong way then such types of defects happen. Also, while implementing the code if the programmer doesn't take care of the corner cases then logical defects happen. It is basically related to the core of the software.
3. **Syntax Defects:** Syntax defects means mistake in the writing style of the code. It also focuses on the small mistake made by the developer while writing the code. Often the developers do the syntax defects as there might be some small symbols escaped. For example, while writing a code in C++ there is a possibility that a semicolon (;) is escaped.
4. **Multithreading Defects:** Multithreading means running or executing the multiple tasks at the same time. Hence in the multithreading process there is the possibility of complex debugging. In multithreading processes sometimes, there is a deadlock and starvationis created that may lead to the system's failure.
5. **Interface Defects:** Interface defects means the defects in the interaction of the software and the users. The system may suffer different kinds of interface testing in the forms of the complicated interface, unclear interface or the platform-based interface.
6. **Performance Defects:** Performance defects are the defects when the system or the software application is unable to meet the desired and the expected results. When the system or the software application doesn't fulfil the user's requirements then that is the performance defect. It also includes the response of the system with the varying load on the system.
7. **Software Error:** A software error occurs during the development of the software. This error can be in the form of a grammatical error, a logical error where the outcome of a sequence of executions will not result in what was intended or a misinterpretation of the user requirements in the actual written code. It may be in the form of user documentation not matching the software applications operation. An error may or may not be detected during the coding or testing of the program before it is released to a customer.
8. **Software Fault:** A software fault occurs as a result of an error that remains in the executing program. Not all faults however are detected and the software may continue executing

without any obvious problems. There are cases where software faults go undetected for many years of a program's existence.

9. **Software Failure:** A software failure is a fault that results in a detectable problem; hence it is referred to as a failure. A failure would cause the application to malfunction in an obvious manner that warrants the attention of system maintenance.

7.3 Common Causes of Design Defects

1. **Inadequate Requirements Analysis:** Failing to thoroughly understand and define user requirements can result in designs that do not meet user needs.
2. **Insufficient Testing and Validation:** Skipping or inadequately conducting testing and validation processes can allow defects to go unnoticed until after the product is in use.
3. **Rushed Development Timelines:** Pressure to meet tight deadlines can lead to shortcuts in the design process, resulting in overlooked issues.
4. **Lack of Cross-Functional Collaboration:** Failure to involve relevant stakeholders and experts from various disciplines can result in oversight of critical design aspects.
5. **Inadequate Research:** Lack of market research or technological research can result in designs that are out of touch with current trends or user preferences.
6. **Design Changes:** Frequent changes in design specifications can lead to confusion and errors in the final product.
7. **Inadequate Testing Environments:** Testing in environments that do not accurately simulate real-world conditions can lead to missed defects.
8. **Inadequate Training and Skill Sets:** Design teams lacking the necessary skills or knowledge can produce subpar designs.
9. **Lack of Prototyping:** Skipping the prototyping phase can result in designs that have not been adequately refined and tested.

To mitigate design defects, organizations should prioritize thorough requirements analysis, comprehensive testing, and ongoing collaboration among all stakeholders. Additionally, continuous improvement processes and feedback loops can help identify and rectify design defects as early as possible in the development cycle.

7.4 Stages of defect in software development life cycle

Defects, also known as bugs or issues, can occur at various stages of the software development life cycle (SDLC). Identifying and addressing defects early in the process is crucial for delivering a high-quality software product. Here are the typical stages where defects can be introduced and detected in the SDLC:

1. **Requirements Gathering and Analysis:**

Introduction of Defects-Misunderstanding or miscommunication of requirements can lead to defects.

Detection: Requirements reviews and validation with stakeholders can help uncover issues.

2. **System Design:**

Introduction of Defects-Incomplete or incorrect design specifications can introduce defects.

Detection: Peer reviews and design walkthroughs can catch design-related defects.

3. **Coding/Implementation:**

Introduction of Defects: Errors in coding, such as logical errors, syntax errors, and typos, can be introduced during this phase.

Detection: Code reviews, static code analysis tools, and automated testing can help identify coding defects.

4. **Unit Testing:**

Introduction of Defects: Defects in individual units or components of code can go unnoticed.

Detection: Developers typically perform unit testing, and automated unit tests are executed to find defects within isolated code segments.

5. **Integration Testing:**

Introduction of Defects: Integration issues, such as incorrect data exchanges between components, can lead to defects.

Detection: Integration testing focuses on identifying defects that arise when different components interact.

6. **System Testing**

Introduction of Defects: System-level defects, like functional and performance issues, can emerge

Detection: System testing exercises the entire software system to uncover defects related to overall functionality and performance.

7. **User Acceptance Testing**

Introduction of Defects: Defects that impact end-users may surface during UAT.

Detection: Users or stakeholders conduct UAT to identify defects from a user perspective.

8. **Regression Testing**

Introduction of Defects: New code changes or fixes can inadvertently introduce new defects or re-introduce old ones.

Detection: Automated regression testing ensures that existing functionality is not compromised by recent changes.

9. **Deployment of Release**

Introduction of Defects: Configuration issues, deployment errors, or compatibility problems can occur during deployment.

Detection: Careful deployment planning, rollback procedures, and monitoring are essential to catch defects during this phase.

10. **Maintenance and Support**

Introduction of Defects: Changes or updates made to a live system may introduce defects.

Detection: Ongoing monitoring, customer feedback, and continuous testing help detect and address defects in the post-release phase.

It's important to note that while these stages represent a typical SDLC, different development methodologies (e.g., Agile, Waterfall, DevOps) may involve variations in the order and frequency of these stages. Regardless of the methodology, a robust quality assurance process is crucial for identifying and addressing defects at every stage of development.

7.5 Types of Software Testing

Software testing is a crucial process in software development that helps identify defects or issues in a software application. There are various types of software testing methods and techniques, each designed to uncover different types of defects. Here are some common types of software testing and the defects they aim to uncover:

1. **Unit Testing:** Unit testing is focused on testing individual components or units of code in isolation. It helps identify defects within a specific code module, such as incorrect logic or syntax errors.
2. **Integration Testing:** Integration testing checks how different components/modules of an application work together when integrated. It helps uncover defects related to the interaction between these components.
3. **Functional Testing:** Functional testing verifies that the software functions according to its specifications. It aims to uncover defects related to incorrect behaviour, missing functionality, or usability issues.
4. **Regression Testing:** Regression testing ensures that new code changes do not introduce defects or break existing functionality. It helps identify defects that might have been reintroduced unintentionally.
5. **User Interface (UI) Testing:** UI testing checks the graphical user interface (GUI) of the software. It looks for defects in the layout, appearance, and functionality of the user interface.
6. **Performance Testing:** Performance testing assesses the software's speed, responsiveness, and scalability. It helps uncover defects related to slow performance, bottlenecks, or resource utilization issues.
7. **Stress Testing:** Stress testing pushes the software beyond its expected limits to find defects related to system failure, memory leaks, or crashes under extreme conditions.
8. **Security Testing:** Security testing focuses on identifying vulnerabilities and weaknesses in the software's security mechanisms. It helps uncover defects related to data breaches or unauthorized access.
9. **Usability Testing:** Usability testing assesses the software's user-friendliness and how easy it is for users to accomplish their tasks. It helps uncover defects related to poor user experience.
10. **Compatibility Testing:** Compatibility testing checks how the software performs on different devices, browsers, or operating systems. It helps uncover defects related to platform-specific issues.
11. **Accessibility Testing:** Accessibility testing ensures that the software is accessible to users with disabilities. It helps uncover defects related to compliance with accessibility standards.
12. **Localization Testing:** Localization testing verifies that the software is adapted for different languages and cultures. It helps uncover defects related to translation, cultural sensitivity, and formatting.
13. **Alpha and beta Testing:** Alpha and beta testing involve real users testing the software in a controlled environment (alpha) or a broader audience (beta). They help uncover defects that might not be apparent during internal testing.

Each of these testing types serves a unique purpose in the software development lifecycle and helps in identifying defects and ensuring the quality and reliability of the software product. The choice of which testing types to use depends on the project's requirements, goals, and constraints.

Assignment 7.1.

- List down the types of Software Testing.
- List down the common causes of design defects.
- List down the types of defects in the software development.

SUMMARY

- Design defects can lead to safety hazards, inefficiencies, and customer dissatisfaction.
- Common design defects include insufficient safety measures, poor ergonomics, and complexity.
- Types of software defects encompass arithmetic, logical, syntax, and multithreading defects.
- Inadequate requirements analysis and rushed development timelines are common causes of design defects.
- Defects can occur at various stages of the software development life cycle.
- Software testing types include unit, integration, performance, and usability testing.
- Compatibility, security, and accessibility testing are crucial for defect identification.
- Alpha and beta testing involve real users in a controlled and broader environment, respectively.
- Identifying and addressing defects early in the software development process is essential for delivering high-quality software products.

Check Your Progress**A. Multiple-Choice Questions (MCQs)**

1. Design defects can lead to various issues, including safety hazards and inefficiencies. What is one common design defect mentioned in the chapter? (a) High costs (b) User satisfaction (c) Compatibility (d) Sustainability
2. Which type of defects are related to mistakes in arithmetic expressions and are often caused by code congestion? (a) Arithmetic Defects (b) Logical Defects (c) Syntax Defects (d) Interface Defects
3. What type of defect can occur when the programmer doesn't understand the problem clearly or thinks incorrectly? (a) Arithmetic Defects (b) Logical Defects (c) Syntax Defects (d) Multithreading Defects
4. Which stage of the software development life cycle (SDLC) introduces defects related to coding errors and typos? (a) Requirements Gathering and Analysis (b) System Design (c) Coding/Implementation (d) Integration Testing
5. What type of testing focuses on uncovering defects related to the interaction between different components/modules of an application? (a) Unit Testing (b) Integration Testing (c) Functional Testing (d) Security Testing
6. Which type of testing checks the graphical user interface (GUI) of the software for defects in layout, appearance, and functionality? (a) Regression Testing (b) Usability Testing (c) Security Testing (d) UI Testing

7. Defects introduced by new code changes or fixes that inadvertently reintroduce old issues are identified through: (a) Unit Testing (b) Integration Testing (c) Functional Testing (d) Regression Testing
8. What does usability testing assess about software? (a) Speed and responsiveness (b) Security vulnerabilities (c) User-friendliness and ease of use (d) Platform-specific issues
9. Security testing aims to identify: (a) Arithmetic Defects (b) System failures (c) Data breaches and vulnerabilities (d) Interface defects
10. What type of defect remains in the executing program but is not detected and may not cause immediate problems? (a) Software Error (b) Software Fault (c) Software Failure (d) Syntax Defect

B. Fill in the blanks

1. _____ are defects made by the developer in arithmetic expressions or mistakes in finding solutions.
2. Logical defects occur when the programmer doesn't understand the problem clearly or thinks in a _____ way.
3. Syntax defects are related to mistakes in the writing _____ of the code.
4. Inadequate requirements analysis can result in designs that do not meet user _____.
5. Integration testing focuses on identifying defects that arise when different _____ interact.
6. Security testing helps uncover defects related to data breaches and _____ access.
7. Defects introduced by new code changes are identified through _____ testing.
8. Performance testing assesses the software's speed, responsiveness, and _____.
9. Compatibility testing checks how the software performs on different devices, _____, or operating systems.
10. The _____ of defects is crucial for delivering a high-quality software product.

C. True or False

1. Design defects can lead to various issues, including safety hazards, but they don't affect user satisfaction.
2. Logical defects occur when the programmer thoroughly understands the problem.
3. Usability testing focuses on assessing the software's performance and scalability.
4. Security testing aims to identify vulnerabilities and weaknesses in the software's security mechanisms.
5. Stress testing is a type of performance testing.
6. Interface defects are related to defects in arithmetic expressions.
7. Alpha and beta testing involve real users testing the software during the development phase.
8. Inadequate requirements analysis can lead to designs that perfectly meet user needs.
9. Compatibility testing checks how the software performs on different devices, browsers, or operating systems.
10. Defects introduced by new code changes are always immediately detected and resolved.

D. Short Question Answers

1. What are some potential consequences of design defects in products or systems?
2. Name one common design defect related to user discomfort.
3. How do logical defects differ from other types of software defects?
4. What is the impact of inadequate requirements analysis on design defects?
5. At which stage of the software development life cycle can defects be introduced during requirements gathering?
6. What is the purpose of integration testing, and what types of defects does it aim to uncover?
7. Explain the role of performance testing in identifying software defects.
8. Why is security testing important in defect identification?
9. What types of defects can usability testing help uncover?
10. Differentiate between alpha and beta testing and their significance in defect identification.

PSSCIVE Draft Study Material © Not to be Published

Session 8. Resolve Design Defects

Meet Ankit, a kid who loved building with LEGO and drawing pictures. But sometimes, there were small mistakes. So, he learned about "Resolve Design Defects." Ankit discovered it was like being a problem solver:

Spot the Mistake: Like finding a missing piece in a puzzle, he looked closely to see what was wrong.

Think of a Solution: Just like figuring out how to complete a puzzle, Ankit used his imagination to come up with ways to fix the mistake.

Fix the Mistake: Similar to placing the missing puzzle piece, he made changes to make his creations look better.

By following these steps, Ankit improved his LEGO creations and drawings as shown in Figure 8.1. It was like being a superhero for his creations, making them even more fantastic!



Fig 8.1: Ankit resolving design defects

In this chapter, you will understand how to Resolve design Defects, Scope of improvement for future design, Coding tools, and Recording and documentation of Design Defects.

8.1 Resolve design Defects

Resolving design defects is a crucial part of the product development process. These defects can lead to issues with functionality, usability, and even safety. Here are some steps you can take to resolve design defects effectively:

1. **Identify Defects:** The first step is to identify and document all design defects. This can be done through various means, including usability testing, code reviews, and customer feedback. Ensure that you have a comprehensive list of defects.
2. **Prioritize Defects:** Not all defects are equal in terms of their impact on the product. Prioritize them based on severity and potential consequences. Critical defects that affect safety or security should be addressed immediately, while less critical defects can be scheduled for later.
3. **Root Cause Analysis:** For each defect, conduct a root cause analysis to understand why it occurred. This involves tracing the defect back to its source, whether it's a coding error, a miscommunication in the design phase, or some other issue. This step is essential for preventing similar defects in the future.

4. **Involve Cross-Functional Teams:** Resolving design defects often requires input and collaboration from various teams, including design, development, quality assurance, and product management. Ensure that all relevant stakeholders are involved in the process.
5. **Design Changes:** If the defect is related to the design itself, work with the design team to make necessary revisions. This may involve creating new design mock-ups, wireframes, or prototypes to address the issues.
6. **Code Fixes:** If the defects are related to the code, developers should fix the issues following coding best practices. This might involve debugging, refactoring, or rewriting parts of the code.
7. **Testing:** After making design and code changes, conduct thorough testing to ensure that the defects have been resolved. This includes unit testing, integration testing, and user acceptance testing.
8. **Regression Testing:** Be aware that fixing one defect can sometimes introduce new defects or break existing functionality. Therefore, perform regression testing to ensure that other parts of the product remain unaffected.
9. **Documentation:** Update documentation such as user manuals, design specifications, and code comments to reflect the changes made to resolve the defects.
10. **User Feedback:** If possible, involve end-users or beta testers in the validation process. Gather feedback from them to ensure that the defect resolution has met their expectations and hasn't introduced new usability issues.
11. **Communication:** Keep all stakeholders informed about the progress of defect resolution. Transparency and clear communication are essential to manage expectations and maintain trust.
12. **Monitoring:** Even after resolving defects, continue to monitor the product in production. Sometimes, issues may only become apparent in a real-world environment, and it's important to address them promptly.
13. **Continuous Improvement:** Use the experience gained from resolving defects to improve your design and development processes. Implement preventive measures to reduce the likelihood of similar defects occurring in the future.

Remember that the resolution of design defects is an iterative process, and it may require multiple cycles of identification, analysis, and correction. Regularly reviewing and improving your development and design processes can help minimize the occurrence of defects in the first place.

8.2 Scope of improvement for future design

The scope of improvement for future design is vast and ever-evolving, as design disciplines continually adapt to new technologies, trends, and user needs. Some of the key areas where designers can focus on improving their work in the future:

1. **User Centered Design:** Prioritizing user needs and preferences is fundamental to design. Conduct thorough user research, usability testing, and user feedback analysis to create designs that truly resonate with your target audience.
2. **Accessibility:** Designing with accessibility in mind is crucial. Ensure that your designs are inclusive and comply with accessibility standards (e.g., WCAG) to make them usable by people with disabilities.

3. **Mobile and Responsive Design:** As mobile device usage continues to rise, designers should excel in creating responsive designs that work seamlessly on various screen sizes and devices.
4. **Data Driven Design:** Leverage data analytics and user behaviour insights to make informed design decisions. A/B testing and user data can help refine and optimize designs over time.
5. **Sustainability:** Consider the environmental impact of design choices. Sustainable design focuses on reducing waste, using eco-friendly materials, and creating products that have a minimal carbon footprint.
6. **Human centred AI Design:** With the growing use of AI and machine learning, designers should focus on creating AI-driven interfaces that are intuitive and enhance user experiences rather than replacing them.
7. **Ethical design:** Ensure that your designs respect user privacy and adhere to ethical principles. Avoid dark patterns and deceptive practices that manipulate users.
8. **Collaboration and Adaptability:** Collaborate effectively with cross-functional teams, including developers, marketers, and product managers. Designers who can communicate and work well with others often produce more successful products. Be prepared to adapt to new technologies and design trends. Staying up-to-date with the latest tools and techniques is essential.
9. **Continuous Learning:** Design is a constantly evolving field. Commit to lifelong learning by attending workshops, conferences, and staying updated with design blogs and publications. The scope of improvement in design is limitless, and it's essential to stay curious, open to change, and willing to experiment with new ideas and technologies to push the boundaries of design excellence in the future.

8.3 Coding tools

Coding tools are software programs and utilities that help developers write, debug, and manage code more efficiently. These tools are essential for programmers to streamline their workflow and produce high-quality software. There are many coding tools available, each serving a specific purpose. Here are some common categories of coding tools:

1. Integrated Development Environments (IDEs):

Visual Studio Code: A highly customizable, free, and open-source code editor developed by Microsoft. It supports various programming languages and extensions.

PyCharm: An IDE specifically designed for Python development, offering features like code analysis, debugging, and version control.

2. Code Editors:

Sublime Text: A lightweight yet powerful text editor known for its speed and extensibility.

Atom: An open-source, hackable text editor developed by GitHub with a strong emphasis on customization and community-driven packages.

3. Version Control Systems (VCS):

Git: A distributed version control system widely used for tracking changes in source code during software development.

GitHub: A web-based platform for hosting and collaborating on Git repositories.

4. Code Collaboration and Sharing:

GitHub/GitLab/Bitbucket: Platforms for hosting and collaborating on code repositories, managing issues, and facilitating code reviews.

Slack: A team communication tool that can integrate with coding platforms to keep development teams in sync.

5. Debugging Tools:

GDB: The GNU Debugger, used for debugging C, C++, and other languages.

Xcode: Apple's integrated development environment for macOS and iOS app development, including debugging tools.

6. Package Managers:

Npm (Node Package Manager): Used for managing and distributing Java Script packages.

Pip: A package manager for Python.

7. Containerization and Virtualization:

Docker: A platform for developing, shipping, and running applications in containers.

VirtualBox: An open-source virtualization tool for running virtual machines on your local system.

8. Text Editors for Terminal:

Vim: A highly configurable, text-based text editor often used for coding in terminal environments.

Emacs: Another highly extensible text editor often used for coding and other text-based tasks.

9. Task Runners and Build Tools:

Grunt: A JavaScript task runner used for automating repetitive tasks in web development.

Webpack: A popular JavaScript module bundler and build tool.

10. Code Linters and Formatters:

ESLint: A linter for identifying and fixing problems in JavaScript code.

Prettier: A code formatter that enforces consistent code style across projects.

11. Database Management Tools:

phpMyAdmin: A web-based tool for managing MySQL databases.

Sequel Pro: A macOS application for managing MySQL databases.

12. Continuous Integration/ Continuous Deployment (CI/CD) Tools:

Jenkins: An open-source automation server for building, testing, and deploying code.

Travis CI: A cloud-based CI/CD service for automating the testing and deployment of code.

13. Text and Documentation Generation:

Doxygen: A tool for generating documentation from annotated source code.

Javadoc: A documentation generation tool for Java.

The choice of coding tools depends on the programming languages you use, your specific development needs, and personal preferences. Many developers use a combination of these tools to create efficient and productive workflows.

8.4 Recording and documentation of Design Defects

Recording and documenting design defects is a crucial part of the product development and quality assurance process. Proper documentation helps ensure that defects are identified, tracked, and addressed effectively. The steps to consider when recording and documenting design defects:

1. The first step is to identify and understand the design defect. This may involve conducting tests, analyzing performance data, or simply noticing issues reported by users or team members.
2. Record basic Information e.g. Provide a clear and concise description of the defect, including its symptoms, impact and Specify where in the design the defect occurs.
3. Whenever possible, include evidence that supports the existence of the defect. This could include screenshots, log files, or test results.
4. Specify the version(s) of the design or product in which the defect is found. This is crucial for tracking whether the defect has been fixed in subsequent releases.
5. Group similar defects into categories or classes to help with analysis and prioritization. Common defect categories include functional, performance, security, and usability defects.
6. Consider using a defect tracking system or software to record and manage defects. Popular tools include JIRA, Bugzilla, or custom-built systems tailored to your organization's needs.
7. Track the status of each defect, from its initial discovery through its resolution. Common statuses include open, in progress, resolved, and closed.
8. Prioritize defects based on their severity and impact. Critical defects that pose safety risks or severely impact functionality should be addressed immediately, while lower-priority defects can be scheduled for future releases.
9. When a defect is fixed, document the details of the resolution, including the code changes made, tests conducted, and the version in which the fix will be included.
10. After a defect is resolved, ensure it is thoroughly retested to verify that the issue has been completely addressed and that no new issues have been introduced. Once a defect has been validated as resolved and passes all necessary tests, close it in the defect tracking system.
11. Regularly generate defect reports and share them with relevant stakeholders to provide visibility into the status of defect management efforts.
12. Conduct post-mortems or root cause analysis to understand why defects occurred and how similar issues can be prevented in future design or development projects. Use the data and insights from defect tracking to continuously improve the design and development processes.

By following these steps and maintaining detailed records, organizations can effectively manage and mitigate design defects, leading to higher-quality products and improved customer satisfaction.

Assignment 8.1.

- List down the common categories of coding tools.
- List down any three steps to resolve design defects.

SUMMARY

- Identifying and documenting design defects is the first crucial step in the resolution process.
- Prioritizing defects based on severity helps in effective defect resolution.
- Root cause analysis is necessary to understand why defects occurred and prevent future occurrences.
- Collaboration with cross-functional teams, including design, development, and quality assurance, is essential.
- Design changes and code fixes are made to address design defects.

- Thorough testing, including regression testing, ensures defect resolution and prevents new issues.
- Documentation updates reflect changes made to resolve defects.
- User feedback and continuous monitoring validate defect resolution.
- Clear communication with stakeholders and regular defect reporting is crucial.
- The defect resolution process is iterative and continuous improvement is key.

Check Your Progress

A. Multiple Choice Questions (MCQs)

1. What are the potential consequences of unresolved design defects? (a) Increased customer satisfaction (b) Improved functionality (c) Decreased usability (d) Lower safety standards
2. What is the purpose of root cause analysis in resolving design defects? (a) To blame individuals for defects (b) To understand why defects occurred and prevent similar issues (c) To hide the source of defects (d) To prioritize defects based on their severity
3. Which teams should be involved in resolving design defects? (a) Only the design team (b) Development and quality assurance teams (c) Only the product management team (d) No other teams are needed.
4. What types of testing are performed after making design and code changes to resolve defects? (a) Unit testing and integration testing (b) Only unit testing (c) Only user acceptance testing (d) No testing is required.
5. What kind of documentation should be updated during defect resolution? (a) Bug reports (b) Meeting minutes (c) User manuals, design specifications, and code comments (d) No documentation updates are necessary.
6. In the scope of improvement for future design, what is essential for designers to prioritize? (a) Avoiding user research and feedback. (b) Staying static and not adapting to new technologies. (c) Focusing on user-centered design and usability testing. (d) Ignoring accessibility standards.
7. What does sustainable design focus on? (a) Maximizing waste and carbon footprint (b) Reducing waste and using eco-friendly materials (c) Ignoring environmental impact (d) Prioritizing aesthetics over sustainability.
8. What is an important aspect of AI design? (a) Replacing human experiences with AI (b) Creating AI-driven interfaces that enhance user experiences (c) Avoiding AI in design altogether (d) Ethical design.
9. Why is collaboration with cross-functional teams important for designers? (a) It slows down the design process (b) It has no impact on design quality (c) Effective collaboration often results in more successful products (d) Collaboration with cross-functional teams is not necessary
10. What is the significance of continuous monitoring even after defect resolution? (a) It is unnecessary (b) To catch any new defects in the design phase (c) Some issues may only become apparent in a real-world environment (d) To blame the development team for defects

B. Fill in the blanks

1. Resolving design defects is a crucial part of the _____ development process.
2. The first step in resolving design defects is to _____ and document all defects.
3. Prioritizing defects based on severity is essential to address the most critical issues _____.
4. After making design and code changes, thorough testing is conducted to ensure defects _____.
5. Regression testing is important to ensure that defect resolution doesn't introduce new defects or break existing _____.
6. Even after defect resolution, continuous _____ of the product in production is necessary.
7. Use the experience gained from resolving defects to improve your design and _____ processes for reducing defect occurrence in the future.
8. The scope of improvement in design is limitless, and it is essential to stay curious and willing to _____ with new ideas and technologies.
9. Proper documentation of design defects helps ensure that defects are identified, tracked, and _____ effectively.
10. If a defect is related to the design itself, working with the design team to make necessary _____ is necessary.

C. True or False

1. Resolving design defects is not an essential part of the product development process.
2. Identifying and documenting all design defects is not a necessary step in resolving them.
3. Prioritizing defects based on severity is not crucial for effective defect resolution.
4. Root cause analysis is not required to understand why defects occurred and prevent similar issues in the future.
5. Sustainable design focuses on reducing waste and using eco-friendly materials.
6. Ethical design involves dark patterns and deceptive practices that manipulate users.
7. Collaboration with cross-functional teams often results in more successful products.
8. Designers should avoid staying curious and experimenting with new ideas and technologies to improve their work.
9. Involving end-users or beta testers in the validation process is not important for ensuring defect resolution meets their expectations.
10. Continuous monitoring of the product in production is not necessary after defect resolution.

D. Short Question Answers

1. What is the first step in resolving design defects?
2. Why is it important to prioritize defects based on severity?
3. Explain the significance of root cause analysis in defect resolution.
4. How does involving cross-functional teams contribute to effective defect resolution?
5. What are the primary actions taken to address design defects related to the design itself?

6. How do developers address defects related to the code?
7. Why is thorough testing, including regression testing, crucial in defect resolution?
8. In what ways can documentation be updated during the defect resolution process?
9. Why is user feedback important during the validation of defect resolution?
10. What is the role of continuous improvement in managing design defects?

PSSCIVE Draft Study Material © Not to be Published

Module 2**HTML AND CSS****Module Overview**

In this Module, you will first learn about " HTML " covering HTML, HTML Elements, HTML Attributes, and Document Structure. Then, we move to "Concept of lists – Unordered lists, ordered lists, Definition list ", where you will learn about Unordered Lists, Ordered Lists, Definition Lists, Image and Image mapping and HTML Hyperlink. Then, we discuss "FRAME IN WEBPAGE", which covers frame in webpage, its benefits and drawbacks, Creating Vertical Columns, Create client-side form in webpage, and XHTML.

Then, we discuss "CSS" covering will understand about CSS, CSS Versions, CSS Syntax, Types of CSS, Multiple Style Sheets and Background and borders of style elements. Lastly, we discuss "CSS box model", covering CSS box model, CSS Dimensions, CSS to display positioning, CSS Visibility, CSS Display and CSS Scrollbars. This Module equips you with essential knowledge and skills in HTML.

Learning Outcomes**Module Structure**

Session 1. HTML

Session 2. Concept of lists – Unordered lists, Ordered lists, Definition list

Session 3. CSS

Session 4. CSS box model

Session 1. HTML

Meet Ankit, a kid who loves the internet. He wanted to know how web pages worked, so he learned about "HTML". HTML was like a magic recipe with special tags "<>" to arrange things on a web page, just like arranging toys. Ankit used it to create his web page, add pictures, stories, and links. It was like being a digital artist and storyteller. With HTML, he made cool web pages to share his ideas and creativity with others, just like writing a story or drawing a picture as shown in Figure 1.1.



Fig. 1.1: Ankit learning HTML

In this chapter, you will learn about HTML, HTML Elements, HTML Attributes, and Document Structure.

1.1 HTML

HTML (Hypertext Markup Language) is a text-based method for defining the organization of content within an HTML document. It serves as a markup language essential for establishing the layout and substance of web pages. Often employed alongside CSS (Cascading Style Sheets) and JavaScript, HTML plays a crucial role in crafting interactive and visually captivating websites. By employing markup instructions, HTML instructs a web browser on the presentation of text, images, and various multimedia elements on a webpage. Its primary function is to structure web page content, with the option of incorporating scripts for interactivity and dynamic functionality, with JavaScript being the prevalent choice for scripting purposes.

HTML is a recognized standard established by the World Wide Web Consortium (W3C) and is broadly adhered to by leading web browsers across both desktop and mobile platforms. Notably, HTML5 represents the latest iteration of this specification.

Did You Know?

HTML stands for Hyper Text Markup Language. It's a markup language used to create web pages.

1.2 HTML Elements

HTML documents are composed of elements that serve to delineate the organization and substance of a web page. These elements are enclosed within angle brackets and customarily manifest as pairs, consisting of an opening tag and a corresponding closing tag.

For example:

`<p>This is a paragraph.</p>`

`Visit Example`

HTML (Hypertext Markup Language) offers a diverse array of elements that empower you to format and showcase content within web pages. These elements serve as the fundamental components of any HTML document. Here are a selection of frequently utilized HTML elements:

1. Text Elements:

`<p>`: Defines a paragraph of text.

`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`: Headings of different levels (1 being the highest) used to structure content.

`<a>`: Creates hyperlinks to other web pages or resources.

`` and ``: Used to emphasize or highlight text.

`<blockquote>`: Defines a block of quoted text.

`<code>`: Represents computer code within the text.

`<pre>`: Preformatted text, such as code blocks, where whitespace is preserved.

`<sub>` and `<sup>`: To display subscript and superscript text.

2. Lists:

``: Specifies an unsorted (bulleted) list.

``: Designates an ordered (numbered) list.

``: Denotes individual list items found within `` and ``.

3. Container Elements:

`<div>`: Specifies block level containing elements.

``: Specifies inline level containing elements.

4. Links and Anchors:

`<a>`: Creates hyperlinks to other web pages or resources.

`<link>`: Specifies relationships between the current document and external resources like stylesheets.

`<nav>`: Represents a section of navigation links.

5. Images and Multimedia:

``: Embeds images in a web page.

`<audio>`: Embeds audio content.

`<video>`: Embeds video content.

`<iframe>`: Embeds external web content, like maps or external websites.

6. Forms:

`<form>`: Establishes an HTML form to collect user input.

`<input>`: Represents diverse input field types (text, password, radio buttons, checkboxes, etc.).

`<textarea>`: Generates a multi-line text input field.

`<select>`: Specifies a drop-down list.

`<button>`: Defines a clickable button inside a form.

`<label>`: Supplies a label for form components.

`<option>`: Provides radio buttons.

7. Tables:

`<table>`: Defines a table.

`<thead>`: Represents header of a HTML table.

`<tbody>`: Defines body of a HTML table.

`<tfoot>`: Defines footer of a HTML table.

`<tr>`: Represents a table row.

`<th>`: Defines a table header cell.

`<td>`: Defines a table data cell.

`<caption>`: Provides a caption or title for a table.

8. Semantic Elements (introduced in HTML5):

`<header>`: Represents a container for introductory content.

`<footer>`: Represents a container for footer content.

`<nav>`: Represents a section of navigation links.

`<section>`: Represents a thematic grouping of content.

`<article>`: Defines a self-contained composition.

`<aside>`: Represents content that is tangentially related to the content around it.

`<main>`: Specifies the main content of a document.

`<figure>`: Specifies self-contained content, frequently with a caption (`<figcaption>`), and is typically referenced as a single unit.

`<figcaption>`: Represents a caption or a legend associated with a `<figure>` or with other content.

1. Meta Information:

`<head>`: Contains meta-information about the document, such as the title and links to external resources.

`<meta>`: Provides metadata about the HTML document (e.g., character encoding).

`<title>`: Provides title of the HTML page (document).

10. Comments:

`<!-- ... -->`: Used to add comments within HTML code for documentation or notes. Comments are not displayed on the web page.

These are just some of the basic HTML elements. HTML is highly extensible, and developers can create custom elements using web components. The choice of elements depends on the content and structure of your web page and the semantics you want to convey. Properly structuring your HTML document using these elements ensures better accessibility, SEO, and maintainability of your web pages.

Assignment 1.1.

Define the following HTML elements:

- Text
- Images and Multimedia
- Forms
- Tables
- Links and Anchors

1.3 HTML Attributes:

Attributes are a way to include extra details or alter the behavior of HTML elements. They are specified within the opening tag of an element and are typically in the form of name-value pairs. Attributes are frequently used with HTML elements to offer supplementary information or adjust their functionality. Some common attributes are:

src: Specifies the source URL (used in ``, `<script>`, `<a>`, etc.).

href: Specifies the destination URL (used in `<a>`, `<link>`, etc.).

alt: Provides alternative text for images (used in ``).

class: Assigns a CSS class for styling purposes.

id: Provides a unique identifier for elements.

style: Defines inline CSS styles.

placeholder: Sets a placeholder text for input fields.

disabled: Disabled form elements.

title: Provides title of the HTML element.

required: Specifies that a field must be filled out in a form.

HTML tags and attributes are combined to structure and format web content, create interactive forms, and define the layout and behavior of web pages. Understanding these elements and attributes is crucial for web development.

```

```

```
<a href="https://www.example.com" target="_blank">Visit Example</a>
```

1.4 Document Structure:

An HTML document follows a specific structure:

<!DOCTYPE html>: The document's type and version, in this case, HTML5, should be indicated. To commence an HTML document, it is advisable to include a Document Type Declaration (DTD). This declaration specifies the HTML version you are employing, setting the foundation for the document's structure and interpretation.

<html>: The `<html>` element, serving as the root element of an HTML document, encompasses all other elements present on the page. Typically, it includes two child elements: `<head>` and `<body>`.

<head>: The `<head>` section is responsible for housing meta-information related to the document, encompassing details like the title, character encoding, and references to external resources. While these details are not visible on the web page itself, they hold significance for search engines and browsers. Common elements that can be found in the `<head>` section include:

<meta>: Defines metadata such as character encoding and viewport settings.

<title>: Sets the title of the web page, which appears in the browser's title bar or tab.

<link>: Links to external resources like stylesheets (CSS) and web fonts.

<script>: Links to or embeds JavaScript files.

<style>: Contains internal CSS styles.

<body>: Contains the visible content of the web page. The `<body>` section contains the visible content of your web page, including text, images, forms, and interactive elements. This is where you structure the main content of your page using HTML elements like headings, paragraphs, lists, links, and more.

Assignment 1.2.

1. Define the following HTML attributes:
 - (a) class
 - (b) id
 - (c) style
2. Define the following HTML document specific structure:
 - (a) <meta>
 - (b) <link>
 - (c) <script>

1.5 HTML tags vs Elements.

HTML tags and HTML elements are related but distinct concepts in web development. Understanding the difference between them is essential for working with HTML effectively.

1.5.1 HTML Tags:

Definition: HTML tags serve as the building blocks of an HTML document, delineating the start and end of an HTML element. These tags are enclosed within angle brackets, typically appearing in pairs which consist of an opening tag and a closing tag. The opening tag specifies the element's name, while the closing tag shares the same name but is preceded by a forward slash (/). For instance, the opening tag <p> and closing tag </p> are used together to define a paragraph element.

Purpose: Tags serve as markers that tell the browser how to interpret and render the content between them. They provide structure and formatting instructions to the web page.

1.5.2 HTML Elements:

Definition: HTML elements are made up of an opening tag, closing tag, and the content (if any) that resides between them. An HTML element, as a whole, includes both the tags and the content enclosed within those tags. For example, <p>Hello, World!</p> is an HTML element that represents a paragraph containing the text "Hello, World!"

Purpose: HTML elements function as the fundamental constituents of an HTML document, playing a pivotal role in shaping and delivering the content of a web page. These elements encompass various aspects of a webpage's structure and content, encompassing essentials like headings, paragraphs, images, links, and a multitude of other components. Additionally, elements can be enriched by attributes that furnish supplementary information related to the element.

In summary, HTML tags are the individual markup components that denote the start and end of an HTML element, while HTML elements encompass the entire structure, including the tags and any content between them. Here are a few examples to illustrate the difference:

Tag: <h1> (opening tag) and </h1> (closing tag)

Element: <h1>Welcome to my website!</h1>

Tag: (opening tag) and (closing tag)

Element: Visit Example

When working with HTML, you'll be creating and manipulating HTML elements, which consist of both the tags and their content, to structure and present your web page's content effectively. Following a simplified example of a complete HTML document with a basic structure as shown in Figure 1.2.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Web Page</title>
  <link rel="stylesheet" href="styles.css">
  <script src="myscript.js"></script>
</head>
<body>
  <header>
    <h1>Welcome to My Website</h1>
    <nav>
      <!-- Navigation Links go here -->
    </nav>
  </header>
  <main>
    <article>
      <h2>Article Title</h2>
      <p>This is the content of the article.</p>
    </article>
    <!-- More content goes here -->
  </main>
  <footer>
    <p>&copy; 2023 My Website</p>
  </footer>
</body>
</html>

```

Fig. 1.2: A complete HTML document with a basic structure

Output

Welcome to My Website

Article Title

This is the content of the article.

© 2023 My Website

Fig. 1.3: Output for a complete HTML document with a basic structure

1.6 HTML BASIC TAGS

HTML (Hypertext Markup Language) uses a variety of basic tags to structure the content of a web page. These tags are fundamental building blocks for creating web documents.

1. `<!DOCTYPE html>`: Specifies the document type and version of HTML being used. For HTML5, the declaration is:

```
<!DOCTYPE html>
```

2. `<html>`: The root element that contains all other elements on the page. It includes the `<head>` and `<body>` sections.

```
<html>
```

```
<!-- Content goes here -->
```

```
</html>
```

3. `<head>`: Contains metadata about the document, such as character encoding, title, and links to external resources like stylesheets and scripts.

```
<head>
```

```
<!-- Metadata and links go here -->
```

```
</head>
```

4. `<title>`: This element serves the crucial role of establishing the title for the web page, a title that is prominently displayed in the browser's title bar or tab.

```
<title>My Web Page</title>
```

5. `<meta>`: Specifies metadata about the document, such as character encoding and viewport settings.

```
<meta charset="UTF-8">
```

6. `<body>`: Contains the visible content of the web page, including text, images, links, and more.

```
<body>
```

```
<!-- Content goes here -->
```

```
</body>
```

7. `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`: Headings of different levels, used to structure content with `<h1>` being the highest level and `<h6>` the lowest.

```
<h1>Main Heading</h1>
```

```
<h2>Subheading</h2>
```

8. `<p>`: Defines a paragraph of text.

```
<p>This is a paragraph.</p>
```

1.7 HTML formatting tags

HTML offers an array of formatting tags, empowering you to regulate the visual styling of text and content within a web page. Below are several frequently used HTML formatting tags:

Text Formatting:

``: Makes text bold.

``: Represents strong importance and is typically rendered as bold text.

`<i>`: Renders text in italics.

``: Represents emphasized text and is typically rendered in italics.

`<u>`: Underlines text.

`<s>` or `<strike>`: Renders text with a strikethrough.

`<sup>`: Makes text superscript (elevated above the baseline).

`<sub>`: Makes text subscript (lowered below the baseline).

CODE-

`<p>This is bold and <i>italic</i>text.</p>`

`<p>This is ^{superscript} and _{subscript}text.</p>`

Font Style and Size (Not recommended, use CSS for styling instead):

``: Sets font size, color, and face.

Code- `<p>Styled text</p>`

Text Alignment:

`<div>`: Defines a division or section for styling purposes and can be used to control text alignment.

`<center>`: Centers text and content within the element (deprecated in HTML5, use CSS for centering).

Code- `<div style="text-align: center;">Centered text</div>`

Line Breaks and Whitespace:

`
`: Inserts a line break within text.

`<pre>`: Preformatted text, preserving whitespace and line breaks.

CODE- This is some text.
Here is a new line.

`<pre>`

This text

has

preserved

line breaks.

`</pre>`

Quotations:

`<blockquote>`: Defines a block of quoted text.

`<q>`: Defines a short inline quotation.

CODE- `<blockquote>`

This is a blockquote.

`</blockquote>`

`<p>He said, <q>Quote me!</q></p>`

Abbreviations and Acronyms:

`<abbr>`: Represents an abbreviation or acronym, and you can provide a title attribute for the full text.

CODE- `<p><abbr title="World Wide Web">WWW</abbr> is amazing!</p>`

These are some of the basic HTML formatting tags. While HTML can be used for simple formatting, more advanced and responsive styling is typically accomplished using CSS (Cascading Style Sheets). CSS allows for greater control over the presentation of web content and separates styling from content, leading to more maintainable and flexible web pages as shown in Figure 1.3.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Formatting Example</title>
</head>
<body>
  <h1>Text Formatting</h1>
  <p>This is a <b>bold</b> and <i>italic</i> text. Also <u>underline</u> and <s>strikethrough</s> text.</p>
  <p>This is <sup>superscript</sup> and <sub>subscript</sub> text.</p>

  <h1>Font Style and Size</h1>
  <p><font size="4" color="red" face="Arial">Styled text using &lt;font&gt;</font></p>

  <h1>Text Align</h1>
  <div style="text-align: center;">This text is centered using inline styling</div>
  <center>This text is centered using center (deprecated)</center>

  <h1>Line breaks and whitespaces</h1>
  <p>This is some text. <br> Here is a new line.</p>

  <pre>
  This text
  has
  preserved
  line breaks.
  </pre>

  <h1>Quotations</h1>
  <blockquote>
  This is a blockquote.
  </blockquote>
  <p>He said, <q>Quote me!</q></p>

  <h1>Abbreviations and Acronyms</h1>
  <p><abbr title="World Wide Web">WWW</abbr>is amazing!</p>
</body>
</html>

```

Fig. 1.4: Basic HTML formatting tags

Output-

Text Formatting

This is a **bold** and *italic* text. Also underline and ~~strickethrough~~ text.

This is ^{superscript} and _{subscript} text.

Font Style and Size

Styled text using

Text Align

This text is centered using inline styling
This text is centered using center (deprecated)

Line breaks and whitespaces

This is some text.
Here is a new line.

```
This text  
has  
preserved  
line breaks.
```

Quotations

This is a blockquote.

He said, "Quote me!"

Abbreviations and Acronyms

.WWWis amazing!

Fig. 1.5: Output for basic HTML formatting tags

Assignment 1.3.

- List down the HTML BASIC tags.
- List down the HTML Text Formatting tags.
- List down the HTML Text Alignment tags.

1.8 HTML Color Coding

In HTML, you can specify colors using various methods, including color names, hexadecimal color codes, RGB values, and HSL values. Here's an overview of these methods:

Color Names:

HTML incorporates a collection of predefined color names at your disposal, simplifying the process of specifying colors directly within your HTML code. For instance:

CODE-'

```
<p style="color: red;">This text is red.</p>
<p style="color: blue;">This text is blue.</p>
```

Common color names include red, blue, green, black, white, yellow, purple, orange, and many more.

1.8.1 Hexadecimal Color Codes:

Hexadecimal color codes, a prevalent approach for defining colors in HTML, comprise a # symbol followed by six hexadecimal digits. These digits signify the red, green, and blue (RGB) components of the color.

CODE-

```
<p style="color: #FF0000;">This text is red.</p>
<p style="color: #0000FF;">This text is blue.</p>
```

Each pair of hexadecimal digits represents the intensity of the red, green, and blue channels, respectively, in that order. For instance, #FF0000 is pure red, and #0000FF is pure blue.

1.8.2 RGB Color Values:

You can specify colors using RGB values, where you define the intensity of the red, green, and blue channels separately. RGB values are specified using the rgb() function. For example:

CODE-

```
<p style="color: rgb(255, 0, 0);">This text is red.</p>
<p style="color: rgb(0, 0, 255);">This text is blue.</p>
```

Each value within the rgb() function can range from 0 to 255, representing the intensity of the respective color channel.

1.8.3 HSL Color Values:

HSL (Hue, Saturation, Lightness) is another method for specifying colors in HTML. It offers more control over color properties than RGB. HSL values are specified using the hsl() function. For example:

```
<p style="color: hsl(0, 100%, 50%);">This text is red.</p>
<p style="color: hsl(240, 100%, 50%);">This text is blue.</p>
```

The first value (0 to 360) represents the hue, the second value (0% to 100%) represents the saturation, and the third value (0% to 100%) represents the lightness.

These are the primary methods for specifying colors in HTML. You can use these color codes and values in HTML attributes like color, background-color, or within CSS styles to define the

colors of text, backgrounds, borders, and other elements on your web page As shown in Figure 1.6.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Color Example</title>
</head>
<body>
  <h1 style="color: red;">This heading uses a color name (red).</h1>
  <p style="color: #0000ff">This paragraph uses a hexadecimal color code (#0000FF, blue).</p>
  <div style="color: rgb(0, 128, 0);">This div uses a RGB color values (0, 128, 0 [green]).</div>
  <span style="color: hsl(240, 100%, 50%);">This span uses HSL color values (240, 100%, 50% [blue]).</span>
  <p style="color: rgba(255, 0, 0, 0.5);">This paragraph uses RGBA color values (255, 0, 0, 0.5 [semi transparent red]).</p>
  <p style="background-color: #ffff00;">This paragraph has a yellow background using hexadecimal color code.</p>
  <p style="background-color: rgb(255, 192, 203);">This paragraph has a pink background using RGB color values.</p>
  <p style="background-color: hsl(120, 100%, 75%);">This paragraph has a light green background using HSL color values.</p>
  <p style="border: 2px solid #ffa500;">This paragraph has an orange border using a hexadecimal color code.</p>
</body>
</html>
```

Fig. 1.6: HSL Color Values in HTML

Output

This heading uses a color name (red).

This paragraph uses a hexadecimal color code (#0000FF, blue).

This div uses a RGB color values (0, 128, 0 [green]).

This span uses HSL color values (240, 100%, 50% [blue]).

This paragraph uses RGBA color values (255, 0, 0, 0.5 [semi transparent red]).

This paragraph has a yellow background using hexadecimal color code.

This paragraph has a pink background using RGB color values.

This paragraph has a light green background using HSL color values.

This paragraph has an orange border using a hexadecimal color code.

Fig. 1.7: Output for HSL Color Values in HTML

1.8.4 Div and Span tags for grouping

The `<div>` and `` tags are commonly used in HTML for grouping and structuring content, but they serve different purposes and are used in different contexts as shown in Figure 1.8.

<div> (Division):

The `<div>` element is a block-level container used for grouping and structuring larger sections of content on a web page.

It is often used to create divisions or sections of content that can be styled and formatted together.

`<div>` elements are typically used to define the layout structure of a web page, such as headers, footers, sidebars, and content areas.

They can be used in conjunction with CSS to apply styles, such as backgrounds, borders, margins, and padding.

`<div>` elements create block-level boxes that typically start on a new line and take up the full width of their parent container.

```
<div id="header">
|   <h1>Welcome to My Website</h1>
</div>
<div id="sidebar">
|   <!-- Sidebar content goes here -->
</div>
<div id="content">
|   <!-- Main content goes here -->
</div>
<div id="footer">
|   &copy; 2024 My Website
</div>
```

Fig. 1.8: Div and Span tags

1.8.5 (Inline Span):

The `` element is an inline-level container used for grouping and applying styles to smaller, inline portions of text or content within a block-level element.

It is often used to target specific parts of a text or inline content for styling purposes, such as applying CSS styles, formatting, or JavaScript manipulation.

`` elements do not introduce line breaks by themselves; they only affect the content within their parent element.

They are useful when you want to style or manipulate specific words or phrases within a larger block of text.

Example:

```
<p>This is a <span style="color: blue;">blue</span> word in a sentence.</p>
```

```
<p>Click <span onclick="alert('Hello!')">here</span> to trigger an alert.</p>
```

In summary, `<div>` is used for structuring and grouping larger sections of content and creating block-level containers, while `` is used for inline-level grouping and styling of smaller

portions of text or content within a larger block-level element. Both elements are essential for organizing and styling content on a web page effectively as shown in figure 1.1.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Div and Span Example</title>
</head>
<body>
  <div style="background-color: #f0e60c;">Header</div>

  <div>
    <p>This is a <span style="background-color: yellow; font-weight: bold;">highlighted</span> word in a paragraph.</p>
    <p>Click <span style="background-color: yellow; font-weight: bold; cursor: pointer;"
      onclick="alert('hello')"
    >here</span>to trigger a alert.</p>
  </div>

  <div style="background-color: #f0e60c;">&copy; 2024 My Website</div>
</body>
</html>
```

Fig. 1.9: `` (Inline Span) in HTML

Output

Header

This is a **highlighted** word in a paragraph.

Click **here**to trigger a alert.

© 2024 My Website

Fig. 1.10: Output for `` (Inline Span) in HTML

SUMMARY

- HTML (Hypertext Markup Language) is essential for structuring web pages and is often used in conjunction with CSS and JavaScript.
- HTML elements are enclosed within angle brackets and consist of opening and closing tags.
- HTML includes a wide range of elements for structuring content, such as headings, links, lists, images, forms, tables, and more.
- Attributes are used to provide additional details or modify the behavior of HTML elements.
- HTML documents follow a specific structure with elements like `<!DOCTYPE html>`, `<html>`, `<head>`, and `<body>`.

- HTML tags denote the start and end of an element, while HTML elements include both tags and content.
- Formatting tags like ``, `<i>`, and `<u>` are used for text styling, and `` can define font properties.
- Color in HTML can be specified using color names, hexadecimal codes, RGB values, and HSL values.
- The `<div>` element is used for grouping and structuring larger sections of content, while `` is used for inline styling.
- Proper understanding of HTML elements and attributes is crucial for web development and content presentation.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What does HTML stand for? (a) Hypertext Markup Language (b) Hyper Transfer Markup Language (c) High-Level Text Markup Language (d) Hypertext Text Markup Language
2. Which organization established HTML as a recognized standard? (a) W3C (World Wide Web Consortium) (b) IEEE (Institute of Electrical and Electronics Engineers) (c) ISO (International Organization for Standardization) (d) IETF (Internet Engineering Task Force)
3. What is the latest iteration of the HTML specification? (a) HTML4 (b) HTML5 (c) XHTML (d) HTMLX
4. Which HTML element serves as the root element and encompasses all other elements? (a) `<body>` (b) `<head>` (c) `<html>` (d) `<root>`
5. Which HTML tag is used to define a paragraph of text? (a) `<para>` (b) `<text>` (c) `<p>` (d) `<line>`
6. What HTML tag is used to create hyperlinks to other web pages or resources? (a) `<link>` (b) `<a>` (c) `<hyper>` (d) `<href>`
7. Which HTML element represents emphasized text and is typically rendered in italics? (a) `<italic>` (b) `` (c) `` (d) `<emphasis>`
8. What is the purpose of the `<blockquote>` element in HTML? (a) To create a section of navigation links (b) To format text in bold (c) To define a block of quoted text (d) To represent an abbreviation or acronym
9. Which HTML tag is used to insert a line break within text? (a) `<line>` (b) `<lb>` (c) `
` (d) `<newline>`
10. What is the purpose of the `<div>` element in HTML? (a) To group and structure larger sections of content (b) To define inline spans of text (c) To create hyperlinks (d) To represent a table cell

B. Fill in the blanks

1. HTML stands for _____ Language.
2. The latest iteration of the HTML specification is _____.
3. HTML documents are composed of elements enclosed within _____ brackets.

4. The HTML tag used to define a paragraph of text is_____.
5. The opening and closing tags of an HTML element make up the _____.
6. The HTML tag used for creating _____ to other web pages or resources is <a>.
7. The _____ element is used to define a block of quoted text.
8. The _____ element is commonly used for grouping and structuring larger sections of content.
9. The _____ element is typically used for styling and grouping smaller portions of text or content.
10. _____ allows specifying colors using color names like "red" or "blue."

C. True or False

1. HTML is a text-based method used to define the organization of content within a document.
2. HTML elements are typically used for grouping and applying styles to larger sections of content.
3. The <meta> tag specifies metadata about the HTML document, such as character encoding and viewport settings.
4. The element is a block-level container used for structuring content.
5. Hexadecimal color codes consist of six digits that represent the red, green, and blue (RGB) components of a color.
6. The <blockquote> element is used to define headings of different levels.
7. The <head> section contains the visible content of the web page, including text, images, and links.
8. The tag should be used for styling text instead of CSS.
9. The <sub> tag is used to create subscript text.
10. The <abbr> element is used to insert line breaks within text.

D. Short Question Answers

1. What does HTML stand for, and what is its primary purpose in web development?
2. How does HTML work in conjunction with CSS and JavaScript in web development?
3. Describe the structure of an HTML element, including opening and closing tags.
4. Give an example of an HTML element used for defining a paragraph of text.
5. Explain the difference between HTML tags and HTML elements.
6. What is the purpose of HTML attributes, and how are they specified in an HTML element?
7. Why is it important to properly structure your HTML document using semantic elements?
8. What is the role of the <head> section in an HTML document, and what kind of information does it contain?
9. How does the <div> element differ from the element in HTML, and when should each be used?
10. Name three different methods for specifying colors in HTML, and briefly explain how each works.

Session 2. Concept of lists – Unordered lists, Ordered lists, Definition list

Meet Aman, a curious kid who loved organizing things. He learned about the "Concept of Lists" in a fun way. He found out there were three types of lists:

Unordered List: Like putting toys on a shelf without any order, just a bunch of things together.

Ordered List: It's like arranging toys from big to small or from old to new on a shelf.

Definition List: This list paired things up, like matching toy cars with their colors, and each thing had a special explanation.

Aman learned that lists were super helpful for organizing thoughts, just like arranging toys neatly or pairing them up. Lists made it easy to remember things and keep them in order, like a secret tool for staying organized as Shown in figure 2.1.



Fig. 2.1: Aman arranging toys

In this chapter, you will learn about Unordered Lists, Ordered Lists, Definition Lists, Image and Image mapping and HTML Hyperlink.

Within HTML, you have the flexibility to generate various list types for structuring and displaying content on a webpage. These primarily consist of unordered lists, ordered lists, and definition lists.

2.1 Unordered Lists ()

Unordered lists are used to represent lists of items that don't have a specific numerical or sequential order. Items in an unordered list are typically preceded by bullet points (or other symbols) to indicate each list item.

The element is used to define an unordered list, and each list item is enclosed within (list item) tags.

Common uses include navigation menus, itemized lists, and any list where the order of items doesn't matter.

Example

```
<ul>
```

```
<li>Item 1</li>
```

```
<li>Item 2</li>
```

```

    <li>Item 3</li>
</ul>

```

Did you know?

Unordered lists, which have no inherent order and each item is bulleted.

2.2 Ordered Lists ():

Ordered lists are used for enumerating items in a specific numerical or sequential manner. Each item in an ordered list is usually prefaced by numbers or letters that signify their position.

To create an ordered list, you use the element, and each individual item within the list is encapsulated in tags. Ordered lists are useful for scenarios such as outlining numbered procedural steps, establishing rankings, or any situation where the sequential order of items holds significance.

Example

```

<ol>
    <li>First step</li>
    <li>Second step</li>
    <li>Third step</li>
</ol>

```

Did you know?

Ordered lists, which have an inherent order and each item is numbered.

2.3 Definition Lists (<dl>, <dt>, <dd>)

Definition lists are used to represent a list of terms (definitions) and their corresponding descriptions or definitions.

The <dl> element is used to define a definition list. Each term is enclosed within a <dt> (definition term) element, and its corresponding description is enclosed within a <dd> (definition description) element.

Common uses include glossaries, dictionaries, and any situation where you need to present terms and their explanations.

Example

```

<dl>
    <dt>HTML</dt>
    <dd>Hypertext Markup Language</dd>
    <dt>CSS</dt>
    <dd>Cascading Style Sheets</dd>
    <dt>JavaScript</dt>
    <dd>A programming language for the web</dd>
</dl>

```

Did you know?

HTML Description List or Definition List displays elements in definition form like in dictionary. The <dl>, <dt> and <dd> tags are used to define description list.

Type of List in HTML as shown in Figure 2.2.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>List Examples</title>
</head>
<body>
  <h1>Type of Lists</h1>

  <!-- Unordered Lists -->
  <h2>Unordered Lists</h2>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>

  <!-- Ordered Lists -->
  <h2>Ordered Lists</h2>
  <ol>
    <li>First Step</li>
    <li>Second Step</li>
    <li>Third Step</li>
  </ol>

  <!-- Definition Lists -->
  <h2>Definition Lists</h2>
  <dl>
    <dt>HTML</dt>
    <dd>Hypertext Markup Language</dd>
    <dt>CSS</dt>
    <dd>Cascading Style Sheets</dd>
    <dt>JavaScript</dt>
    <dd>A programming language for the web.</dd>
  </dl>
</body>
</html>
```

Fig. 2.2: Type of List in HTML

Output-

Type of Lists

Unordered Lists

- Item 1
- Item 2
- Item 3

Ordered Lists

1. First Step
2. Second Step
3. Third Step

Definition Lists

HTML

Hypertext Markup Language

CSS

Cascading Style Sheets

JavaScript

A programming language for the web.

Fig. 2.3: Output for Type of List in HTML

2.4 Image and Image mapping

In HTML, the `` element is used to showcase images within a web page. Furthermore, you have the option to implement image maps, which enable you to delineate interactive regions within an image. Each of these regions can link to distinct URLs or activate specific actions as shown in Figure 2.4.

Note: The image is inserted using the `` tag.

Here's an overview of both the `` element and the concept of image mapping:

2.4.1 The `` Element (Image Embedding):

The `` element serves the purpose of embedding images directly within a web page. It is a self-contained tag, so there's no need for a closing `` tag.

- The `src` attribute is utilized to designate the source URL of the image file. This URL can be either an absolute path or a relative path to the image file.
- The `alt` attribute is responsible for supplying alternative text for the image. This text is shown in cases where the image cannot be loaded or for accessibility reasons.
- Attributes such as `width` and `height` can be configured to manage the image's dimensions. Nevertheless, it is often recommended to use CSS for styling purposes.

Example:

```

```

2.4.2 Image Mapping (Image Maps):

Image mapping allows you to define clickable regions on an image, each associated with a specific action or URL. This is useful for creating interactive diagrams, navigation menus, or image-based forms.

When crafting an image map, the process entails using the <map> element to delineate the map. Contained within this <map> element, you employ <area> elements to specify the regions of the image that are clickable.

The <map> element has a name attribute that should match the usemap attribute of the associated element.

Each <area> element can have different attributes, depending on the shape of the clickable area:

For rectangular areas: shape, coords, and href.

For circular areas: shape, coords, and href.

For polygonal areas: shape, coords, and href.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Maps</title>
</head>
<body>
  <h2>Image Maps</h2>

  <map name="deskmap">
    <area shape="rect" coords="175, 242, 420, 358"
    href="https://en.wikipedia.org/wiki/Computer_keyboard" alt="Keyboard" target=_blank>

    <area shape="rect" coords="444, 251, 481, 357"
    href="https://en.wikipedia.org/wiki/Computer_mouse" alt="mouse" target=_blank">

    <area shape="rect" coords="375, 14, 481, 357"
    href="https://en.wikipedia.org/wiki/Book" alt="Diary" target=_blank">
  </map>
</body>
</html>
```

Fig. 2.4: Image and Image mapping in HTML

Output**Image Maps**

Fig. 2.5: Output for Image and Image mapping in HTML

Run the above code in your code editor and try to click on keyboard, mouse, and diary. It will redirect you to different web pages based on which object you have clicked.

2.5 HTML Hyperlink

In HTML, you can create hyperlinks (or links) to navigate between web pages, resources, or locations on the internet. Hyperlinks are created using the `<a>` (anchor) element. Here's how you can create hyperlinks in HTML:

```
<a href="URL">Link Text</a>
```

Here's a breakdown of the important parts of an HTML hyperlink:

<a>: The `<a>` element is used to define the hyperlink.

href: The href attribute specifies the URL or destination of the link. It can be an absolute URL (e.g., "https://www.example.com") or a relative URL (e.g., "page.html").

Link Text: Indeed, the link text is the visible text displayed on a web page, and it acts as the clickable link. This textual content is positioned between the opening `<a>` tag and its corresponding closing `` tag. It is what users interact with to navigate to the linked destination.

Here are a few examples of how to use hyperlinks in HTML as shown in Figure 2.6.

Linking to an External Website: `Visit Example.com`

Linking to an Internal Page (Relative URL): `Learn About Us`

Linking to an Email Address (Mailto): `Contact Us`

Linking to a Phone Number (Tel): `Call Us`

Linking to a File (e.g., PDF): `Download PDF`

Linking to a Specific Section of a Page (Internal Anchors): `Go to Section 2`


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hyperlinks Examples</title>
</head>
<body>
  <h1>Types of Hyperlinks</h1>

  <!-- Link to an External Website -->
  <p><a href="https://www.example.com">Visit example.com</a></p>

  <!-- Link to an Internal Page (Relative URL) -->
  <p><a href="about.html">Learn About Us</a></p>

  <!-- Link to an Email Address (mailto) -->
  <p><a href="mailto:info@example.com">Contact Us</a></p>

  <!-- Link to a Phone Number (tel) -->
  <p><a href="tel: +919000000000">Call Us</a></p>

  <!-- Link to a File (e.g., PDF) -->
  <p><a href="document.pdf">Download PDF</a></p>

  <!-- Link to a specific section of a page (Internal Anchor) -->
  <p><a href="#section2">Go to Section 2</a></p>

  <!-- Anchor Point for Internal Link -->
  <h2 id="section2">Section 2</h2>
</body>
</html>

```

Fig. 2.6: Type of Hyperlinks in HTML

OUTPUT

Types of Hyperlinks

[Visit example.com](https://www.example.com)

[Learn About Us](about.html)

[Contact Us](mailto:info@example.com)

[Call Us](tel: +919000000000)

[Download PDF](document.pdf)

[Go to Section 2](#section2)

Section 2

Fig. 2.6: Output for Type of Hyperlinks in HTML

We have used different <a> (anchor) elements to create various types of hyperlinks:

A link to an external website (<https://www.example.com>).

A link to an internal page (about.html) within the same website.

A mailto link to send an email to info@example.com.

A mailto link to call a phone number +91 9000000000.

A link to download a PDF file (document.pdf).

An internal anchor link to navigate to a specific section on the same page (#section2).

For the internal anchor link, we've defined an anchor point with the corresponding id attribute (id="section2") to create a destination within the same page.

Assignment 2.1.

- List down the different HTML Hyperlinks.
- List down the different attributes of <area> element in Image mapping.

2.6 Uniform Resource Locator (URL) and URL encoding

A Uniform Resource Locator (URL) serves as a standardized address employed to identify and pinpoint resources on the internet. URLs play a foundational role in web navigation, allowing users to reach websites, web pages, files, and various other online resources. A standard URL comprises several essential components:

Scheme: This specifies the protocol or method used to access the resource. Common schemes include "http," "https," "ftp," "mailto," and more.

Domain: This is the internet address of the server where the resource is hosted. It can be represented as an IP address (e.g., "192.168.1.1") or a domain name (e.g., "www.example.com").

Port: An optional element that denotes the port number on the server to establish a connection with. When omitted, it automatically defaults to the standard port associated with the selected scheme, such as 80 for HTTP and 443 for HTTPS.

Path: This identifies the specific resource or location on the server. It is often represented as a directory or file path (e.g., "/folder/file.html").

Query: An optional part that furnishes supplementary parameters to the resource, frequently utilized in web applications to transmit data (e.g., "?search=keyword")

Fragment: Another optional component that specifies a particular section or fragment within the resource (e.g., "#section2").

A complete URL might look like this:

<https://www.example.com:8080/path/to/resource?param1=value1¶m2=value2#section3>

2.7 URL Encoding

URL encoding, also known as percent encoding, is a mechanism used to represent reserved or special characters within a URL. URLs are restricted to a specific set of characters, and not all characters can be used directly. Reserved characters, such as spaces or special symbols, must be encoded to ensure they are transmitted correctly.

The process involves replacing each reserved character with a "%" sign followed by two hexadecimal digits that represent the character's ASCII code. For example:

Space becomes %20.

Ampersand (&) becomes %26.

Question mark (?) becomes %3F.

URL encoding is essential when passing data via query parameters in a URL or when handling file paths that may contain special characters. Most programming languages and web frameworks provide built-in functions for URL encoding and decoding to handle this process automatically.

Did You Know?

URL encoding converts non-ASCII characters into a format that can be transmitted over the internet.

2.8 Table in web page

Tables within web pages serve the purpose of structuring and displaying data in a well-organized grid format. They are composed of rows and columns, with each intersection forming a cell. Tables find application in a multitude of scenarios, including showcasing tabular data, constructing grids, arranging information, and establishing page layouts.

Elements of a Table:

To create tables in HTML, you need to use a set of specific HTML elements that define the structure and content of the table. Here are the key elements of an HTML table:

<table>: The <table> element is used to define the entire table. It acts as a container for all the table-related elements, including rows, columns, and cells.

<tr>: The <tr> (table row) element is used to define a row within the table. Each row contains one or more cells.

<th>: The <th> (table header cell) element is used to define header cells within a table row. Header cells typically contain column headings and are rendered in bold by default.

<td>: The <td> (table data cell) element is used to define regular data cells within a table row. Data cells contain the actual content of the table.

<caption>: The <caption> element is used to provide a title or caption for the table. It is typically placed within the <table> element but outside of any rows.

Example 2.7 of Creating a Simple Table:

Here's an example of code a simple table with two rows and three columns as shown in Figure 2.7.

```
<table>
  <caption>Sample HTML Table</caption>
  <thead>
    <tr>
      <th>Header 1</th>
      <th>Header 2</th>
      <th>Header 3</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Data 1</td>
      <td>Data 2</td>
      <td>Data 3</td>
    </tr>
  </tbody>
</table>
```

Fig. 2.7: Simple table with two rows and three columns

In this example

<table> defines the entire table.

<caption> provides a title for the table.

<tr> defines two rows.

<thead> defines table headers.

<tbody> defines table body.

<th> defines header cells for column headings.

<td> defines data cells with actual content.

This creates a basic table structure with headers in the first row and data in the second row.

Header 1	Header 2	Header 3
Data 1	Data 2	Data 3

Fig. 2.8: Output for Simple table with two rows and three columns

Assignment 2.2.

- List down the elements used for creating a table.
- List down the several essential components of URL.

Complex Table example in HTML as shown in Figure 2.9.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Complex Table Example</title>
</head>
<body>
  <h1>Complex Table Example</h1>
  <table style="width: 100%; border-collapse: collapse;">
    <caption>Employee Information</caption>
    <thead>
      <tr>
        <th style="background-color: #f2f2f2; border: 1px solid #ddd; padding: 8px; text-align: left;">Employee ID</th>
        <th style="background-color: #f2f2f2; border: 1px solid #ddd; padding: 8px; text-align: left;">First Name</th>
        <th style="background-color: #f2f2f2; border: 1px solid #ddd; padding: 8px; text-align: left;">Last Name</th>
        <th style="background-color: #f2f2f2; border: 1px solid #ddd; padding: 8px; text-align: left;">Department</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">0001</td>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">Kunal</td>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">Singh</td>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">Engineering</td>
      </tr>
      <tr>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">0002</td>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">Rajesh</td>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">Kahar</td>
        <td style="border: 1px solid #ddd; padding: 8px; text-align: left;">Engineering</td>
      </tr>
    </tbody>
    <tfoot>
      <tr>
        <td colspan="2" style="border: 1px solid #ddd; padding: 8px; text-align: left;">Total Employees:</td>
        <td colspan="2" style="border: 1px solid #ddd; padding: 8px; text-align: left;">2</td>
      </tr>
    </tfoot>
  </table>
</body>
</html>

```

Fig. 2.9: Complex Table Example in HTML

OUTPUT-

Complex Table Example

Employee Information			
Employee ID	First Name	Last Name	Department
0001	Kunal	Singh	Engineering
0002	Rajesh	Kahar	Engineering
Total Employees:		2	

Fig. 2.10: Output for Complex Table Example in HTML

2.9 <tfoot>, <tbody>, <colgroup>and <caption>

2.9.1 <tfoot> Element: The <tfoot> element is used to define a footer section for a table. It typically appears after the <tbody> (table body) section and before the <table> element is closed. The content inside <tfoot> often contains summary or footer information related to the table. It's a useful element for providing context or summary data at the bottom of a table. Here's an example of how to use the <tfoot> element as shown in Figure 2.11.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Table</title>
</head>
<body>
  <table>
    <thead>
      <tr>
        <th>Header 1</th>
        <th>Header 2</th>
        <th>Header 3</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>Data 1</td>
        <td>Data 2</td>
        <td>Data 3</td>
      </tr>
    </tbody>
    <tfoot>
      <tr>
        <td colspan="3">Footer Text</td>
      </tr>
    </tfoot>
  </table>
</body>
</html>

```

Fig. 2.11: <tfoot> element in HTML

Output

Header 1	Header 2	Header 3
Data 1	Data 2	Data 3
Footer Text		

Fig. 2.12: Output for <tfoot> element in HTML

2.9.2 <tbody> Element: The <tbody> element plays a pivotal role in defining the central body of a table. It typically encompasses the rows and data cells within the table. Although not obligatory, it is considered good practice to employ <tbody> for organizing the primary content of the table.

2.9.3 <caption> Element: The <caption> element is used to provide a title or caption for a table. It appears above or below the table, and it can provide context or a brief description of the table's content. The <caption> element is optional, but it is recommended for accessibility and to improve table clarity. Here's an example of how to use the <tbody> element as shown in Figure 2.13.


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Employee List</title>
</head>
<body>
  <table>
    <caption>Employee List</caption>
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Department</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>001</td>
        <td>Kunal Singh</td>
        <td>Engineering</td>
      </tr>
      <tr>
        <td>002</td>
        <td>Rajesh Kahar</td>
        <td>Engineering</td>
      </tr>
      <!-- More rows... -->
    </tbody>
  </table>
</body>
</html>

```

Fig. 2.13: <caption> element in HTML

Output

Employee List		
ID	Name	Department
001	Kunal Singh	Engineering
002	Rajesh Kahar	Engineering

Fig. 2.14: Output <caption> element in HTML

2.9.4 <colgroup> Element:

The <colgroup> element is used to define groups of columns in a table and apply styling or other attributes to those columns collectively. It is often used in conjunction with the <col> element to specify attributes for individual columns or groups of columns. Here's an example of how to use the <colgroup> element as shown in Figure 2.15.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Understanding colgroup</title>
</head>
<body>
  <table>
    <colgroup>
      <col style="background-color: #f2f2f2;">
      <col style="font-weight: bold;">
    </colgroup>
    <thead>
      <tr>
        <th>Header 1</th>
        <th>Header 2</th>
        <th>Header 3</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>Data 1</td>
        <td>Data 2</td>
        <td>Data 3</td>
      </tr>
      <!-- More rows... -->
    </tbody>
  </table>
</body>
</html>

```

Fig. 2.15: <colgroup> element in HTML

OUTPUT

Header 1	Header 2	Header 3
Data 1	Data 2	Data 3

Fig. 2.16: output for <colgroup> element in HTML

In this example, the <colgroup> element is used to group the first two columns together, and inline styles are applied to set background color and font weight for those columns. The <col> elements within <colgroup> allow you to apply additional attributes to individual columns within the group.

Assignment 2.3.

1. Write down a code of the following element:

- <tbody>
- <colgroup>
- <caption>
- <tfoot>

2.10 Create client-side form in webpage

Designing a client-side form in a web page entails the utilization of HTML to craft the form's layout and integrating form components like text fields, radio buttons, checkboxes, and a submit button.

HTML introduces the <form> element, which serves as the foundation for constructing a structured segment within a web page encompassing interactive form elements. Forms play a pivotal role in web development, facilitating a wide array of applications such as user registration, search functionality, data submission, and more. The <form> element acts as a cohesive container, encompassing all the form controls, and also dictates the rules governing the handling of data entered into the form by users. It establishes the framework for collecting and processing user input within the context of web applications and websites.

Here's a breakdown of the key attributes and purposes of the <form> element:

2.2.1 Action (action) Attribute: The action attribute specifies the URL or script to which the form data should be sent for processing when the user submits the form.

Example: <form action="submit_form.php" method="post">

2.2.2 Method (method) Attribute: The method attribute defines the HTTP method used to send the form data to the server. Common methods are "get" and "post."

GET appends form data to the URL and is suitable for simple queries. POST sends form data in the HTTP request body and is used for more complex or sensitive data.

Example: <form action="submit_form.php" method="post">

2.2.3 Form Controls:

Form controls are interactive elements like text inputs, radio buttons, checkboxes, select dropdowns, and buttons that allow users to input data or make selections. Form controls are placed within the <form> element.

2.2.4 name Attribute: The name attribute is used to identify each form control when the form is submitted. The name is sent along with the user's input data.

Example: <input type="text" name="username">

2.2.5 id Attribute:

The id attribute provides a unique identifier for each form control. It is often used with labels to associate them with their corresponding form controls for better accessibility.

Example: <label for="username">Username:</label><input type="text" id="username" name="username">

2.2.6 for Attribute:

The for attribute in <label> elements specifies which form control it is associated with. It matches the id of the form control.

Example: <label for="username">Username:</label><input type="text" id="username" name="username">

2.2.7 required Attribute: The required attribute can be added to form controls to indicate that a user must provide input before submitting the form. It enforces input validation on the client side.

Example: `<input type="text" name="name" required>`

2.2.8 Submit Button (<input type="submit">) or Submit Action: A submit button or form action (e.g., JavaScript function) is used to trigger the submission of the form data to the server when clicked.

Example: `<input type="submit" value="Submit">`

2.2.9 Reset Button (<input type="reset">) (Optional): A reset button allows users to clear the form's input fields and reset them to their initial values.

Example: `<input type="reset" value="Reset">`

The `<form>` element, along with its associated form controls and attributes, provides a structured and standardized way to collect and submit data on web pages. Form data can be processed on the server using server-side scripts (e.g., PHP, Python, Node.js) or handled on the client side with JavaScript.

2.2.10 Text Input (<input type="text">): `<label for="name">Name:</label>`: The `<label>` element provides a label for the input field. The `for` attribute is associated with the `id` attribute of the corresponding input field for accessibility.

`<input type="text" id="name" name="name" required>`: This creates a text input field with the `id` of "name," the `name` attribute to identify the field in the form submission, and `required` to indicate that this field is mandatory.

2.2.11 Email Input (<input type="email">): Similar to the text input, but with the `type` attribute set to "email" to ensure valid email input.

2.2.12 Radio Buttons (<input type="radio">): Radio buttons are used for mutually exclusive options (e.g., gender). Each radio button has a unique `id`, a `name` attribute to group them, and a `value` attribute to specify the value associated with the option.

2.2.13 Checkboxes (<input type="checkbox">): Checkboxes allow multiple selections (e.g., interests). Each checkbox has a unique `id`, a `name` attribute to group them (using an array in this case), and a `value` attribute to specify the value associated with the option.

2.2.14 Textarea (<textarea>): `<label for="message">Message:</label>`: A label for the textarea. `<textarea id="message" name="message" rows="4" required></textarea>`: This creates a multi-line text input field with the `id` of "message," the `name` attribute, `rows` attribute to specify the visible number of rows, and `required` to make it mandatory.

2.2.15 Submit Button (<input type="submit">): This input element creates a button that, when clicked, submits the form.

2.2.16 <select>: The `<select>` element in HTML is used to create a dropdown list or a select box, allowing users to choose one or more options from a list of predefined choices. The `<select>` element is often combined with `<option>` elements to define the available choices within the dropdown list as shown in Figure 2.17.

```
<select>
  <option value="option1">Option 1</option>
  <option value="option2">Option 2</option>
  <option value="option3">Option 3</option>
  <!-- More options... -->
</select>
```

Fig. 2.17: code for <select>

Let's break down the key components of the <select> element and its associated elements:

2.2.17 <select> Element: The <select> element creates the dropdown list container. It may have attributes such as name and id to identify and reference the select box in HTML or JavaScript.

Example: `<select name="gender" id="gender">`

2.2.18 <option> Elements: Inside the <select> element, you can include one or more <option> elements to define the individual choices in the dropdown list. Each <option> element represents a single choice. The value attribute of each <option> element specifies the value that will be sent to the server when the form is submitted. The text content between the opening and closing <option> tags is what the user sees as the option label.

2.2.19 Multiple Selection (Optional): You can allow users to select multiple options by adding the multiple attribute to the <select> element. This creates a multi-select dropdown.

Example: `<select name="interests" multiple>`

2.2.20 Disabled Options (Optional): You can disable specific options by adding the disabled attribute to the <option> element. Disabled options cannot be selected by the user.

Example: `<option value="disabled_option" disabled>Disabled Option</option>`

2.2.21 Preselected Option: You can specify which option should be preselected when the page loads by adding the selected attribute to the <option> element.

Example: `<option value="preselected_option" selected>Preselected Option</option>`

Overall, these form elements allow users to input and submit data, and the associated attributes provide instructions and validation rules for the form fields. When the form is submitted, the data is sent to the server for processing based on the action and method attributes of the <form> element as shown in Figure 2.18.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Client Side Form Example</title>
</head>
<body>
  <h1>Contact Us</h1>

  <!-- Form Element -->
  <form action="submit_form.php" method="POST">
    <!-- Text Input -->
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <br>
    <!-- Email Input -->
    <label for="email">Email Address</label>
    <input type="email" id="email" name="email" required>
    <br>
    <!-- Select -->
    <label for="color">Select your favourite color:</label>
    <select name="color" id="color">
      <option value="red">Red</option>
      <option value="green">Green</option>
      <option value="blue">Blue</option>
      <option value="yellow">Yellow</option>
    </select>
    <br>
    <!-- Radio Buttons -->
    <label>Gender:</label>
    <input type="radio" id="male" name="gender" value="male">
    <label for="male">Male</label>
    <input type="radio" id="female" name="gender" value="female">
    <label for="female">Female</label>
    <br>
    <!-- Textarea -->
    <label for="message">Message</label>
    <textarea name="message" id="message" cols="30" rows="10" required></textarea>
    <br>
    <!-- Submit Button -->
    <input type="submit" value="Submit">
  </form>
</body>
</html>

```

Fig. 2.18: Contact Us in HTML

OUTPUT

Contact Us

Name:

Email Address

Select your favourite color: Red

Gender: Male Female

Message

Fig. 2.19: Output for Contact Us in HTML

Assignment 2.4.

- List down the syntax of the following key attributes and its purposes of the <form> element:
 - Checkboxes
 - Submit Button
 - Text Input
 - Name

2.11 Use Headers and Metadata of the document

2.11.1 Metadata of the Document: Metadata serves the crucial purpose of furnishing information about the document itself, encompassing essential details such as the document's title, character encoding, authorship, and an array of other pertinent attributes. This suite of metadata elements forms an integral component of web development and is indispensable for facilitating effective communication and interpretation of web documents. Among the commonly employed metadata elements are:

Title (<title>): The <title> element plays a pivotal role in the determination of the HTML document's title, a piece of information that is prominently displayed in the browser's title bar or tab. Beyond its evident role in enhancing user experience, the <title> element bears significant weight in the realm of search engine optimization (SEO), making it an essential facet of web development.

Base (<base>): The <base> element specifies a base URL for relative URLs within the document. It helps resolve relative URLs for linked resources.

Link (<link>): The <link> element is primarily used to link external resources such as stylesheets (CSS) and icon files (favicon). It can also be used for linking to other documents and defining relationships.

Styles (<style>): The <style> element is used to include internal CSS styles within the HTML document. It can define CSS rules for styling the content.

Script (<script>): The <script> element is used to include JavaScript code within the HTML document. It can be placed in the <head> or <body> section of the document.

2.11.2 HTML Meta Tag: The <meta> tag is a versatile tool employed for furnishing metadata pertinent to the HTML document. Typically residing within the <head> section of the document, the <meta> tag serves to offer essential information. Notable attributes associated with the <meta> tag encompass:

charset: Specifies the character encoding for the document (e.g., <meta charset="UTF-8">).

name and content: Used to define various metadata, such as author, description, keywords, and viewport settings (e.g., <meta name="author" content="John Doe">).

2.11.3 XHTML (Extensible Hypertext Markup Language): XHTML is a stricter and more XML-compliant version of HTML. It follows XML syntax rules, including proper nesting and closing of tags. Some key characteristics of XHTML include:

Tags and attributes must be lowercase.

All tags must be properly nested and closed (e.g., <p>...</p>).

Attribute values must be enclosed in double quotes.

Self-closing tags for elements like and
 (e.g., <imgsrc="image.jpg" />).

2.11.4 HTML Deprecated Tags and Attributes: Deprecated HTML tags and attributes are considered outdated or obsolete and should be avoided in modern web development. They may still work in browsers but are not recommended for use. Some examples include as shown in Figure 2.20.

Deprecated Tags: , <center>, <strike>, <frame>, <frameset>, etc.

Deprecated Attributes: align, bgcolor, border, cellpadding, cellspacing, etc.

Modern web development emphasizes the use of CSS for styling and layout and JavaScript for interactivity instead of relying on deprecated HTML attributes and tags.

It is important to keep up with current HTML and web development standards to ensure compatibility, accessibility, and a positive user experience on various devices and browsers.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="author" content="Vijay Goswami">
  <meta name="description" content="A sample webpage with metadata">
  <meta name="keywords" content="HTML, CSS, JavaScript, Metadata">
  <title>Document</title>
  <link rel="stylesheet" href="metadata.css">
</head>
<body>
  <h1>Welcome to My Webpage</h1>
  <p>This is a sample webpage with metadata, styles and scripts.</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
  
  <script>
    // JavaScript code goes here
    console.log("Hello World!")
  </script>
</body>
</html>

```

Fig. 2.20: HTML Deprecated Tags and Attributes

Output-

Welcome to My Webpage

This is a sample webpage with metadata, styles and scripts.

- Item 1
- Item 2
- Item 3

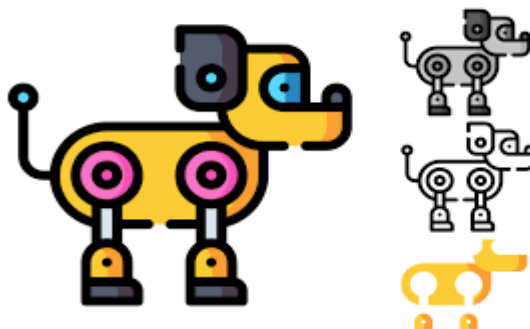


Fig. 2.21: Output for HTML Deprecated Tags and Attributes

Explanation

In this example: <meta> tags provide metadata about the document, including character encoding (charset), viewport settings (viewport), authorship (author), description (description), and keywords (keywords).

The <title> element defines the title of the webpage that appears in the browser's title bar or tab.

The <link> element references an external CSS stylesheet named "styles.css" for styling the webpage.

The element is a self-closing tag for displaying an image.

The <script> element includes JavaScript code for the web page. In this case, it logs a message to the browser console.

This example demonstrates the use of various metadata-related elements and attributes, the <meta> tag, and how XHTML syntax is used for self-closing tags like .

SUMMARY

- HTML lists come in three types: unordered, ordered, and definition lists.
- Unordered lists use and for items with no specific order.
- Ordered lists use and for items in a specific order, often numbered.
- Definition lists use <dl>, <dt>, and <dd> for terms and their explanations.
- Images are embedded using with attributes like "src" and "alt" for accessibility.
- Image maps use <map> and <area> to create clickable regions in images.
- Hyperlinks are created with <a> and use "href" for destination URLs.
- URL components include scheme, domain, port, path, query, and fragment.
- URL encoding replaces reserved characters with "%xx" codes.
- Tables in HTML are structured with <table>, <tr>, <th>, and <td> elements.
- The <caption> element provides titles for tables.
- Elements like <tfoot>, <tbody>, and <colgroup> enhance table structure and styling.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. Which HTML element is used to create an unordered list? (a) `` (b) `` (c) `` (d) `<dl>`
2. What type of list is used for items with a specific numerical order? (a) Unordered list (b) Ordered list (c) Definition list (d) Hyperlink list
3. In an HTML ordered list, what typically precedes each item? (a) Bullet points (b) Numbers or letters (c) Indentation (d) Hyperlinks
4. Which HTML element is used to define a definition list? (a) `` (b) `` (c) `` (d) `<dl>`
5. What is the purpose of an HTML definition list? (a) To create navigation menus (b) To create itemized lists (c) To present terms and their descriptions (d) To create tables
6. Which HTML element is used for embedding images directly within a web page? (a) `<image>` (b) `` (c) `<picture>` (d) `<link>`
7. What does the "alt" attribute of the `` element specify? (a) The image source URL (b) The image size (c) Alternative text for the image (d) The image caption
8. Image mapping is used for: (a) Embedding images in web pages (b) Creating clickable regions within an image (c) Changing image colors (d) Adjusting image dimensions
9. To create an image map, which HTML element is used to delineate the map? (a) `` (b) `<map>` (c) `<area>` (d) `<a>`
10. In an HTML hyperlink, what does the "href" attribute specify? (a) The image source URL (b) The destination of the link (c) The hyperlink's text (d) The color of the link

B. Fill in the blanks

1. The _____ element is used to define an unordered list in HTML.
2. Each item in an _____ list is usually prefaced by numbers or letters that signify their position.
3. To create an ordered list, you employ the _____ element.
4. _____ lists are used to represent a list of terms and their corresponding descriptions or definitions.
5. The _____ element is used to define a definition list.
6. The _____ element is used to showcase images within a web page.
7. The "alt" attribute of the `` element provides alternative text for the _____.
8. Image _____ allows you to define clickable regions on an image.
9. In HTML, you can create _____ using the `<a>` element.
10. The _____ element defines the central body of a table.

C. True or False

1. Unordered lists are primarily used for items that require a specific numerical order.
2. The `` element is used for creating ordered lists in HTML.
3. Definition lists are often used for creating navigation menus.

4. The element is used for embedding videos in a web page.
5. The "alt" attribute of the element is used for providing a title for the image.
6. Image mapping is used to define clickable regions on images, such as interactive diagrams.
7. The <map> element in image mapping specifies the clickable regions on an image.
8. The <a> element is used to create hyperlinks in HTML.
9. The "href" attribute in hyperlinks specifies the visible link text on a web page.
10. URL encoding is used to represent reserved characters in URLs to ensure their correct transmission.

D. Short Question Answers

1. What is the purpose of an unordered list in HTML, and how is it defined?
2. Explain the use of the and elements in creating an unordered list.
3. When would you use an ordered list in HTML, and how is it defined?
4. Describe the elements used to create an ordered list, including their tags.
5. What are definition lists used for, and which HTML elements are involved in creating them?
6. Explain the structure of a definition list, including the <dl>, <dt>, and <dd> elements.
7. How do you embed images in an HTML web page using the element?
8. What is the purpose of the "alt" attribute in the element, and why is it important?
9. Describe the concept of image mapping and its use in HTML.
10. How do you create hyperlinks in HTML, and which element is used for this purpose?

Session 3. CSS

Meet Ankit, a kid who loves the internet. He learned about "CSS." CSS was like magic paint for web pages. With it, he could: Change Colors, Style Fonts, Arrange Layout, Add Decorations. With CSS, Ankit made his web page look amazing, like being a digital artist. He could create fun and stylish web pages. CSS was a magic paintbrush for the internet, making it colorful and exciting. It was like having a special tool to make web pages awesome! As shown in figure 3.1.



Fig.3.1: Ankit learning CSS

In this chapter, you will understand about CSS, CSS Versions, CSS Syntax, Types of CSS, Multiple Style Sheets and Background and borders of style elements.

3.1 CSS (Cascading Style Sheets)

CSS, denoting Cascading Style Sheets, is a stylesheet language of utmost significance within the realm of web development. Its primary role is to delineate and dictate the presentation and stylistic attributes of HTML (Hypertext Markup Language) documents. As a foundational technology, CSS plays a pivotal role in exercising authority over the arrangement, aesthetics, and visual demeanour of web pages.

Did You Know?

CSS is generally used with HTML to change the style of web pages and user interfaces.

3.1.1 CSS Syntax:

The basic syntax of a CSS rule consists of a selector, a declaration block, and within the declaration block, properties and values as shown in Figure 3.2.

```
css Copy code  
  
selector {  
  property: value;  
}
```

Fig. 3.2: CSS Syntax

- **Selector:** This targets the HTML element(s) you want to style. Selectors can be made up of HTML elements (type selectors), classes, IDs, attributes, and more.

- **Declaration Block:** Enclosed in curly braces {}, this contains one or more declarations separated by semicolons;.
- **Property:** This is the style attribute you want to change (e.g., color, width, font-size).
- **Value:** This specifies the value you want to apply to the property (e.g., #333, 15px, bold).

1. Selectors: Selectors are patterns used to select and target HTML elements that you want to style. CSS rules are applied to elements based on the selectors. Here are some common types of selectors:

Element Selector: Selects all instances of a particular HTML element. For example:

```
p {
  /* Styles for all <p> elements */
}
```

Class Selector: Selects elements with a specific class attribute. Class selectors start with a dot (.). For example:

```
.highlight {
  /* Styles for elements with class="highlight" */
}
```

ID Selector: Selects an element with a specific id attribute value. ID selectors start with a hash (#). For example:

```
#header {
  /* Styles for the element with id="header" */
}
```

Descendant Selector: Selects elements that are descendants of a specified element. For example:

```
ul li {
  /* Styles for <li> elements within <ul> elements */
}
```

Pseudo-Class Selector: Selects elements in a particular state or condition. For example:

```
a:hover {
  /* Styles for links when hovered over */
}
```

Universal selector: The universal selector in CSS is denoted by an asterisk (*). It is a special selector that matches all elements in an HTML document. When you use the universal selector, it selects every element on the page, allowing you to apply styles to all elements simultaneously.

Here's an example of how the universal selector can be used in CSS:

```
/* Apply a common style to all elements on the page */
* {
  margin: 0;
  padding: 0;
  border: none;
}
```

2. Properties: Properties are the CSS attributes that define how an HTML element should be styled. Each CSS rule consists of one or more properties followed by a colon (:) and their corresponding values. Here are some common properties:

- **color:** Sets the text color.
- **font-size:** Specifies the font size.
- **background-color:** Sets the background color.
- **margin, padding:** Controls spacing around elements.
- **border:** Defines borders around elements.
- **text-align:** Aligns text within an element.
- **width, height:** Sets the width and height of elements.

3. Values: Values are assigned to properties and define how the selected elements should be styled. Values can be specific values (e.g., 12px, #FF0000), keywords (e.g., bold, center), or functional notations (e.g., rgba(255, 0, 0, 0.5) for a semi-transparent color). Values are followed by a semicolon (;) to separate multiple property-value pairs.

Here's a basic example of a CSS rule with selectors, properties, and values:

```
/* Selector: Selects all <p> elements */
p {
  /* Property-Value Pairs */
  color: blue; /* Text color set to blue */
  font-size: 18px; /* Font size set to 18 pixels */
  margin-bottom: 10px; /* Bottom margin set to 10 pixels */
}
```

Explanation: In this example, the selector `p` selects all `<p>` elements, and the properties (`color`, `font-size`, `margin-bottom`) determine how those elements are styled. Each property is followed by a colon, and its value is specified.

CSS allows you to create complex rules by combining multiple selectors, properties, and values to style web elements as desired. CSS files are often separated from HTML files, and multiple CSS rules can be defined in a single stylesheet for consistency and maintainability.

3.1.2 Key Concepts and Features of CSS:

Separation of Concerns: CSS separates the content (HTML) from its presentation, allowing web developers to create structured and semantic HTML documents while applying styling separately.

Selectors: CSS uses selectors to target HTML elements. Selectors define which elements should be styled. Common selectors include element selectors (e.g., `p`, `h1`, `a`), class selectors (e.g., `button`), and ID selectors (e.g., `#header`).

Properties: CSS properties determine how HTML elements are styled. Properties include attributes like `color`, `font-size`, `margin`, `padding`, `background-color`, and many more.

Values: CSS properties are assigned values that dictate how the selected elements should appear. For example, `color: blue;` sets the text color to blue.

Cascading: The term "cascading" in CSS refers to the order of precedence when multiple styles are applied to the same element. The cascade allows you to define styles at different levels, including user-defined styles, author styles, and browser defaults.

Inheritance: Some CSS properties are inherited from parent elements to their child elements. For example, if you set a font family on the body element, it will apply to all text within that element unless overridden by specific styles.

Selectors and Combinators: CSS provides a wide range of selectors and combinators for targeting specific elements or groups of elements. These include descendant selectors, child selectors, adjacent sibling selectors, and more.

Responsive Design: CSS is crucial for creating responsive web designs that adapt to different screen sizes and devices. Media queries are commonly used to apply styles based on viewport dimensions.

Various CSS Frameworks: Bootstrap, Foundation, TailwindCSS, Materialize and SemanticUI etc.

Example of CSS:

Here's a simple example of CSS code that styles a paragraph element: **/* CSS code in an external stylesheet (styles.css) */**

```
p {
  color: #333; /* Text color (dark gray) */
  font-size: 16px; /* Font size (16 pixels) */
  line-height: 1.5; /* Line height (1.5 times the font size) */
  margin-bottom: 20px; /* Bottom margin (20 pixels) */
}
```

In this example, the CSS code targets all <p> elements on the web page and sets their text color, font size, line height, and margin. These styles are defined in an external style sheet file named "styles.css" and linked to the HTML document using the <link> element.

CSS is a versatile and powerful language for creating visually appealing and responsive web designs. It plays a vital role in web development by providing control over the layout and presentation of web content.

3.2 CSS Versions

- CSS has evolved over the years, with different versions and modules. Some key versions include:
- CSS1: The first version of CSS introduced basic styling properties and selectors.
- CSS2: Added more advanced selectors, positioning, and support for media types.
- CSS3: A modular version with various modules like CSS3 Selectors, CSS3 Color, CSS3 Text, CSS3 Backgrounds, and more. It introduced features like gradients, transitions, and animations.
- CSS4: An informal term used for the ongoing development of CSS. It includes newer features and enhancements. It hasn't been released to the public yet.

Did You Know?

There are three versions of CSS:

CSS1: The oldest version, dating back to 1996

CSS2: Released in 1998

CSS3: The latest version, containing new features and functionalities

Assignment 3.1.

List down the CSS syntax of the following:

- Descendant Selector
- ID Selector
- Pseudo-Class Selector

3.3 CSS Syntax

CSS (Cascading Style Sheets) has a straightforward syntax that consists of selectors, properties, and values. Let's explore each part of the CSS syntax in more detail:

Did You Know?

Inline CSS used for quick and specific styling. Inline CSS is processed faster than external CSS.

3.4 Types of CSS

Let's delve into the concepts of internal CSS and inline CSS, both of which are used to apply styles to HTML elements but differ in where and how they are implemented as shown in Figure 3.3.

- 1. Inline CSS:** Inline CSS entails the application of styles directly to individual HTML elements via the use of the "style" attribute. This method operates as follows:

Usage: You add the style attribute to an HTML element and define CSS rules as attribute values. These styles are applied only to the specific element.

Scope: Inline CSS styles are limited to the element on which they are applied.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inline CSS Example</title>
</head>
<body>
  <h1 style="color: green; font-family: Arial, Helvetica, sans-serif;">This is a styled heading.</h1>
  <p style="color: blue; font-size: 18px;">This is a styled paragraph.</p>
</body>
</html>
```

Fig. 3.3: Types of CSS

Pros of Inline CSS:

- Provides granular control over individual elements' styles.
- Useful for quick, one-off styling for specific elements.
- Overrides external and internal CSS styles if needed.

Cons of Inline CSS:

- Leads to code repetition if the same styles are applied to multiple elements.
- Not practical for maintaining consistent styles across an entire website.
- Can make HTML markup less clean and harder to read.

2. Internal CSS: Internal CSS, also referred to as embedded CSS or document-level CSS, encompasses the practice of specifying CSS styles directly within the HTML document. This method typically involves placing the CSS definitions within the HTML file, often within the <head> section by employing a <style> element. To further elucidate its functioning as shown in Figure 3.4.

Did You Know?

Internal CSS used for multiple elements within the same HTML document. Internal CSS is also known as embedded cascading style sheet.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Internal CSS Example</title>
  <style>
    /* CSS rules for styling paragraphs */
    p {
      color: ■blue;
      font-size: 18px;
    }
    /* Additional CSS rules for other elements */
    h1 {
      color: ■green;
      font-family: Arial, Helvetica, sans-serif;
    }
  </style>
</head>
<body>
  <h1>This is a styled heading.</h1>
  <p>This is a styled paragraph.</p>
</body>
</html>

```

Fig. 3.4: Internal CSS

Pros of Internal CSS:

- Keeps CSS styles within the same HTML document, making it easy to maintain.
- Allows for reusable styles applied to multiple elements in the document.
- Useful for small to medium-sized projects where a single document contains all styles.

Cons of Internal CSS:

- Not as organized as external CSS for larger projects.
- Styles are not reusable across multiple HTML documents.

- Mixing content and styles in the same document can become cluttered in complex projects.
- 3. External CSS:** External CSS encompasses the creation of a distinct CSS file and the establishment of a connection with an HTML document as shown in Figure 3.5.

Did You Know?

External CSS contains separate CSS files that contain only style properties.

The following is a detailed set of instructions along with explanations for employing external CSS:

Step 1: Create Your CSS File

Start by creating a new CSS file with a .css extension. You can do this using a text editor or code editor of your choice, such as Visual Studio Code, Notepad, or Sublime Text. Name the file something descriptive, like styles.css, to make it easy to identify.

Step 2: Define CSS Styles in the External File

Inside your CSS file (styles.css in this case), you can define CSS rules just like you would in an internal <style> element or inline style attributes. For example:

```
/* styles.css */
/* Define CSS rules for paragraphs */
p {
    color: blue;
    Font-size: 18px;
}
/* Define CSS rules for headings */
h1, h2 {
    color: green;
    font-family: Arial, sans-serif;
}
```

In this example, we've defined CSS rules for paragraphs and headings. The p selector targets all <p> elements, setting their text color and font size. The h1, h2 selector targets both <h1> and <h2> elements, changing their text color and font family.

Step 3: Save Your CSS File

After defining your CSS styles in the external file

Step 4: Create Your HTML File

Begin by creating a new HTML file or using an existing one where you want to apply the external CSS styles. You can use a plain text editor like Notepad, Visual Studio Code, or any other code editor to create and edit HTML files.

Step 5: Save Your HTML File

Save your HTML file with a .html extension. Choose a descriptive name for your HTML file, such as index.html, so you can easily identify it.

Step 6: Place the CSS File in the Same Directory

Ensure that your CSS file (styles.css) is saved in the same directory as your HTML file. This is a common practice for organizing related files.

Step 7: Link the CSS File to Your HTML Document

Inside your HTML file, you'll use the <link> element within the <head> section to connect your CSS file to the HTML document. Here's the code to do that:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Web Page</title>
  <!-- Link your external CSS file -->
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <!-- Your HTML content goes here -->
  <h1>This is a heading.</h1>
  <p>This is a paragraph.</p>
</body>
</html>

```

Fig. 3.5: External CSS

Explanation: Explanation of the <link> element attributes:

- rel: Specifies the relationship between the HTML document and the linked resource. In this case, stylesheet indicates that the linked file is a stylesheet.
- type: Defines the MIME type of the linked resource. For CSS, use "text/css".
- href: Specifies the path to the external CSS file. In this example, "styles.css" is in the same directory as the HTML file, so you can use a relative path.

Step 7: Save Your HTML File

After linking the CSS file in your HTML document, save the HTML file again.

Step 8: Open the HTML File in a Web Browser

You can now open your HTML file using a web browser to see how the external CSS styles are applied to your HTML content.

By following these steps, you have successfully linked an external CSS file to your HTML document, allowing you to maintain a separation of content and presentation and apply consistent styling across multiple pages.

Assignment 3.2.

- List down the pros and cons of Inline CSS.
- List down the steps of External CSS.
- List down the pros and cons of Internal CSS
- Write down the code of Internal CSS.

When to Use Each:

Use internal CSS when you want to apply styles to multiple elements within the same HTML document and maintain some organization. It's suitable for small to medium-sized projects.

Use inline CSS for quick, one-time styling adjustments to specific elements when external or internal stylesheets aren't necessary or when you need to override existing styles temporarily.

In practice, most web development projects use a combination of external, internal, and inline CSS, depending on the project's size, complexity, and specific styling needs.

3.5 Multiple Style Sheets

Using multiple style sheets in your HTML document allows you to organize and apply different sets of styles to various elements or parts of your webpage. Here's how to use multiple style sheets in your HTML document.

Step 1: Create Your CSS Files

Start by creating the separate CSS files that contain your different sets of styles. Each CSS file should have a unique purpose or target specific elements on your webpage. Name them descriptively to make it clear what each file does. For example:

- styles.css: Contains general styles for the entire page.
- header.css: Contains styles for the header section.
- sidebar.css: Contains styles for the sidebar.
- footer.css: Contains styles for the footer.

Step 2: Link Your CSS Files in the HTML Document

In your HTML document, within the <head> section, use multiple <link> elements to link each CSS file. Each <link> element should reference a different CSS file using the href attribute as shown in Figure 3.6.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Multiple External Stylesheets Example</title>
  <link rel="stylesheet" href="style.css">
  <link rel="stylesheet" href="header.css">
  <link rel="stylesheet" href="sidebar.css">
  <link rel="stylesheet" href="footer.css">
</head>
<body>
  <!-- Your HTML content goes here -->
  <header>
    <h1>Welcome to My Website</h1>
  </header>

  <nav>
    <!-- Sidebar Content -->
  </nav>

  <main>
    <!-- Main Content -->
  </main>

  <footer>
    <!-- Footer Content -->
  </footer>
</body>
</html>

```

Fig. 3.6: Link Your CSS Files in the HTML Document

Step 3: Organize and Define Styles in CSS Files

In each of your CSS files (styles.css, header.css, sidebar.css, footer.css), you can define styles relevant to their respective sections or elements.

Example:

```

/* styles.css */
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0;
}
/* header.css */
header {
  background-color: #333;
  color: #fff;
}
/* sidebar.css */
nav {

```

```
background-color: #555;
color: #fff;
}
/* footer.css */
footer {
background-color: #333;
color: #fff;
text-align: center;
}
```

Each CSS file contains styles that target specific parts of the webpage. In this example, we're styling the body, header, navigation (sidebar), and footer separately.

Step 4: Save and Preview Your Webpage

Save your HTML document along with the CSS files in the same directory. When you open the HTML file in a web browser, it will apply the styles from all linked CSS files to the respective parts of your webpage.

Using multiple style sheets in this way helps maintain clean and organized code, making it easier to manage and update styles for different sections of your website. It also promotes reusability and consistency in your design.

3.6 Value lengths and percentage

In CSS, you can specify values for various properties using different units, including lengths and percentages. These units are essential for defining sizes, dimensions, and positions of elements on a web page. Let's explore the concepts of value lengths and percentages in CSS:

1. Length Values: Length values represent measurements of size or distance. CSS supports several length units, each serving specific purposes:

- **Pixels (px):** Pixels are a common unit for screen-based design. They provide a fixed and consistent size regardless of the user's screen resolution. For example, `width: 200px;` sets the width of an element to 200 pixels.
- **Em (em):** The "em" unit in CSS is a measurement relative to the font size of the parent element. For example, if the font size of a parent element is 16 pixels, applying a style of `font-size: 1.5em;` to a child element would result in that child element having a font size of 24 pixels.
- **Rem (rem):** Similar to "em," but relative to the font size of the root (<html>) element. This ensures consistent scaling based on the root font size.
- **Centimeters (cm), Millimeters (mm), Inches (in):** These units are used for print styles and physical media. For example, `width: 2cm;` sets the width to 2 centimeters.
- **Points (pt):** Points are typically used in print media. One point is approximately 1/72 of an inch. For example, `font-size: 12pt;` sets the font size to 12 points.
- **Picas (pc):** Picas are used in typography and equal 12 points. For example, `line-height: 1.5pc;` sets the line height to 18 points.

2. Percentage Values: Percentage values represent a portion of a parent element's size or another relative reference point. They are often used for responsive design and scaling elements relative to their container. Common uses include:

- **Width and Height:** You can set the width or height of an element as a percentage of its parent element's dimensions. For example, `width: 50%`; makes an element occupy half of its parent's width.
- **Margins and Padding:** You can specify margins and padding in percentages to create responsive spacing. For instance, `margin: 5%`; creates a margin equal to 5% of the element's parent width.
- **Font Size:** Setting font sizes in percentages allows text to scale proportionally. For example, `font-size: 120%`; increases the font size by 20% relative to the parent's font size.

Example:

Here's an example that combines both length and percentage values to create a responsive layout:

```
.container {
  width: 80%; /* 80% of the parent's width */
  margin: 0 auto; /* Center the container horizontally */
  padding: 10px; /* 10 pixels of padding inside the container */
  font-size: 16px; /* 16 pixels font size */
}
.box {
  width: 200px; /* 200 pixels width */
  margin: 2em; /* 2 times the font size of the parent as margin */
}
```

In this example, the container's width is set to 80% of its parent's width, and the box's width is fixed at 200 pixels. The margin for the box is specified in "em" units, which are relative to the parent's font size.

Using length and percentage values strategically helps create flexible and responsive web designs that adapt to different screen sizes and devices.

3.7 Background and borders of style elements,

3.7.1 Background Styles:

Background Color (background-color): You can set the background color of an element using the `background-color` property. This property takes a color value as its argument. For example:

```
.box {
  background-color: #f0f0f0; /* Sets the background color to light gray */
}
```

Background Image (background-image): You can use a background image instead of a solid color. The `background-image` property allows you to specify an image URL. For example:

```
.header {
  background-image: url('header-bg.jpg'); /* Sets a background image */
}
```

Background Repeat (background-repeat): You can control how the background image repeats with the `background-repeat` property. Common values include `repeat` (default), `no-repeat`, `repeat-x`, and `repeat-y`.

```
.background {
```



```
background-image: url('pattern.png');
background-repeat: repeat-x; /* Repeat the image horizontally */
}
```

Background Size (background-size): This property determines how the background image is sized within its container. You can use values like cover, contain, or specific dimensions.

```
.cover-image {
background-image: url('cover-image.jpg');
background-size: cover; /* Scales the image to cover the entire element */
}
```

3.7.2 Border Styles:

Border Width (border-width): You can set the width of an element's border using the border-width property. It accepts values like thin, medium (default), thick, or specific lengths like 2px as shown in Figure 3.7.

```
.bordered-element {
border-width: 2px; /* Sets a 2-pixel border width */
}
```

Border Color (border-color): The border-color property defines the color of the border. You can specify a color value.

```
.bordered-element {
border-color: #333; /* Sets a dark gray border color */
}
```

Border Style (border-style): Use the border-style property to set the style of the border, such as solid, dotted, dashed, double, etc.

```
.bordered-element {
border-style: dashed; /* Sets a dashed border style */
}
```

Border Radius (border-radius): This property rounds the corners of an element, creating a border with curved edges.

```
.rounded-element {
border-radius: 10px; /* Rounds all four corners with a 10-pixel radius */
}
```

Border Shorthand (border): You can use the border shorthand property to set border properties (width, style, color) in one declaration.

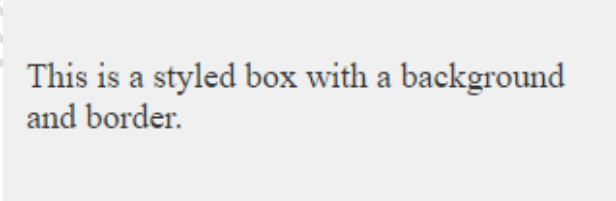
```
.bordered-element {
border: 2px dashed #333; /* Sets border width, style, and color in one line */
}
```

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Background and Border Example</title>
  <style>
    /* Styling for a container with background and border */
    .styled-box {
      width: 300px;
      height: 150px;
      background-color: #f0f0f0; /* Light gray background color */
      border: 2px solid #333; /* Dark gray solid border */
      border-radius: 10px; /* Rounded corners with a 10-pixels radius */
      padding: 20px; /* Padding inside the container */
    }
    /* Text styles within the container */
    .styled-box p {
      color: #333; /* Text color */
      font-size: 18px; /* Font size */
    }
  </style>
</head>
<body>
  <div class="styled-box">
    <p>This is a styled box with a background and border.</p>
  </div>
</body>
</html>

```

Fig. 3.7: Border Styles

Output-


This is a styled box with a background and border.

Fig. 3.8: Output for Border Styles

Assignment 3.3.

- Write down the syntax of the following background elements:
 - Background Repeat
 - Background Color
- Write down the syntax of the following border style elements:
 - Border Color
 - Border Style

3.8 Font related properties

In CSS, you can control various aspects of fonts and text styling using font-related properties. These properties allow you to specify the font family, size, weight, style, and more. Here are some commonly used font-related properties:

Font Family (font-family): This property defines the typeface or font family for text. You can specify multiple fonts as a fallback in case the preferred font is not available on the user's system.

```
body {
  font-family: Arial, Helvetica, sans-serif;
}
```

Font Size (font-size): The font-size property sets the size of the font. You can use various units such as pixels (px), ems (em), percentages (%), or keywords like small, medium, large, etc.

```
h1 {
  font-size: 24px;
}
```

Font Weight (font-weight): This property adjusts the thickness or weight of the font. Common values include normal, bold, bolder, and numeric values (e.g., 400, 700).

```
strong {
  font-weight:
}
```

Font Style (font-style): The font-style property is used to specify whether the font should be displayed in a normal, italic, or oblique style.

```
em {
  font-style: italic;
}
```

Line Height (line-height): The line-height property determines the amount of vertical space between lines of text. It can be set as a unit value, a percentage, or a keyword.

```
p {
  line-height: 1.5; /* Sets line height to 1.5 times the font size */
}
```

Text Decoration (text-decoration): The text-decoration property controls the decoration of text, such as underlines, overlines, and strikethroughs.

```
a {
  text-decoration: none; /* Removes text decoration from links */
}
```

Text Transform (text-transform): This property changes the capitalization of text, making it uppercase, lowercase, or capitalized (each word capitalized).

```
.uppercase {
  text-transform: uppercase; /* Converts text to uppercase */
}
```

Text Color (color): The color property specifies the color of the text. It accepts various color values, including hexadecimal, RGB, and color names.

```
h2 {
```

```
color: #336699; /* Sets text color to a shade of blue */
}
```

Letter Spacing (letter-spacing): This property adjusts the space between characters in text. You can specify a positive or negative value.

```
.spaced-text {
  letter-spacing: 2px; /* Increases letter spacing by 2 pixels */
}
```

Assignment 3.4.

- List down the syntax of the following font related properties:
 - Font Size
 - Font Style
 - Text Transform
 - Text Decoration

3.9 Layout the table data using CSS List Table

Let us look at a simple HTML table with just some data in it, i.e., with no CSS styling added at all as shown in Figure 10.9.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <table>
    <caption>Table of Indian States and its Capitals</caption>
    <thead>
      <tr>
        <th>SN</th>
        <th>State</th>
        <th>Capital</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>Uttar Pradesh</td>
        <td>Lucknow</td>
      </tr>
      <tr>
        <td>2</td>
        <td>Madhya Pradesh</td>
        <td>Bhopal</td>
      </tr>
      <tr>
        <td>3</td>
        <td>Maharashtra</td>
        <td>Mumbai</td>
      </tr>
    </tbody>
  </table>
</body>
</html>

```

Fig. 3.9: Layout the table data using CSS List Table

Output

Table of Indian States and its Capitals		
SN	State	Capital
1	Uttar Pradesh	Lucknow
2	Madhya Pradesh	Bhopal
3	Maharashtra	Mumbai

Fig. 3.10: Output for Layout the table data using CSS List Table

In order to style tables with CSS, we usually don't put any additional class or id. Instead, we directly apply the style to the HTML element using the CSS properties. In case we need to provide some special styles like colouring only a single cell or some special effect to a particular cell, then we may need to provide some class or id.

CSS code to add some table styles:

CSS offers a variety of properties that enable you to manage the visual presentation and structure of tables. These properties are typically used with HTML elements such as tables, table rows, table headers, and table cells. Here are some frequently used CSS properties for customizing tables as shown in Figure 3.3.

Border Collapse (border-collapse): This property controls how table borders are collapsed. The two common values are collapse (borders are collapsed into a single border) and separate (each cell has its own border).

```
table {
    border-collapse: collapse; /* Collapse borders into a single border */
}
```

Border Spacing (border-spacing): This property sets the space between adjacent cell borders when border-collapse is set to separate. It takes two values: horizontal and vertical spacing.

```
table {
    border-collapse: separate;
    border-spacing: 5px 10px; /* Sets 5px horizontal and 10px vertical spacing */
}
```

Width (width) and Height (height): These properties control the width and height of the table.

```
table {
    width: 100%; /* Makes the table width 100% of its container */
    height: 200px; /* Sets the table height to 200 pixels */
}
```

Text Alignment (text-align): This property sets the horizontal alignment of text within table cells. Common values are left, center, right, and justify.

```
th, td {
    text-align: center; /* Center-align text in table cells */
}
```

Vertical Alignment (vertical-align): This property controls the vertical alignment of content within table cells. Common values include top, middle, bottom, and baseline.

```
th, td {
    vertical-align: middle; /* Vertically center content in table cells */
}
```

Padding (padding): The padding property adds space between the content and the border of table cells.

```
th, td {
    padding: 10px; /* Adds 10px of padding inside table cells */
}
```

Font Size (font-size): This property sets the font size for text within table cells.

```
th, td {
    font-size: 16px; /* Sets font size to 16 pixels */
}
```


Background Color (background-color): You can set the background color of table elements, including table rows, table headers, and table cells.

```
th {
    background-color: #f0f0f0; /* Sets background color for table headers */
}
td {
    background-color: #ffffff; /* Sets background color for table cells */
}
```

These CSS properties allow you to control the layout, appearance, and styling of tables in your web pages, making it possible to create visually appealing and well-structured tables.

```
<!DOCTYPE html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Table Example</title>
  <style>
    table {
      border-collapse: collapse;
      width: 100%;
      border: 2px solid #333;
    }
    th, td {
      border: 1px solid #333;
    }
    th {
      background-color: #f0f0f0;
      font-weight: bold;
    }
    td {
      background-color: #ffffff;
      font-size: 16px;
    }
    /* Alternating row colors for better readability */
    tr:nth-child(even) {
      background-color: #f9f9f9;
    }
  </style>
</head>
<body>
  <table>
    <caption>Table of Indian States and its Capitals</caption>
    <thead>
      <tr>
        <th>SN</th>
        <th>State</th>
        <th>Capital</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>Uttar Pradesh</td>
        <td>Lucknow</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

Fig. 3.11: CSS code to add some table styles

OUTPUT

Table of Indian States and its Capitals		
SN	State	Capital
1	Uttar Pradesh	Lucknow

Fig. 3.12: Output for CSS code to add some table styles

SUMMARY

- CSS is crucial for web development, as it controls the presentation and style of HTML documents.
- Key concepts include separation of concerns, selectors, properties, values, cascading, and inheritance.
- CSS provides various selectors and combinators to target specific elements.
- CSS can be linked as an external stylesheet, applied internally, or used inline.
- Length and percentage values are used to define sizes and positions of elements.
- CSS allows you to style backgrounds and borders, specify font properties, and manage table layouts.

Check Your Progress**A. Multiple-Choice Questions (MCQs):**

1. What does CSS stand for? (a) Computer Style Sheets (b) Cascading Style Sheets (c) Creative Style Sheets (d) Centralized Style Sheets
2. CSS primarily defines the _____ of HTML documents. (a) Structure (b) Presentation (c) Behavior (d) Interaction
3. Which CSS property is used to set the background color of an element? (a) Color (b) Font-size (c) Background-color (d) Margin
4. The CSS property that determines how text should be aligned within an element is: (a) text-align (b) line-height (c) font-style (d) background-color
5. Which unit of measurement is relative to the font size of the parent element in CSS? (a) px (b) em (c) cm (d) in
6. Which CSS property rounds the corners of an element, creating a border with curved edges? (a) border-width (b) border-color (c) border-style (d) border-radius
7. What CSS property controls the vertical alignment of content within table cells? (a) text-align (b) vertical-align (c) letter-spacing (d) padding
8. To link an external CSS file to an HTML document, which HTML element is used? (a) <style> (b) <link> (c) <script> (d) <head>
9. Which CSS property specifies the size of background images within an element? (a) background-repeat (b) background-image (c) background-size (d) background-color
10. In CSS, what does the "cascading" refer to? (a) The process of folding elements (b) The order of precedence for applying styles (c) The way fonts cascade from large to small (d) The animation of text

B. Fill in the blanks

1. CSS primarily focuses on the _____ and stylistic attributes of HTML documents.
2. The universal selector in CSS is denoted by an _____.
3. Selectors in CSS define which _____ should be styled.

4. The CSS property that determines how text should be aligned within an element is _____.
5. The "em" unit in CSS is a measurement relative to the font size of the _____ element.
6. The CSS property that adjusts the space between characters in text is _____.
7. CSS provides various selectors and combinators for targeting specific elements or groups of elements, including _____ selectors.
8. CSS is crucial for creating _____ web designs that adapt to different screen sizes.
9. The <link> element is used to connect an external CSS file to an HTML document within the _____ section.
10. _____ CSS is the practice of specifying CSS styles directly within the HTML document.

C. True or False

1. CSS primarily focuses on the structure of HTML documents.
2. The "cascading" in CSS refers to the process of folding elements in a document.
3. The "em" unit in CSS is a measurement relative to the font size of the root (<html>) element.
4. Inline CSS is the best practice for maintaining consistent styles across an entire website.
5. CSS properties determine how HTML elements are structured and organized.
6. External CSS is used to apply styles directly within the HTML document.
7. CSS allows for creating responsive web designs that adapt to different screen sizes and devices.
8. The <link> element is used to connect an external CSS file to an HTML document within the <body> section.
9. The "rem" unit in CSS is an informal measurement that varies based on the context.
10. Multiple style sheets cannot be used in a single HTML document.

D. Short Question Answers

1. What is the primary purpose of CSS in web development?
2. How does CSS separate concerns in web development?
3. Explain the concept of "selectors" in CSS and provide examples.
4. What are some common CSS properties used for styling elements?
5. Describe the meaning of "cascading" in the context of CSS.
6. How does inheritance work for CSS properties?
7. What are media queries, and why are they important in CSS?
8. How can you link an external CSS file to an HTML document?
9. What are the benefits and drawbacks of using internal CSS?
10. When would you use inline CSS in web development?

Session 4. CSS Box Model

Atul, a curious kid, learned about the "CSS Box Model," which helps arrange things on a web page, like a puzzle. It has four parts: Content (where text and images go), Padding (space around content), Border (a frame around content), and Margin (empty space around the whole puzzle). With this model, Atul could design web pages just how he liked, making them organized and attractive, like a puzzle solver and a web page designer all at once. It made the internet more interesting and appealing to explore. As shown in figure 4.1.



Fig. 4.1: Atul using CSS tools for website

In this chapter, you will learn about the CSS box model, CSS Dimensions, CSS to display positioning, CSS Visibility, CSS Display and CSS Scrollbars.

4.1 CSS box model

The CSS box model is a fundamental concept in web design and layout, dictating how elements on a web page are structured and how their dimensions are calculated. This model encompasses four primary components:

1. **Content:** The innermost section of the box is referred to as the content area and it holds the actual content of the element, which can include text, images, or other HTML elements. The dimensions of the content area are determined by the width and height properties.
2. **Padding:** Padding is an invisible area that surrounds the content, and its size is specified using the padding property. Padding creates space between the content and the border of the element. You can set padding values for the top, right, bottom, and left sides individually, or use a single value that applies to all sides uniformly.
3. **Border:** The border encompasses the padding and content of an element, and it's configured using the border property. It forms a visible perimeter around the element, allowing you to define attributes like border width, style, and color.
4. **Margin:** Margins represent an outer, transparent area surrounding the element's border, and their size is specified using the margin property. Margins establish space between the element and adjacent elements on the page. Similar to padding, margin values can be defined for individual sides or as a single value applying to all sides consistently.

Did You Know?

The CSS box model is used to create the design and layout of web pages.

Here's an example of how to use the box model in CSS as shown in code.

```

/* Define the content width and height */
.box {
/* Setup dimensions */
width: 200px;
height: 100px;
/* Add padding */
padding: 10px;
/* Add a border */
border: 2px solid #333;
/* Add margins*/
margin: 20px;

```

In this example, we have a box element with specific dimensions, padding, a border, and margins. The total dimensions of the box element are calculated based on these properties, according to the box model as shown in Figure 4.2.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Box Model Example</title>
  <style>
    /* Define Styling for the box */
    .box {
      width: 200px;
      height: 100px;
      padding: 20px;
      border: 2px solid ■ #333;
      margin: 30px;
      background-color: □ #fefefe;
      text-align: center;
      line-height: 100px;
      font-size: 18px;
    }
    /* Defining Styling for the container */
    .container {
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box">
      This is a box.
    </div>
  </div>
</body>
</html>

```

Fig. 4.2: CSS box model

Output

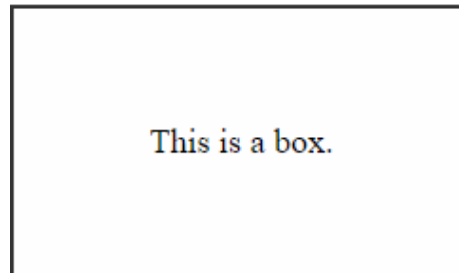


Fig. 4.3: Output for CSS box model

In this example:

- We create a div element with the class `box`, which represents a box-like element.
- CSS styles are applied to this box to demonstrate the CSS box model properties:
 - `width` and `height` set the dimensions of the content area.
 - `padding` adds space inside the content area.
 - `border` adds a border around the padding.
 - `margin` creates space around the entire box.
 - `background-color`, `text-align`, `line-height`, and `font-size` are used for additional styling.
- The `.container` class is used to center-align the box on the page.
- The text "This is a box." is placed inside the box.

You can copy and paste this code into an HTML file and open it in a web browser to see how the CSS box model works in practice. Feel free to adjust the values and experiment with different styles to learn more about how the box model affects the layout of HTML elements.

4.2 CSS Dimensions

CSS provides several properties for controlling the dimensions of elements. The key dimension-related properties include `width`, `height`, `min-width`, `max-width`, `min-height`, and `max-height`. Here's a breakdown of each property:

width and height: These properties set the width and height of an element's content box, which includes the content but excludes padding, border, and margin. You can specify dimensions using various units, such as pixels (px), percentages (%), ems (em), and more.

```
.element {  
  width: 300px;  
  height: 150px;  
}
```

min-width and min-height: These properties define the minimum width and height that an element can be. Even if content inside the element would normally require a smaller size, the element will not become smaller than these specified minimum values.

```
.element {  
  min-width: 100px;  
  min-height: 50px;  
}
```


max-width and max-height: These properties set the maximum width and height an element can have. If content would naturally make the element larger, it won't exceed these specified maximum values.

```
.element {
  max-width: 500px;
  max-height: 300px;
}
```

With these dimension-related properties, you can precisely dictate the size and constraints of elements on your web page. This level of control allows you to craft responsive designs that adjust to various screen sizes, ensure that content remains within defined limits, and maintain the desired overall layout and appearance of your web pages as shown in Figure 4.4.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Dimensions Example</title>
  <style>
    /* Define a container with fixed width and height */
    .container {
      width: 300px;
      height: 200px;
      background-color: #f0f0f0;
      text-align: center;
    }
    /* define a box with minimum and maximum dimensions */
    .box {
      min-width: 100px;
      max-width: 250px;
      min-height: 50px;
      max-height: 150px;
      background-color: #3498db;
      color: #ffffff;
      padding: 10px;
      margin: 20px auto;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box">
      This box has dimension constraints.
    </div>
  </div>
</body>
</html>
```

Fig. 4.4: CSS Dimensions

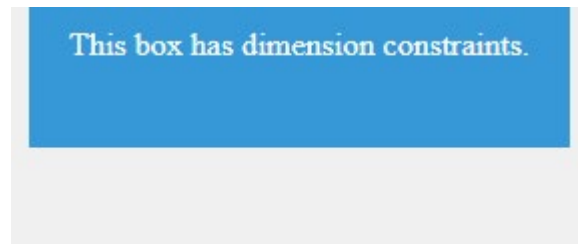
Output

Fig. 4.5: Output for CSS Dimensions

Assignment 4.1.

- List down the name of CSS box model primary components.
- List down the properties name of CSS dimensions.

4.3 CSS to display positioning

CSS offers a variety of properties that allow you to manage the placement and appearance of elements on your web page. Below are some frequently utilized CSS properties associated with positioning and display:

position: This property specifies the positioning method used for an element. It can take values like static (default), relative, absolute, fixed, or sticky. These values determine how an element is positioned within its containing element or the viewport.

```
.element {
  position: relative; /* Use relative positioning */
}
```

top, right, bottom, left: These properties are used in conjunction with the position property to precisely position an element. They specify the distance from the top, right, bottom, or left edge of the containing element.

```
.element {
  position: absolute; /* Use absolute positioning */
  top: 20px;
  left: 30px;
}
```

display: This property controls how an element is displayed in the document flow. Common values include block, inline, inline-block, none, and others. It affects the behavior of other layout properties.

```
.element {
  display: inline-block; /* Display as an inline block element */
}
```

float: The float property is used to make an element float to the left or right within its containing element. It's commonly used for creating layouts with multiple columns or wrapping text around images.

```
.element {
  float: left; /* Float the element to the left */
}
```

clear: The clear property is used to prevent an element from floating next to a floating element. It can be applied to elements following a floating element to ensure they start below it.

```
.clear-element {
  clear: both; /* Clear both left and right floated elements */
}
```

4.4 CSS Visibility (**visibility**)

The visibility property controls the visibility of an element. It can take one of the following values:

- visible (default): The element is visible.
- hidden: The element is hidden but still occupies space on the page.
- collapse: Used with table rows (<tr>) and row groups (<tbody>, <thead>, <tfoot>). It removes a row or row group but still maintains the table's layout.

Example:

```
.hidden-element {
  visibility: hidden; /* Hide the element, but it still occupies space */
}
```

4.5 CSS Display (**display**)

The display property controls how an element is rendered on the page. It can take various values, including:

- block: Renders the element as a block-level element, taking up the full width of its container.
- inline: Renders the element as an inline-level element, flowing within the content.
- inline-block: Combines aspects of both block and inline elements.
- none: Completely hides the element, and it does not occupy space on the page.
- table, table-row, table-cell: Used for creating table-like layouts.

Example:

```
.block-element {
  display: block; /* Render as a block-level element */
}
```

Difference Between {visibility: hidden;} and {display: none;}

Visibility hidden only hides the element on the page but the size of element still acquires space on the page whereas setting up display to none completely hides the element including its space acquisition.

4.6 CSS Scrollbars (**overflow, overflow-x, overflow-y**)

The overflow property controls what happens when the content of an element overflows its allocated space. It can take values like visible, hidden, scroll, or auto as shown in Figure 4.6.

- visible (default): No scrollbars are displayed, and overflowing content is visible outside the element.
- hidden: Content that overflows is clipped and not visible.
- scroll: Always display scrollbars, even if content doesn't overflow (creates space for scrollbars).
- auto: Display scrollbars only when content overflows.

Example:

```
.scrollable-element {
  overflow: scroll; /* Always show both horizontal and vertical scrollbars */
}
```

Assignment 4.2.

- List down the syntax of the following CSS properties associated with positioning and display:
 - Display
 - Clear
 - Position
- List down the properties of CSS Display.
- List down the properties of CSS Scrollbars

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Visibility, Display, Scrollbars Example</title>
  <style>
    /* CSS Visibility */
    .hidden-element {
      visibility: hidden;
    }
    /* CSS Display */
    .block-element {
      display: block;
    }
    .inline-element {
      display: inline;
    }
    .none-element {
      display: none;
    }
    /* CSS Scrollbars */
    .scroll-container {
      width: 200px;
      height: 150px;
      overflow-x: hidden;
      overflow-y: scroll;
      border: 1px solid #ccc;
    }
  </style>
</head>
<body>
  <div class="hidden-element">This is a hidden element.</div>
  <div class="block-element">This is a block level element.</div>
  <div class="inline-element">This is an inline level element.</div>
  <div class="none-element">This is a hidden element.</div>
  <div class="scroll-container">
    <p>This is a scrollable container. If content overflows, scrollbars will appear.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipiscing elit. Omnis numquam consequatur excepturi.
      Saepe a sunt vero voluptas, aliquid iure fugiat.
      Ex dolore ab exercitationem culpa dicta distinctio molestias fuga aliquam!
    </p>
  </div>
</body>
</html>
```

Fig. 4.6: CSS Scrollbars

Output

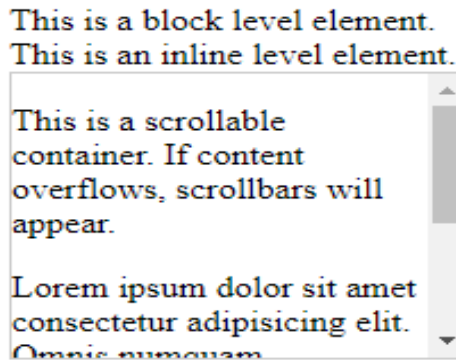


Fig. 4.7: Output for CSS Scrollbars

4.7 CSS Float property

In CSS, the "float" property serves to govern the horizontal alignment of an element within its containing element. When an element is floated, it is removed from the regular document flow and positioned to the left or right of its containing element. This arrangement enables other elements to flow around it.

The float property can have the following values:

1. left: The element is floated to the left, and content flows to its right side.
2. right: The element is floated to the right, and content flows to its left side.
3. none (default): The element is not floated, and content flows as usual.
4. inherit: The element inherits the float property from its parent element.

Did You Know?

The float property is commonly used to wrap text around images.

Here's an example that demonstrates the use of the float property as shown in Figure 4.8.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Float Property Example</title>
  <style>
    /* Define a container */
    .container {
      width: 300px;
      background-color: #f0f0f0;
    }
    /* Define a floated element */
    .floated {
      width: 100px;
      height: 100px;
      background-color: #3498db;
      float: left;
    }
    /* Define a non-floated element */
    .non-floated {
      width: 100px;
      height: 100px;
      background-color: #e74c3c;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="floated">
      Floated element
    </div>
    <div class="non-floated">
      Non-Floated element
    </div>
  </div>
</body>
</html>

```

Fig. 4.8: CSS Float property

Output



Fig. 4.9: Output for CSS Float property

In this example:

- We have a container with a fixed width.
- Inside the container, there are two elements: `.floated` and `.non-floated`.
- The `.floated` element is floated to the left using `float: left;`. As a result, the non-floated element flows to the right of the floated element.
- The non-floated element is not floated and remains in the normal document flow.

You can replicate this code by copying and pasting it into an HTML file. Afterward, you can open the file in a web browser to observe how the "float" property influences the positioning of elements.

Did You Know?

The float property in CSS is used to position elements to the left or right of their container.

SUMMARY

- The CSS box model consists of four components: content, padding, border, and margin, which define an element's layout.
- The content area holds the actual content and is sized using width and height properties.
- Padding creates space between content and the border, while the border defines its style, width, and color.
- Margins provide space outside the border and separate elements on the page.
- CSS dimensions can be controlled with properties like width, height, min-width, max-width, min-height, and max-height.
- The "position" property defines element positioning, with values like static, relative, absolute, fixed, and sticky.
- Properties like top, right, bottom, left are used to precisely position elements.
- "Display" property controls how elements are rendered, with options like block, inline, inline-block, and none.
- The "float" property is used to make elements float left or right.
- "Visibility" property controls element visibility with values like visible, hidden, and collapse.
- "Overflow" property handles content overflow with values like visible, hidden, scroll, and auto.
- Float property influences the horizontal alignment of elements within their containing element.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS (MCQ)

1. What is the primary function of the CSS box model? (a) To define colors for web elements (b) To specify the order of elements on a web page (c) To dictate how elements are structured and sized (d) To manage user interactions with elements
2. Which component of the box model holds the actual content of an element, such as text or images? (a) Padding (b) Border (c) Content (d) Margin
3. What property is used to control the size of the content box in CSS? (a) width (b) padding (c) border (d) margin
4. Which CSS property is used to set the minimum width an element can be? (a) min-width (b) max-width (c) width (d) margin
5. What CSS property determines how an element is positioned within its containing element or viewport? (a) width (b) height (c) position (d) margin
6. What is the purpose of the float property in CSS? (a) To increase the font size of an element (b) To make an element disappear (c) To remove an element from the normal document flow (d) To control the visibility of an element
7. Which value for the visibility property in CSS makes an element hidden but still occupies space on the page? (a) visible (b) hidden (c) collapse (d) block
8. The display property in CSS can have a value of "inline-block." What does this value do? (a) Renders the element as a block-level element (b) Combines aspects of both block and inline elements (c) Completely hides the element (d) Controls the overflow of content
9. What does the "overflow: scroll" value do in the CSS overflow property? (a) Displays no scrollbars (b) Clips overflowing content (c) Always displays scrollbars (d) Displays scrollbars only when content overflows
10. What does "float: left" do in CSS? (a) Floats the element to the left, making content flow to its right (b) Floats the element to the right (c) Makes the element disappear (d) Sets a minimum width for the element

B. Fill in the blanks

1. The _____ box model consists of four primary components: content, padding, border, and margin.
2. Padding creates _____ between the content and the border of an element.
3. The _____ of the content area are determined by the width and height properties.
4. Min-width and _____ properties define constraints on the size of elements.
5. The CSS property _____ is used to specify the position of an element.
6. The _____ property is often used for creating layouts with multiple columns.
7. The visibility property can take values like visible, _____, and collapse.
8. The CSS property "display: inline-block" combines aspects of both block and _____ elements.
9. The "overflow" property controls the content of an element _____ its allocated space.
10. "float: left" makes an element float to the left, and content flows to its _____ side.

C. True or False

1. The content area of an element is determined by the padding property.
2. The "min-width" property ensures that an element will never be smaller in width than the specified value.
3. The "float" property is used to remove an element from the normal document flow.
4. "visibility: hidden" makes an element invisible and does not occupy space on the page.
5. The "display: inline-block" value renders an element as a block-level element.
6. The "overflow: scroll" value displays no scrollbars, even when content overflows.
7. "float: left" makes an element float to the right, causing content to flow to its left side.
8. The "clear" property is used to make an element transparent.
9. The default margin value is 0.
10. The "border" property controls the size of an element's content box.

D. Short Question Answers

1. What are the four primary components of the CSS box model?
2. How is the size of the content area of an element determined in CSS?
3. Explain the purpose of the "padding" property in CSS.
4. Which CSS property is used to set constraints on the minimum width and height of an element?
5. How does "position: relative" work in CSS?
6. How is the "float" property in CSS used, and what are its common values?
7. What does the "visibility: hidden" value do in the CSS "visibility" property?
8. Describe the "display: inline-block" value in the CSS "display" property.
9. What does the "overflow: scroll" value do in the CSS "overflow" property?
10. What is the default value for the margin property in CSS?

Module 3**WEB DEVELOPMENT USING
JAVA SCRIPT****Module Overview**

In this Module, you will first learn about " JavaScript " covering Overview of JavaScript, Features of JavaScript, Difference between JavaScript and Java, print the message "Hello, world" and Browser Object Model. Then, we move to "Conditional logic and Flow control ", where you will learn about Conditional logic in expression, Flow control in JavaScript and Loops. Then, we discuss "Arrays and Functions ", covering Arrays, Array Methods, Arrays and Loops, Functions, Function Return, Functions and Differences between Function Parameters and Arguments.

Then, we discuss " String Manipulation" covering will understand about String Manipulation, Dates and Time, and Forms. Furthermore, explaining about "Manipulate images and Geolocation in Javascript", explaining Manipulate image, Creating New Images with JavaScript, Image Rollover Effect, and Timers and Images. Lastly, we discuss "HTML5 Canvas", covering HTML5 Canvas tag in JavaScript, Drawing Shapes, Gradients, and using Images with the Canvas Tag. This Module equips you with essential knowledge and skills in JavaScript.

Learning Outcomes**Module Structure**

Session 1. JavaScript

Session 2. Conditional Logic and Flow Control

Session 3. Arrays and Functions

Session 4. String Manipulation

Session 5. Manipulate Images using Javascript

Session 6. HTML5 Canvas

Session 1. JavaScript

Ankit, a curious kid, discovered "JavaScript," which was like a magic wand for web pages, making them do exciting things like games and interactive quizzes. With JavaScript, he could animate characters, make web pages respond to clicks, count scores, and change things with a single click. Ankit used JavaScript to create web games and interactive stories, feeling like a digital magician. It made the internet an entertaining place where he could have fun with his friends and share cool things he created as shown in Figure 1.1.



Fig. 1.1: Ankit using javascript

In this chapter, you will understand about Overview of JavaScript, Features of JavaScript, Difference between JavaScript and Java, print the message "Hello, world" and Browser Object Model.

1.1 Overview of JavaScript

JavaScript is a highly versatile and dynamic programming language that plays an important role in web development. Its primary function is to enable developers to enhance web pages with interactivity, modify content on the web, and craft responsive user interfaces within web applications.

Did You Know?

In 1995, Brendan Eich created JavaScript at Netscape Communications Corporation. Initially named "LiveScript".

Client-Side Language

JavaScript serves as a predominant client-side scripting language, functioning within the user's web browser. This dynamic language offers the capability to be directly inserted within HTML documents or externally linked as separate files. It empowers developers to imbue web applications with interactivity, dynamically manipulate webpage content, and craft responsive user interfaces.

Dynamic and Weakly Typed

JavaScript is dynamically typed, eliminating the necessity for explicit data type declarations when working with variables. It's also weakly typed, which allows for flexible variable type conversions.

Interactivity: JavaScript is the backbone of interactivity on the web. It enables features like form validation, animations, and responding to user interactions, making websites more engaging and user-friendly.

Event-Driven: JavaScript is event-driven, meaning it can respond to various events such as user clicks, keyboard inputs, and page loads. Event listeners are used to handle these events.

DOM Manipulation

The Document Object Model (DOM) serves as an interface for programming web documents. JavaScript has the capability to modify the DOM, enabling dynamic alterations to the content and layout of web pages. This functionality is a fundamental component in the development of interactive and adaptable web applications.

Variables and Data Types

JavaScript provides support for a range of data types, encompassing numbers, strings, booleans, objects, arrays, and functions. Variables can be defined using `var`, `let`, or `const`.

Functions: Functions are a fundamental part of JavaScript. You can define functions to encapsulate code and reuse it throughout your program. Anonymous functions (also known as lambda functions) are commonly used.

Control Flow: JavaScript offers a diverse set of control flow structures, including `if...else` statements, `switch` statements, `for` loops, `while` loops, and other constructs. These control flow mechanisms empower developers to manage the flow and execution of their programs, enabling them to make decisions, loop through tasks, and implement various logical operations as needed in their code.

Asynchronous Programming: Asynchronous programming in JavaScript is a way of writing code that allows you to do other things while you wait for a response from a server or other resource. This can be useful for things like making web pages more responsive, or for performing tasks that would take a long time to complete if they were done synchronously.

There are a few different ways to implement asynchronous programming in JavaScript, but the most common is to use callbacks.

Did You Know?

A callback is a function that is called when an asynchronous operation completes.

Libraries

In JavaScript, libraries are collections of pre-written code or functions that can be easily reused in your own JavaScript programs. These libraries provide a set of tools and capabilities to simplify common tasks, such as working with the Document Object Model (DOM), handling animations, making HTTP requests, and more. Popular JavaScript libraries include jQuery, React, Angular, and many others, each designed for different purposes and use cases.

Did You Know?

Libraries is a collection of classes and methods stored for re-usability.

Frameworks

JavaScript frameworks are pre-established tools and guidelines that streamline web development. They provide structure, help manage complex tasks, and promote code reusability. Examples include React, Angular, and Vue.js. They simplify development and offer optimization, compatibility, and community support. Choose a framework based on your project's needs and your team's expertise.

Did You Know?

Framework tries to provide everything required to develop a complete application.

Server-Side Usage

Server-side usage of JavaScript refers to running JavaScript code on the server instead of the web browser. Node.js is a popular platform for server-side development. It's used for creating APIs, interacting with databases, handling file operations, implementing authentication, and more. Server-side JavaScript is versatile and essential for building full-stack applications.

Cross-Browser Compatibility

JavaScript libraries and frameworks often help ensure that code works consistently across different web browsers, addressing browser-specific issues and inconsistencies.

1.1 Features of JavaScript

- Most web browsers can run JavaScript because they have built-in tools.
- JavaScript uses syntax and structure that are similar to the C programming language.
- JavaScript is an object-oriented programming language that revolves around objects, using prototypes instead of classes for inheritance.
- JavaScript is a lightweight language that's read and processed on the go.
- It is a light-weighted and case-sensitive language.
- JavaScript works on various operating systems like Windows, macOS, and more.
- It offers users significant control over web browsers.

Did You Know?

JavaScript is a lightweight, object-oriented programming language used by various websites to script their web pages.

1.2 Difference between JavaScript and Java

Table 1.1: Difference between JavaScript and Java

JavaScript	Java
A scripting language used for web development.	A high-level, general-purpose programming language.
Lightweight and loosely typed.	Strictly typed with a more complex syntax.
Primarily runs in web browsers.	Runs on the Java Virtual Machine (JVM) and is used for various applications.
Prototype-based object-oriented.	Class-based object-oriented.
Interpreted at runtime.	Compiled to bytecode before execution.

Assignment 1.1.

- List down the features of Javascript.
- List down the differences between Javascript and Java.

1.3 Create a Web Page from JavaScript Template**1.3.1 HTML Structure**

- The `<!DOCTYPE html>` declaration specifies the document type and version of HTML being used (HTML5).
- At the core of the HTML document lies the `<html>` element, serving as the foundational element that defines the document's language (in this instance, English).
- Within the `<head>` section, you'll discover crucial metadata pertaining to the document, encompassing details like character encoding and viewport configurations.
- Nestled within the `<head>` section, the `<style>` segment plays a pivotal role by housing inline CSS styles, which are employed to artistically design the various elements found on the web page.

1.3.2 CSS Styles

- The CSS styles defined in the `<style>` section apply to various elements on the page.
- The `body` style removes default margins and padding, sets the font to Arial, and specifies a background image (`background.jpg`) with `background-size: cover` to cover the entire viewport.
- The `.container` style sets up a fixed-position navigation bar with a semi-transparent white background and padding.
- The `#icon` style defines a cursor pointer for the play/pause icon, making it look clickable.

1.3.3 Page Content:

Inside the `<body>`, there's a `<div>` with the class `container`. The `<div>` element you see here serves as the top-of-the-page navigation bar. It incorporates an `<h1>` element featuring the text "MY SOUNDS LIST" and an `` element identified by the ID "icon." The image used, labeled "pause.jpg," originally serves as a visual representation of a pause button.

1.3.4 Audio Player

- An `<audio>` element with the ID `mysound` is included. It has two `<source>` elements inside it, providing audio files in different formats for compatibility. The `src` attribute points to an MP3 file named "XYZ.mp3," and there's an alternate source for an Ogg Vorbis audio file (you should replace "audio.ogg" with the correct file path if you have it).
- The "Your browser does not support the audio element" message will be displayed in case the browser doesn't support HTML5 audio.

1.3.5 JavaScript

- The JavaScript code located at the conclusion of the document manages audio playback and the alteration of the play/pause icon.
- It obtains references to the "mysound" audio element and the "icon" image element by utilizing the `document.getElementById` method.
- When the icon is clicked, the `onclick` event handler switches between playing and pausing the audio. If the audio is in a paused state, it initiates playback and updates the icon to "play.jpg." Conversely, if the audio is currently playing, it halts playback and reverts the icon to "pause.jpg."

CODE-

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```

<style>
  body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
    background-image: url('background.jpg'); /* Replace 'background.jpg' with your image file's
name */
    background-size: cover;
    background-position: center;
  }
  .container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: rgba(255, 255, 255, 0.7);
    padding: 20px;
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
  }

  #icon {
    cursor: pointer;
  }
</style>
</head>
<body>
<div class="container">
<h1>MY SOUNDS LIST</h1>

</div>

<audio id="mysound">
<source src="XYZ.mp3" type="audio/mpeg">
<!-- Provide alternate sources for compatibility -->
<source src="audio.ogg" type="audio/ogg">
  Your browser does not support the audio element.
</audio>

```

```

<script>
    const mysound = document.getElementById("mysound");
    const icon = document.getElementById("icon");

    icon.onclick = function () {
        if (mysound.paused) {
            mysound.play();
            icon.src = "play.jpg";
        } else {
            mysound.pause();
            icon.src = "pause.jpg";
        }
    };
</script>
</body>
</html>

```

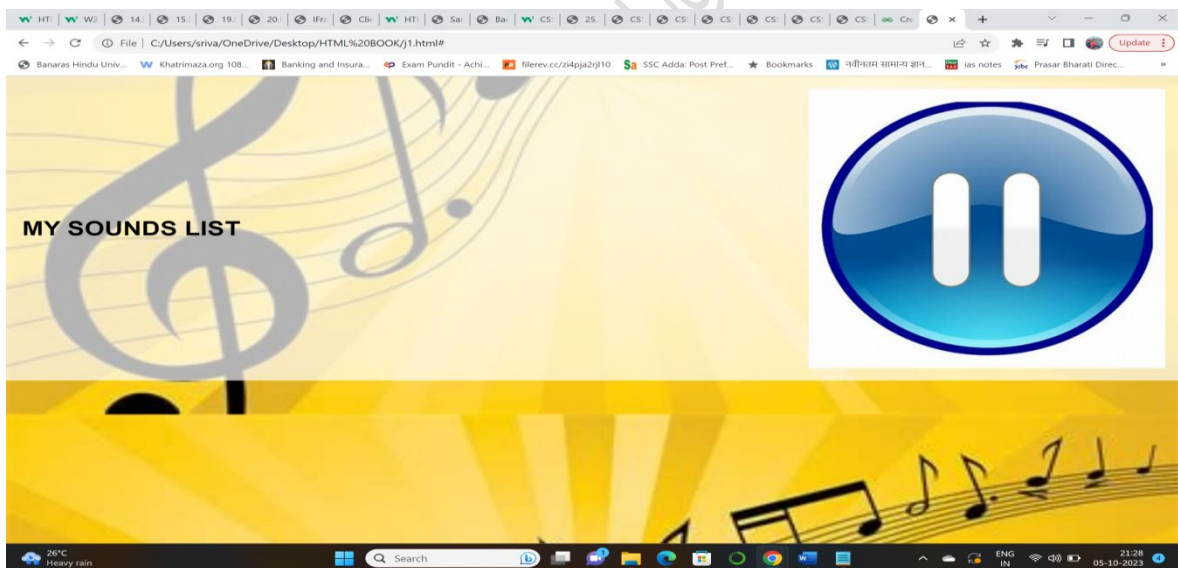
OUTPUT-

Fig. 1.2: Output for JavaScript code "mysound" audio element

1.4 Print the message "Hello, world".

We can create our initial client-side JavaScript script to display the message "Hello, world." To begin, create a new file and input the following code. Please omit the line numbers, which are included for explanatory purposes. Keep in mind the following:

JavaScript is sensitive to letter cases. "A rose" is distinct from "A ROSE" and "A Rose."

"Additional" white spaces, such as spaces, tabs, and line breaks, are disregarded. This means that multiple white spaces are treated as a single blank character. You can use them as needed to enhance the readability of your code as shown in Figure 1.3.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Example: Functions alert() and document.write()</title>
  <script>
    alert("Hello World!")
  </script>
</head>
<body>
  <h1>My First JavaScript Says:</h1>
  <script>
    document.write("<h2><em>Hello World!, again</em></h2>")
    document.write("<p>This document was last modified on " + document.lastModified + "</p>")
  </script>
</body>
</html>

```

Fig. 1.3: JavaScript script to display the message "Hello, world."

Here's a summary and clarification of the key points

1.4.1 Embedding JavaScript in HTML:

Typically, JavaScript code is inserted into an HTML document by utilizing `<script>` tags. The JavaScript code is positioned between the opening `<script>` tag and the closing `</script>` tag.

In HTML4/XHTML, it is permissible to include the `type="text/javascript"` attribute in the opening `<script>` tag, even though it is not mandatory in contemporary HTML5.

1.4.2 Placement of JavaScript:

JavaScript code can be situated in either the `<head>` section, often referred to as "header script," or the `<body>` section, often referred to as "body script," of an HTML document.

To include multiple JavaScript scripts within a single HTML document, you can employ multiple `<script>` elements.

1.4.3 Termination of Statements:

JavaScript statements are terminated by a semicolon `;`, similar to programming languages like Java, C, C++, and C#.

1.4.4 alert() Function:

The `alert(str)` function is used to display a pop-up dialog box with the specified `str` message and an "OK" button.

Strings in JavaScript can be enclosed in either double quotes ("hello") or single quotes ('hello').

1.4.5 The document Object:

JavaScript uses the document object to represent the current web page.

The last modified date of the current document is stored in the `document.lastModified` property.

To dynamically include a specified string (`str`) as part of the current HTML document, the `document.write(str)` function can be employed.

1.4.6 String Concatenation:

The `+` operator in JavaScript can be used to concatenate (join) two strings together, similar to languages like Java.

1.4.7 Commonly-Used Functions:

The `alert()` and `document.write()` functions are common built-in functions provided by JavaScript. `alert()` is used for displaying pop-up messages, while `document.write()` is used to add content to the HTML document.

OUTPUT-

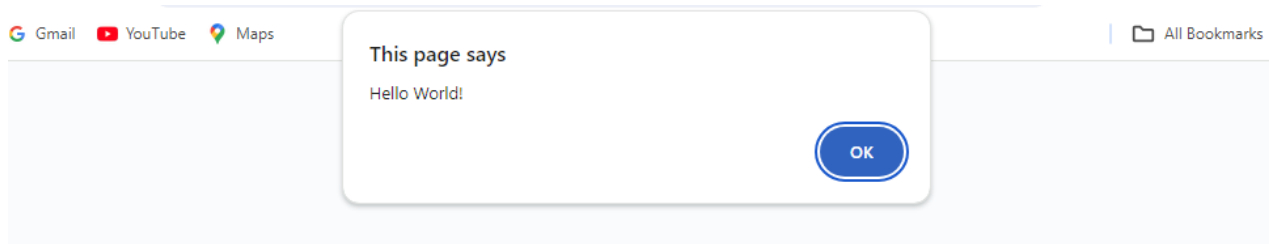


Fig. 1.4: Output for JavaScript script to display the message "Hello, world."

My First JavaScript Says:

Hello World!, again

This document was last modified on 03/12/2024 11:55:42

Fig. 1.5: Output for JavaScript script to display the message "Hello, world again."

Assignment 1.2.

- Write a JavaScript program to print "Hello World".
- Write a JavaScript program to print "Hello World" using `alert()`.

1.5 Debugging JavaScript

Debugging JavaScript code in a web page involves identifying and fixing issues in your code to produce the desired output. Here's a step-by-step demonstration of how to debug and produce the output of a web page created from a JavaScript template:

Suppose we have a simple web page with an HTML structure and JavaScript to display a message when a button is clicked. However, there is a bug in the code, and we need to debug it.

Step 1: Create the HTML Structure code as shown in Figure 1.6.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Debugging Example</title>
</head>
<body>
  <h1>Debugging JavaScript</h1>
  <button id="showMessageButton">Show Message</button>
  <div id="messageDisplay"></div>

  <script src="script.js"></script>
</body>
</html>
```

Fig. 1.6: Create the HTML Structure code

In this HTML code, we have a button with the ID show Message Button and an empty div with the ID message Display. The JavaScript code is included in a separate file called script.js, which we will create in the next step.

Step 2: Create the JavaScript File

Create a file named script.js and include the following JavaScript code:

```
// Attempt to display a message when the button is clicked
document.getElementById("showMessageButton").addEventListener("click", function() {
document.getElementById("messageDisplay").textContent = "Hello, Debugging!";
});
```

Step 3: Debugging Process

Now, let's say there's an issue with the JavaScript code, and when you click the button, nothing happens. To debug this issue, follow these steps:

i. Check the Browser Console:

- Open your web page in a web browser (e.g., Google Chrome, Firefox, or Microsoft Edge).
- Right-click on the page and select "Inspect" or press F12 to open the developer tools as shown in Figure 1.7.
- Go to the "Console" tab. This is where JavaScript errors and messages are displayed as shown in Figure 1.8.

Debugging JavaScript

Show Message

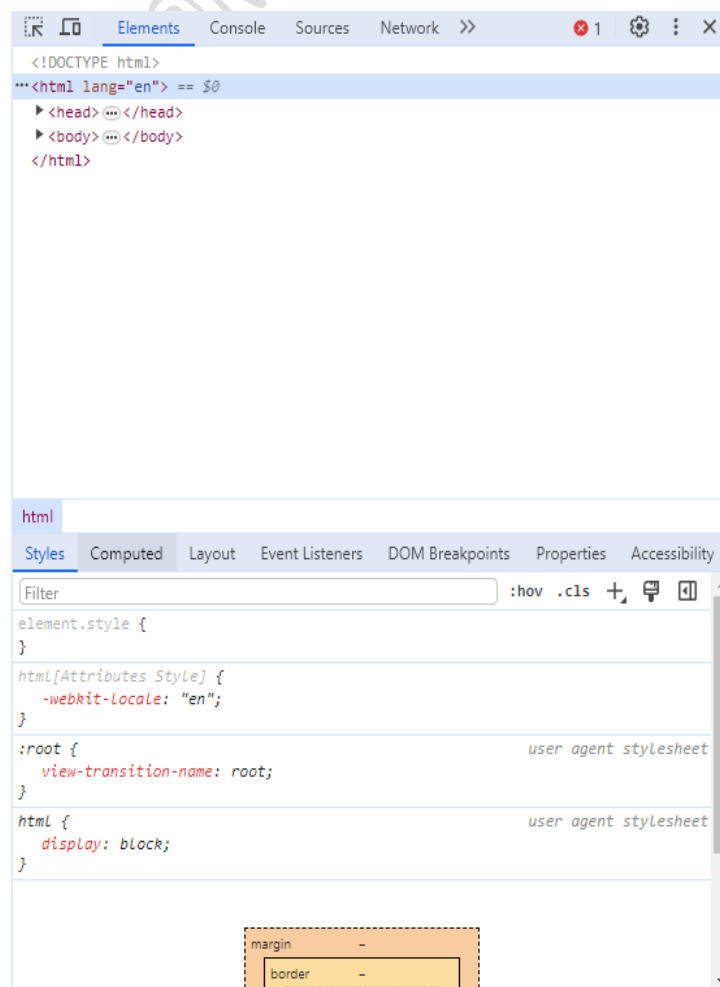


Fig. 1.7: Debugging Process shown in Elements

Debugging JavaScript

Show Message

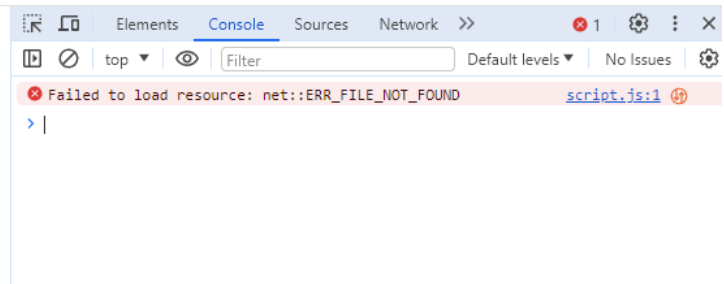


Fig. 1.8: Debugging Process shown in Console

ii. Look for Errors:

- Check if there are any error messages in the console. In this case, there might be an error indicating what's wrong with the JavaScript code.

iii. Fix the Issue:

- In this example, the issue is that we have a typo in the script.js file. We used `getElementById` to add an event listener to the button, but we mistakenly capitalized the "B" in `getElementById`. It should be `getElementById` with a lowercase "b."

// Corrected code

```
document.getElementById("showMessageButton").addEventListener("click", function() {
  document.getElementById("messageDisplay").textContent = "Hello, Debugging!";
});
```

iv. Test Again:

- Save the corrected script.js file.
- Refresh the web page.
- Click the "Show Message" button.

Step 4: Producing the Output

After debugging and fixing the issue, the web page should now work as expected. When you click the "Show Message" button, the message "Hello, Debugging!" should be displayed in the message Display div as shown in Figure 1.9.

Debugging JavaScript

Show Message
Hello, Debugging!

Fig. 1.9: Debugging Process shown in "Hello, Debugging!"

This demonstration showcases the process of debugging JavaScript code within a web page to produce the desired output. Debugging involves identifying issues, checking the browser console for errors, making necessary corrections, and testing the code again to ensure it functions correctly.

1.6 Browser Object Model

The Browser Object Model (BOM) is a set of objects provided by web browsers to interact with the browser itself, the window, and other browser-specific features. It is a separate component from the Document Object Model (DOM), which deals with the structure and content of web documents (HTML, XML, etc.). BOM allows JavaScript to communicate with the browser to perform tasks beyond manipulating web page content.

Here are some of the key components and objects.

1.6.1 Window Object:

- Within the Browser Object Model (BOM), the all-encompassing object is the "window" object, symbolizing the browser window or tab. It acts as the primary interface for engaging with the browser.
- JavaScript allows you to retrieve and modify attributes and functions associated with the window object.
- The window object in JavaScript provides a variety of methods that allow you to interact with and manipulate the browser window or tab. These methods can be used to perform tasks like opening new windows, displaying alerts and prompts, controlling the document's behavior, and more. Here are some commonly used window methods:

// Accessing the window object

window.location.href = "https://www.example.com"; // Changing the URL

window.alert("Hello, BOM!"); // Displaying an alert

alert(message):

Displays a pop-up dialog box with the specified message and an "OK" button.

```
window.alert("This is an alert!");
```

confirm(message):

Displays a pop-up dialog box with the specified message, along with "OK" and "Cancel" buttons. Returns true if "OK" is clicked and false if "Cancel" is clicked.

```
var result = window.confirm("Do you want to continue?");
```

```
if (result) {
```

```
    // User clicked OK
```

```
} else {
```

```
    // User clicked Cancel
```

```
}
```

prompt(message, defaultText):

Displays a pop-up dialog box with the specified message, an input field for text entry, and "OK" and "Cancel" buttons. Returns the entered text as a string if "OK" is clicked, or null if "Cancel" is clicked.

```
var name = window.prompt("Enter your name:", "John Doe");
```

```
if (name !== null) {
```

```
    // User entered a name
```

```
} else {
```

```
    // User clicked Cancel
```

```
}
```

open(url, target, features): Opens a new browser window or tab with the specified url. You can specify the target (e.g., "_blank" for a new tab) and additional features (e.g., window size and properties).

The window.open method is used to open a new browser window or tab. It takes three parameters:

url (string): The URL of the page you want to open in the new window.

target (string): Specifies where to open the new window. Common values include:

"blank": Opens in a new tab or window (default).

"self": Opens in the current tab or window.

"parent": Opens in the parent window or frame.

"top": Opens in the top-level browsing context.

A custom window name: Opens in a window with the specified name (useful for targeting specific windows).

features (string): A list of window features, such as size, position, and behavior. This is a comma-separated string.

```
window.open("https://www.example.com", "_blank", "width=500,height=300");
```

close(): Closes the current browser window or tab if it was opened using JavaScript. Most modern browsers do not allow scripts to close windows that were not opened by scripts for security reasons.

```
window.close();
```

setTimeout(function, delay): Schedules the execution of the specified function after a specified delay in milliseconds. Returns a timer ID that can be used to cancel the timeout using `clearTimeout()`.

```
var timerId = setTimeout(function() {
  console.log("Delayed function executed!");
}, 2000); // Execute after 2 seconds
```

clearTimeout(timerId): Cancels a timeout set using `setTimeout()`. Pass the timer ID returned by `setTimeout()`.

```
clearTimeout(timerId);
```

setInterval(function, delay): Repeatedly executes the specified function at intervals specified by delay in milliseconds. Returns an interval ID that can be used to cancel the interval using `clearInterval()`.

```
var intervalId = setInterval(function() {
  console.log("Interval function executed!");
}, 1000); // Execute every 1 second
```

clearInterval(intervalId): Cancels an interval set using `setInterval()`. Pass the interval ID returned by `setInterval()`.

```
clearInterval(intervalId);
```

These are some of the commonly used methods provided by the window object in JavaScript for controlling the behavior of browser windows and interacting with users through pop-up dialogs.

Navigator Object: The navigator object provides information about the user's browser and system.

```
// Accessing navigator properties
var browserName = navigator.userAgent;
var isMobile = navigator.userAgent.match(/Mobile/);
```

Screen Object: The screen object provides information about the user's screen, such as width, height, and color depth.

```
// Accessing screen properties
```

```
var screenWidth = screen.width;
var screenHeight = screen.height;
```

History Object: The history object allows you to navigate the browser's history, such as moving forward and backward in the user's browsing session.

```
// Accessing history methods
window.history.back(); // Go back one page in history
window.history.forward(); // Go forward one page in history
```

Location Object: The location object provides information about the current URL and allows you to navigate to different URLs.

```
// Accessing location properties
var currentURL = location.href;
// Changing the URL
location.href = "https://www.example.com";
```

Cookies: Browsers provide mechanisms to work with cookies, which are small pieces of data stored on the user's computer.

```
// Creating a cookie
document.cookie = "username=John; expires=Thu, 18 Dec 2023 12:00:00 UTC; path=/";
```

Timers: The BOM includes methods for scheduling code execution, such as `setTimeout()` and `setInterval()`.

```
// Using setTimeout
setTimeout(function() {
  alert("Delayed alert!");
}, 2000); // Display the alert after a 2-second delay
```

The Browser Object Model provides JavaScript with the ability to interact with the browser's environment, manage the browser's history, handle cookies, access user agent information, and more. It plays a crucial role in building dynamic and interactive web applications.

1.6.2 Windows events

Window events in JavaScript allow you to respond to various interactions and changes that occur within the browser window. Here are some commonly used window events:

load Event: The load event is fired when a web page or all its external resources (images, stylesheets, scripts) have finished loading.

```
window.addEventListener("load", function() {
  // Your code to run after the page has loaded
});
```

unload Event: The unload event is fired just before the user leaves the page (e.g., by closing the browser window or navigating to a different page).

```
window.addEventListener("unload", function() {
  // Your code to perform cleanup before leaving the page
});
```

resize Event: The resize event is fired when the browser window is resized.

```
window.addEventListener("resize", function() {
```

```
// Your code to handle window resizing  
});
```

scroll Event: The scroll event is fired when the user scrolls the page.

```
window.addEventListener("scroll", function() {  
    // Your code to respond to scrolling  
});
```

beforeunload Event: The beforeunload event is fired just before the user leaves the page. It can be used to display a confirmation dialog.

```
window.addEventListener("beforeunload", function(event) {  
    event.returnValue = "Are you sure you want to leave this page?";  
});
```

These are just a few examples of window events in JavaScript as shown in Figure 1.10. You can use event listeners to respond to these events and perform actions based on user interactions or changes in the browser window. Event handling is a fundamental part of building interactive and dynamic web applications as shown in code.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Window Object Example</title>
  <script>
    function showAlert() {
      window.alert("This is an alert!")
    }
    function showConfirmation() {
      var result = window.confirm("Do you want to continue?")
      if (result) {
        console.log("User clicked OK.")
      }else{
        console.log("User clicked cancel.")
      }
    }
    function showPrompt() {
      var name = window.prompt("Enter your name:", "Vijay Goswami")
      if (name !== null) {
        console.log("User entered: " + name)
      }else{
        console.log("User clicked cancel.")
      }
    }
    function openNewWindow() {
      window.open("https://www.example.com", "_blank", "width=500, height=300")
    }
    function setDelayedFunction() {
      window.setTimeout(function() {
        console.log("Delayed function executed!")
      }, 2000) //Execute after 2 seconds
    }
    var intervalId
    function startInterval() {
      intervalId = window.setInterval(function() {
        console.log("Interval function executed!")
      }, 1000) //Execute every 1 second
    }
    function stopInterval() {
      clearInterval(intervalId)
      console.log("Interval stopped.")
    }
  </script>
</head>
<body>
  <h1>Window Object Example</h1>
  <button onclick="showAlert()">Show Alert</button>
  <button onclick="showConfirmation()">Show Confirmation</button>
  <button onclick="showPrompt()">Show Prompt</button>
  <button onclick="openNewWindow()">Open New Window</button>
  <button onclick="setDelayedFunction()">Set Delayed Function</button>
  <button onclick="startInterval()">Start Interval</button>
  <button onclick="stopInterval()">Stop Interval</button>
</body>
</html>

```

Fig. 1.10: window events in JavaScript

OUTPUT

Window Object Example

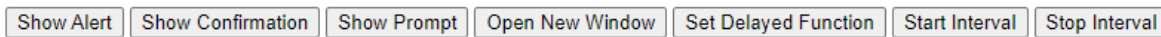


Fig. 1.11: Output for window events in JavaScript

After clicking on Show Alert button as shown in Figure 1.12.

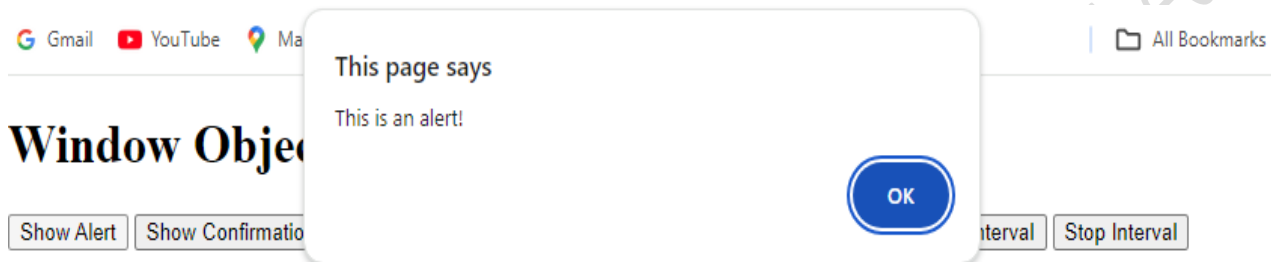


Fig. 1.12: Output for Show Alert button window events in JavaScript

Similarly, it works on each button.

1.7 Document Object Model

The Document Object Model (DOM) serves as a critical programming interface for web documents, providing a structured representation of an HTML or XML document in the form of a tree-like model. This model essentially constructs a tree structure where each node within the tree corresponds to a specific part or element of the document, such as headings, paragraphs, images, and more. The DOM's primary purpose is to facilitate interaction with web pages in a dynamic manner, effectively allowing programs and scripts to manipulate and modify the content, structure, and presentation of a web page.

In practical terms, the DOM empowers developers to use scripting languages like JavaScript to access and modify the various components of a web page, whether it's altering text, changing the appearance of elements, or responding to user interactions. This dynamic capability is at the core of creating interactive and responsive web applications, where content can be updated on-the-fly without requiring the entire page to be reloaded. In essence, the DOM is a foundational component in modern web development, providing the means to transform static web pages into dynamic and user-friendly experiences.

Here are some key concepts and aspects of the DOM:

Tree Structure:

- The DOM represents an HTML or XML document as a tree structure with nodes.
- The top-level node is the document object, which represents the entire web page.
- Nodes in the DOM can represent elements, attributes, text, comments, and more.

Accessing Elements:

- You can access and manipulate elements on a web page by selecting them from the DOM using various methods and properties.
- Common methods include `getElementById`, `getElementsByClassName`, `getElementsByTagName`, and `querySelector`.

- `document.write()` and the various DOM selection methods like `getElementById`, `getElementsByClassName`, `getElementsByTagName`, and `querySelector` are commonly used techniques for accessing and manipulating elements in the Document Object Model (DOM) of a web page. Let's explore each of these methods:

`document.write(content):`

- The `document.write()` method is used to write content directly to the HTML document at the current position, which is typically within the `<body>` element.
- It is often used to add content dynamically to a web page.

```
document.write("<h1>Hello, World!</h1>");
```

Note: Be cautious when using `document.write()`, as it can overwrite the entire document if called after the document has fully loaded. It's generally recommended to use other DOM manipulation methods for better control.

`getElementById(id):` The `getElementById()` method retrieves an element from the DOM by its unique id attribute.

```
var element = document.getElementById("myElement");
```

`getElementsByClassName(className):` The `getElementsByClassName()` method returns a collection (an array-like object) of elements with the specified class name.

```
var elements = document.getElementsByClassName("myClass");
```

`getElementsByTagName(tagName):` The `getElementsByTagName()` method returns a collection of elements with the specified HTML tag name.

```
var paragraphs = document.getElementsByTagName("p");
```

`querySelector(selector):` The `querySelector()` method allows you to select a single element from the DOM by specifying a CSS selector.

It returns the first element that matches the selector.

```
var element = document.querySelector(".myClass");
```

These DOM selection methods are essential for interacting with and manipulating elements on a web page as shown in Figure 1.1. They allow you to access specific elements, modify their content and attributes, and respond to user interactions. When choosing a method, consider the specific use case and the level of flexibility required for your task.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DOM Selection and document.write Example</title>
</head>
<body>
  <h1 id="header">DOM Selection Example</h1>
  <!-- Using document.write() to add content -->
  <script>
    document.write("<p>This is added using document.write()</p>")
  </script>
  <!-- Using getElementById to modify content -->
  <script>
    var header = document.getElementById("header")
    header.innerHTML = "Updated using getElementById"
  </script>
  <!-- Using getElementsByClassName to modify content -->
  <div>
    <p class="myClass">This is selected by getElementsByClassName</p>
  </div>
  <script>
    var elementByClass = document.getElementsByClassName("myClass")[0]
    elementByClass.textContent = "Updated using getElementsByClassName"
  </script>
  <!-- Using getElementsByTagName to modify content -->
  <ul>
    <li>This is selected by getElementsByTagName</li>
  </ul>
  <script>
    var elementsByTagName = document.getElementsByTagName("li")[0]
    elementsByTagName.textContent = "Updated using getElementsByTagName"
  </script>
  <!-- Using querySelector to modify content -->
  <div>
    <p>This is selected by querySelector</p>
  </div>
  <script>
    var elementByQuery = document.querySelector(".myClass")
    elementByQuery.textContent = "Updated using querySelector"
  </script>
</body>
</html>

```

Fig. 1.13: Document Object Model (DOM) methods

Output**Updated using getElementById**

This is added using document.write()

Updated using querySelector

- Updated using getElementsByTagName

This is selected by querySelector

Fig. 1.14: Output for Document Object Model (DOM) methods

Explanation

- `<!DOCTYPE html>`: Declares the document type and version as HTML5.
- `<html lang="en">`: Specifies the document's root element, and `lang="en"` indicates the language is English.
- `<head>`: Contains metadata about the document, including the character set and title.
- `<meta charset="UTF-8">`: Sets the character encoding to UTF-8 for proper text display.
- `<title>DOM Selection and document.write Example</title>`: Sets the title of the web page, displayed in the browser's title bar.
- `<body>`: Contains the visible content of the web page.
- `<h1 id="header">DOM Selection Example</h1>`: Defines an `<h1>` (heading) element with an `id` attribute of "header" and sets its text content.
- `<!-- Using document.write() to add content -->`: A comment that describes the purpose of the following script.
- `<script>`: Indicates the start of a JavaScript script.
- `document.write("<p>This is added using document.write()</p>");`: The `document.write()` method is used to dynamically add HTML content to the page. In this case, it adds a `<p>` element with the specified text content.
- `<!-- Using getElementById to modify content -->`: Another comment describing the purpose.
- `var header = document.getElementById("header");`: This line retrieves the element with the `id` attribute "header" using `getElementById` and stores it in the `header` variable.
- `header.innerHTML = "Updated using getElementById";`: It updates the content of the header element using the `innerHTML` property.
- `<!-- Using getElementsByTagName to modify content -->`: A comment.
- `<div>` and `<p class="myClass">`: HTML elements with a class of "myClass."
- `var elementByClass = document.getElementsByClassName("myClass")[0];`: It retrieves all elements with the class "myClass" and stores the first one in the `elementByClass` variable.
- `elementByClass.textContent = "Updated using getElementsByTagName";`: It updates the text content of the selected element.
- `<!-- Using getElementsByTagName to modify content -->`: A comment.
- `` and ``: Unordered list and list item elements.
- `var elementsByTagName = document.getElementsByTagName("li")[0];`: It retrieves all `` elements and stores the first one in the `elementsByTagName` variable.

- `elementsByTagName.textContent = "Updated using getElementsByTagName";` It updates the text content of the selected `` element.
- `<!-- Using querySelector to modify content -->`: A comment.
- `var elementByQuery = document.querySelector(".myClass");` It selects the first element with the class "myClass" using `querySelector` and stores it in the `elementByQuery` variable.
- `elementByQuery.textContent = "Updated using querySelector";` It updates the text content of the selected element.

These examples demonstrate how to use `document.write()` and various DOM selection methods to access and manipulate elements within an HTML document using JavaScript.

1.8 Variables in Java scripts

In JavaScript, variables play a crucial role in storing and handling data. This versatile language is dynamically typed, meaning that when you declare a variable, you don't have to explicitly specify its data type. Instead, JavaScript automatically determines the data type based on the value you assign to it. Here's a fundamental guide on declaring and utilizing variables in JavaScript

1.8.1 Declaration and Assignment

You can declare a variable using the `var`, `let`, or `const` keyword:

var (used in older versions of JavaScript):

```
var myVariable;
```

let (introduced in ECMAScript 6, widely used for variables that may change):

```
let myVariable;
```

const (introduced in ECMAScript 6, used for constants that shouldn't be reassigned):

```
const myConstant = 10;
```

After declaring a variable, you can assign a value to it:

```
let myVariable = "Hello, world!";
```

1.8.2 Variable Naming Rules

In JavaScript, when it comes to variable names:

- Case matters; `myVariable` and `myvariable` are distinct variables.
- Variable names must commence with a letter, underscore (`_`), or dollar sign (`$`).
- Following characters can include digits (0-9).
- Variable names should not incorporate spaces or special characters (apart from underscore and dollar sign).
- Certain words, such as `let`, `const`, `if`, `for`, and more, are reserved and cannot be employed as variable names.

1.8.3 Data Types

JavaScript has several basic data types as shown in Figure 1.14:

Primitive Data Types: These are immutable and include:

- Number: Floating-point numbers (e.g., 3.14, -42).
- String: Textual data (e.g., "Hello").
- Boolean: `true` or `false`.
- Undefined: Represents uninitialized variables.

- Null: Represents the intentional absence of any object value.

Complex Data Types:

- Object: A collection of key-value pairs (e.g., { name: "Vijay", age: 38 }).
- Array: An ordered list of values (e.g., [1, 2, 3]).

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Variables Example</title>
</head>
<body>
<h1>JavaScript Variables Example</h1>

<script>
  // Declaring and initializing variables
  let name = "Vijay";
  const age = 38;
  var isActive = true;

  // Using variables in calculations
  let birthYear = 2022 - age;
  console.log(name + " is " + age + " years old.");

  // Changing variable values
  isActive = false;
  // Concatenating strings using variables
  let greeting = "Hello, " + name + "!";
  console.log(greeting);
  // Using variables in conditions
  if (isActive) {
    console.log(name + " is active.");
  } else {
    console.log(name + " is not active.");
  }
  // Arrays and objects using variables
  let colors = ["red", "green", "blue"];
  let person = {
    firstName: "Vijay",
    lastName: "Goswami"
  };
  console.log(colors[0]); // Outputs "red"
  console.log(person.firstName); // Outputs "Vijay"
</script>
</body>
</html>

```

Fig. 1.14: JavaScript has several basic data types

Follow these steps to run this code:

1. Open a text editor (e.g., Notepad on Windows or TextEdit on macOS).
2. Copy and paste the above HTML code into the text editor.
3. Save the file with an .html extension, such as index.html.

4. Open the saved HTML file using a web browser by double-clicking the file or right-clicking and choosing "Open with" to select a web browser.
5. Once the HTML file is opened in the browser, open the developer console to view the JavaScript output. You can usually do this by right-clicking anywhere on the page, selecting "Inspect" or "Inspect Element," and navigating to the "Console" tab.

You should see the output of the JavaScript code within the HTML file, including variable assignments, calculations, and conditional statements as shown in Figure 1.15.

Output

JavaScript Variables Example

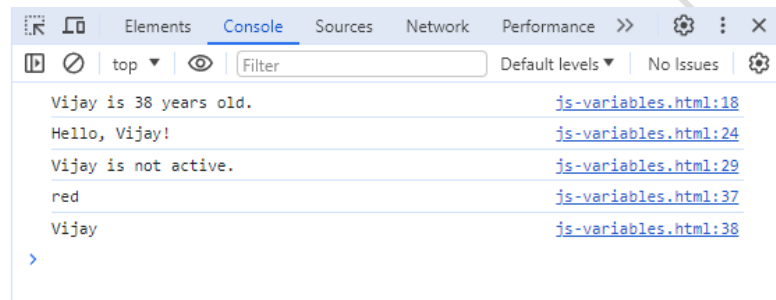


Fig. 1.15: Output for JavaScript Variables

1.9 Mathematical operator used on variables

In JavaScript, you can use various mathematical operators on variables to perform calculations and manipulate numerical data. Here are some of the common mathematical operators and how they can be used with variables:

Addition +: Used to add two or more numbers together.

Example:

```
let num1 = 5;
let num2 = 10;
let sum = num1 + num2;
```

Subtraction -: Used to subtract one number from another.

Example:

```
let num1 = 15;
let num2 = 7;
let difference = num1 - num2;
```

Multiplication *: Used to multiply two or more numbers.

Example:

```
let num1 = 6;
let num2 = 8;
let product = num1 * num2;
```

Division /: Used to divide one number by another.

Example:

```
let num1 = 20;
let num2 = 4;
let quotient = num1 / num2;
```

Modulus %: Returns the remainder when one number is divided by another.

Example:

```
let num1 = 21;
let num2 = 4;
let remainder = num1 % num2;
```

Increment ++ and Decrement --:Used to increase or decrease the value of a variable by 1.

Example:

```
let count = 5;
count++; // Increment by 1
let result = count--; // Decrement by 1 (post-decrement)
```

Assignment Operators (+=, -=, *=, /=):These operators perform an operation and assign the result to a variable in a single step.

Example:

```
let num = 10;
num += 5; // Equivalent to num = num + 5
```

Assignment 1.3

- Write down the purpose and syntax of the following mathematical operator:
 - Modulus %
 - Subtraction -
 - Increment ++ and Decrement –
 - Assignment Operators (+=, -=, *=, /=)
 - Division /

1.10 Operator precedence:

JavaScript determines the order in which operators are evaluated when an expression contains multiple operators. Understanding operator precedence is crucial for writing accurate and predictable JavaScript code. Here's a general overview of operator precedence in JavaScript as shown in Figure 1.16:

- Grouping Parentheses (): The highest precedence goes to expressions enclosed in parentheses. Anything inside parentheses is evaluated first.
- Member Access: and Computed Member Access []: These operators allow you to access properties and elements of objects and arrays, respectively.
- Function Invocation (): When you call a function, its arguments are evaluated before the function is executed.
- New Operator new: Used to create instances of user-defined or built-in objects.
- Increment/Decrement ++ and -- (Postfix): The increment (++) and decrement (--) operators can be used after a variable (postfix), and they modify the variable's value after the current expression is evaluated.
- Logical NOT !: Negates a Boolean value.
- Unary Plus + and Unary Negation -: Used to convert values to numbers with optional sign change.
- Exponentiation **: Calculates the power of a number.
- Multiplication *, Division /, and Remainder %: These operators perform multiplication, division, and calculate the remainder of division, respectively.
- Addition + and Subtraction -: These operators perform addition and subtraction.

- Concatenation + (for strings): The + operator can also be used to concatenate strings.
- Relational Operators (<, >, <=, >=, instanceof, in): These operators compare values and return Boolean results.
- Equality Operators (==, !=, ===, !==): Used to compare values for equality or inequality.
- Logical AND &&: Returns the second operand if the first operand is truthy; otherwise, returns the first operand.
- Logical OR ||: Returns the first operand if it's truthy; otherwise, returns the second operand.
- Conditional (Ternary) Operator ? :: A shorthand way of writing if...else statements.
- Assignment Operators (=, +=, -=, *=, /=, %=, **=): These operators assign values to variables and can also perform additional operations in a single step.
- Comma “,”: The comma operator allows you to evaluate multiple expressions sequentially and returns the value of the last expression.

The operator precedence order listed above gives you a general idea of how JavaScript evaluates expressions. When expressions contain operators with different precedence levels, JavaScript follows these rules to determine the order of evaluation.

It's essential to be aware of operator precedence to avoid unexpected behavior in your code. If needed, you can use parentheses to explicitly specify the order of evaluation within an expression as shown in code.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mathematical Operators Example</title>
</head>
<body>
  <h1>Mathematical Operators Example</h1>

  <script>
    let a = 10
    let b = 5

    let addition = a + b // 15
    let subtraction = a - b // 5
    let multiplication = a * b // 50
    let division = a / b // 2
    let modulus = a % b // 0
    let incrementPost = a++ // 10 (post increment)
    let decrementPost = a-- // 11 (post decrement)

    let exponentiation = 2 ** 3 // 8 (2 to the power of 3)

    let isGreaterThan = a > b // true
    let isEqualTo = a === b // false
    let logicalAnd = true && false // false
    let logicalOr = true || false // true

    let conditional = a > b ? "a is greater" : "b is greater" // a is greater

    console.log("Addition:", addition)
    console.log("Subtraction:", subtraction)
    console.log("Multiplication:", multiplication)
    console.log("Division:", division)
    console.log("Modulus:", modulus)
    console.log("Increment (Post):", incrementPost)
    console.log("Decrement (Post):", decrementPost)
    console.log("Exponentiation:", exponentiation)
    console.log("Is Greater Than:", isGreaterThan)
    console.log("Is Equal To:", isEqualTo)
    console.log("Logical AND:", logicalAnd)
    console.log("Logical Or:", logicalOr)
    console.log("Conditional:", conditional)
  </script>
</body>
</html>

```

Fig. 1.16: Mathematical Operator precedence in JavaScript

Explanation

In this HTML example as shown in Figure 1.17:

- We have an HTML structure with a heading.
- Inside the `<script>` block, we declare variables `a` and `b` and perform various mathematical operations using operators.
- The results are logged to the browser's developer console.

OUTPUT

Mathematical Operators Example

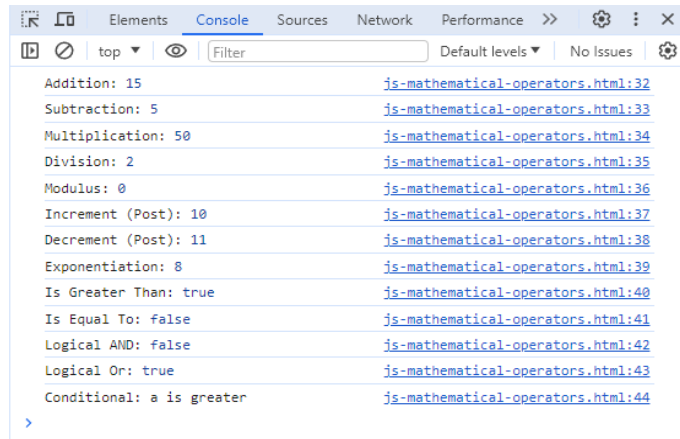


Fig. 1.17: Output for Mathematical Operator precedence in JavaScript

1.11 Storing JavaScript objects in variables

In JavaScript, you can store objects in variables just like any other data type as shown in Figure 1.18. Objects in JavaScript are used to represent collections of key-value pairs, and they can contain various data types, including strings, numbers, other objects, functions, and more. Here's how you can store JavaScript objects in variables.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Objects Example</title>
</head>
<body>
  <h1>Storing Object Example</h1>
  <script>
    // Define an object
    let person = {
      firstName: "Vijay",
      lastName: "Goswami",
      age: 38,
      isStudent: false,
      address: {
        street: "123, Any Locality",
        city: "Any City",
        zipCode: "123456"
      },
      hobbies: ["Reading", "Travelling", "Music"]
    }
    // Store the object in a variable
    let myObject = person

    // Access object properties using the variable
    console.log(myObject.firstName) // Vijay
    console.log(myObject.lastName) // Goswami
    console.log(myObject.address) // 38
    console.log(myObject.isStudent) // false
    console.log(myObject.address.street) // 123, Any Locality
    console.log(myObject.hobbies[0]) // Reading
  </script>
</body>
</html>
```

Fig. 1.18: Storing objects in JavaScript

OUTPUT

```
Storing Object Example  
Vijay js-objects.html:28  
Goswami js-objects.html:29  
▶ {street: '123, Any Locality', city: 'Any City', zipCode: '123456'} js-objects.html:30  
false js-objects.html:31  
123, Any Locality js-objects.html:32  
Reading js-objects.html:33
```

Fig. 1.18: Output for Storing objects in JavaScript

SUMMARY

- JavaScript enhances web pages with interactivity and dynamic content.
- It's primarily a client-side scripting language used in web browsers.
- JavaScript is dynamically typed, allowing flexible variable type conversions.
- It's essential for form validation, animations, and event handling.
- Event-driven, it responds to user interactions with event listeners.
- JavaScript can modify web page content via the Document Object Model (DOM).
- Variables can be defined using var, let, or const.
- Functions are fundamental for code encapsulation and reuse.
- Control flow structures like if...else, for loops, and while loops are available.
- Asynchronous programming is facilitated using callbacks, Promises, and async/await.
- Libraries and frameworks like jQuery and React streamline web development.
- Node.js enables server-side JavaScript development.
- JavaScript libraries ensure cross-browser compatibility.
- JavaScript is case-sensitive, distinguishing between upper and lower case.
- It can be directly inserted into HTML documents using <script> tags.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What is the primary function of JavaScript in web development? (a) Server-side scripting (b) Styling web pages (c) Adding interactivity to web pages (d) Database management
2. Which type of language is JavaScript? (a) Strongly typed (b) Loosely typed (c) Statically typed (d) Compiled
3. JavaScript is primarily a client-side scripting language. What does this mean? (a) It runs on the server (b) It runs in the web browser (c) It is used for database management (d) It is a statically typed language
4. What is the primary role of event listeners in JavaScript? (a) Validate forms (b) Handle user interactions (c) Define data types (d) Manage server-side operations
5. Which acronym represents the Document Object Model? (a) HTML (b) DOM (c) CSS (d) URL
6. What are the three ways to declare variables in JavaScript? (a) var, const, let (b) int, string, bool (c) variable, let, const (d) variable1, variable2, variable3
7. Which JavaScript feature is used to handle asynchronous programming? (a) Promises (b) Callbacks (c) Async/await (d) All of the above
8. Which JavaScript library/framework is known for enhancing user interfaces? (a) jQuery (b) React (c) Angular (d) All of the above
9. Which runtime environment allows JavaScript to run on servers? (a) Node.js (b) Java (c) Python (d) Ruby
10. Which of the following is not a valid way to declare a variable in JavaScript? (a) let (b) variable (c) const (d) var

B. Fill in the blanks

1. Variables in JavaScript can be defined using var, let, or _____.
2. Functions are commonly used for encapsulating code in _____.
3. _____ allows JavaScript to run on server-side environments.
4. JavaScript is _____ to letter case.
5. The _____ function is used to display a pop-up dialog box in JavaScript.
6. The navigator object provides information about the user's _____ and system.
7. The history object allows you to navigate the browser's _____.
8. JavaScript provides mechanisms to work with _____, which are small pieces of data stored on the user's computer.
9. The _____ object provides information about the current URL and allows you to navigate to different URLs.
10. JavaScript control flow structures empower developers to manage the flow and _____ of their programs.

C. True or False

1. JavaScript libraries and frameworks ensure consistent code execution across different web browsers.
2. JavaScript uses a class-based object-oriented programming model.
3. JavaScript is a case-sensitive language.
4. JavaScript is primarily a server-side scripting language.
5. JavaScript can be directly inserted within HTML documents.
6. JavaScript is used for form validation and animations on web pages.
7. JavaScript is a case-insensitive language.
8. The Document Object Model (DOM) is a set of objects provided by web browsers to interact with the browser itself.
9. The navigator object provides information about the user's screen.
10. The `setTimeout()` function in JavaScript schedules the execution of a function after a specified delay.

D. Short Question Answers

1. What role does JavaScript play in web development?
2. Explain the difference between client-side and server-side scripting.
3. What is the Document Object Model (DOM), and how does it relate to JavaScript?
4. Name three ways to declare variables in JavaScript.
5. What is the purpose of event listeners in JavaScript?
6. How does JavaScript handle asynchronous programming, and why is it important?
7. Can you name at least three popular JavaScript libraries or frameworks for web development?
8. What is the significance of Node.js in JavaScript development?
9. Why is JavaScript considered a case-sensitive language?
10. Explain Operator precedence?

Session 2. Conditional Logic and Flow Control

Ankit, a kid who loved puzzles and games, learned about "Conditional Logic and Flow Control," which was all about making smart choices. It was like being the director of his own adventure, where he could ask questions, make decisions based on answers, and guide the flow of a story or a game, just like in a choose-your-own-adventure book. With these tools, he created fun games and stories where players could make choices, making the digital world more interactive and entertaining, just like in a fun puzzle or a thrilling game. As shown in Figure 2.1.



Fig.2.1: Ankit using Conditional Logic and Flow Control

In this chapter, you will learn about Conditional logic in expression, Flow control in JavaScript and Loops.

2.1 Conditional logic in expression

Conditional logic in JavaScript involves making decisions based on conditions and executing different code blocks depending on whether those conditions are true or false. This is typically achieved using `if`, `else if`, and `else` statements. Here's an explanation of conditional logic and some examples of its use:

1. `if` Statement:

In JavaScript, the "if" statement is employed to run a block of code only if a specified condition holds true. If the condition is false, the code block is not executed.

Syntax:

```
if (condition) {  
    // Code to be executed if the condition is true  
}
```

Example:

```
let age = 20;  
if (age >= 18) {  
    console.log("You are an adult.");  
}
```

2. `else if` Statement:

The "else if" statement provides a way to define and evaluate additional conditions when the preceding "if" condition turns out to be false.

Syntax:

```
if (condition1) {
    // Code to be executed if condition1 is true
} else if (condition2) {
    // Code to be executed if condition2 is true
}
```

Example

```
let temperature = 25;
if (temperature < 0) {
    console.log("It's freezing!");
} else if (temperature >= 0 && temperature < 20) {
    console.log("It's cool.");
} else {
    console.log("It's warm.");
}
```

3. else Statement:

The else statement is used to execute a block of code if the preceding if and else if conditions are false.

Syntax:

```
if (condition) {
    // Code to be executed if the condition is true
} else {
    // Code to be executed if the condition is false
}
```

Example

```
let isRaining = false;
if (isRaining) {
    console.log("Don't forget your umbrella.");
} else {
    console.log("No need for an umbrella today.");
}
```

4. Ternary Operator (? :):

The ternary operator offers a concise method for constructing conditional expressions. It assesses a condition and delivers one of two values depending on whether the condition is true or false.

Syntax:

Let result = condition ? valueIfTrue : valueIfFalse;

Example

```
let age = 20;
let message = age >= 18 ? "You are an adult." : "You are not an adult.";
```

```
console.log(message);
```

Did You Know?

Conditional logic is a powerful tool that can be used to control the flow of your code. By using conditional statements, you can make your code more flexible and responsive to different conditions.

5. Comparison Operators

Comparison operators in JavaScript are used to compare values and return a Boolean result (true or false). Common comparison operators include:

`==` (Equality): Checks if two values are equal.

`!=` (Inequality): Checks if two values are not equal.

`===` (Strict Equality): Checks if two values are equal in both value and data type.

`!==` (Strict Inequality): Checks if two values are not equal in either value or data type.

`<` (Less Than): Determines if the left operand is smaller than the right operand.

`>` (Greater Than): Examines whether the left operand is larger than the right operand.

`<=` (Less Than or Equal): Checks if the left operand is less than or equal to the right operand.

`>=` (Greater Than or Equal): Evaluates if the left operand is greater than or equal to the right operand.

Example

```
let x = 5;
```

```
let y = 10;
```

```
console.log(x === y); // Outputs: false
```

6. Logical Operators

Logical operators in JavaScript are used to perform logical operations on Boolean values. The common logical operators include:

`&&` (Logical AND): Returns true if both operands are true.

`||` (Logical OR): Returns true if at least one operand is true.

`!` (Logical NOT): Negates the value of the operand.

Example

```
let isRaining = true;
```

```
let isCold = false;
```

```
if (isRaining && !isCold) {
```

```
  console.log("Bring an umbrella.");
```

```
} else if (isCold || !isRaining) {
```

```
  console.log("Wear a jacket.");
```

```
}
```

7. Nested if Statements

Nested if statements are used when you want to have multiple levels of conditional checks. One if statement is placed inside another.

Example

```
let x = 10;
```

```
if (x > 5) {  
  if (x < 15) {  
    console.log("x is between 5 and 15.");  
  }  
}
```

8. Switch Statements:

A switch statement is used to perform different actions based on different conditions. It provides a more organized and efficient alternative to multiple if...else if...else statements, as shown in Figure 2.2.

```
let day = "Monday";  
switch (day) {  
  case "Monday":  
    console.log("It's the start of the workweek.");  
    break;  
  case "Friday":  
    console.log("It's almost the weekend.");  
    break;  
  default:  
    console.log("It's some other day.");  
}
```

Fig. 2.2: Switch Statements

All in one example of Conditional logic in expression as shown in Figure 2.3.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Conditional Logic Example</title>
</head>
<body>
<h1>Conditional Logic Example</h1>
<script>
  // Variables
  let temperature = 25;
  let isSunny = true;
  let isWeekend = false;
  // Comparison operators
  let isHot = temperature > 30;
  // Ternary operator
  let weatherMessage = isSunny ? "It's a sunny day!" : "It's not sunny today.";
  // Logical operators
  let outdoorActivity = isSunny && !isHot;
  // Nested if statements
  if (outdoorActivity) {
    if (isWeekend) {
      console.log("Enjoy outdoor activities on the weekend.");
    } else {
      console.log("Enjoy outdoor activities on weekdays.");
    }
  } else {
    console.log("Consider indoor activities.");
  }
  // Switch statement
  switch (temperature) {
    case 30:
      console.log("It's 30°C.");
      break;
    case 25:
      console.log("It's 25°C.");
      break;
    default:
      console.log("Temperature is not 30°C or 25°C.");
  }
}
</script>
</body>
</html>

```

Fig. 2.3: Conditional logic

Output

Conditional Logic Example

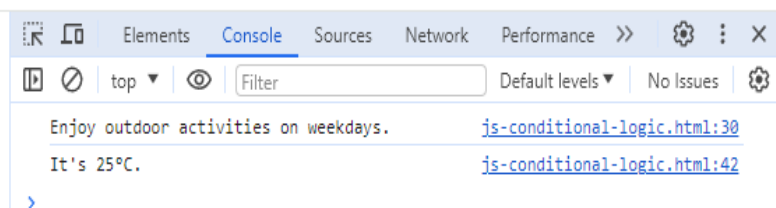


Fig. 2.4: Output for Conditional logic

Code Explanation

- 1. HTML Structure:** This section defines the basic HTML structure for the webpage, including the document type (`<!DOCTYPE html>`), the HTML tag, the character set, and the title of the page.
- 2. JavaScript Code:** The JavaScript code is enclosed within the `<script>` tags and runs within the webpage's context.
- 3. Variable Declarations:** `let temperature = 25;` Declares a variable `temperature` with a value of 25, representing the current temperature.
let isSunny = true; Declares a variable `isSunny` with a value of `true`, indicating sunny weather.
let isWeekend = false; Declares a variable `isWeekend` with a value of `false`, indicating it's not the weekend.
- 4. Comparison Operator:** `let isHot = temperature > 30;` Compares the `temperature` variable with 30 using the greater-than (`>`) operator. It assigns `true` to `isHot` if the temperature is greater than 30, otherwise `false`.
- 5. Ternary Operator:** `let weatherMessage = isSunny ? "It's a sunny day!" : "It's not sunny today.";` Uses the ternary operator (`? :`) to create a `weatherMessage` based on whether `isSunny` is `true` or `false`.
- 6. Logical Operator:** `let outdoorActivity = isSunny && !isHot;` Uses the logical AND (`&&`) and NOT (`!`) operators to determine if it's suitable for outdoor activities based on sunny weather and not being too hot.
- 7. Nested if Statements:** These nested if statements check if `outdoorActivity` is `true` and if it's the weekend (`isWeekend`). Depending on these conditions, it logs messages to the console.
- 8. Switch Statement:** The switch statement checks the value of `temperature` and executes different code blocks based on the `temperature` value.

The code provides a simple example of how conditional logic works in JavaScript, making decisions and displaying messages based on various conditions.

Assignment 2.1.

- Write down the syntax of the following statements:
- if
 - Else if
 - Switch
 - Nested if

2.2 Flow control in JavaScript

Flow control in JavaScript refers to the mechanisms and structures used to control the execution flow of a program. It allows you to make decisions, loop through code, and execute different code blocks based on conditions. Control structures in programming refer to the constructs and statements that allow you to control the flow of execution in your code. These structures help you make decisions, repeat tasks, and manage the logical flow of your program.

Here are some key flow control concepts in JavaScript:

1. Conditional Statements code as shown in Figure 2.5.

```
// If Statements: An if statement allows you to execute a block of code if a specified condition is true
if (condition) {
    // Code to execute if the condition is true
}

// Else Statements: You can use an else statement to specify a block of code to be executed if the
// condition in the if statement is false.
if (condition) {
    // Code to execute if the condition is true
}else{
    // Code to execute if the condition is false
}

// Else If Statements: To handle multiple conditions, you can use else if statements in conjunction with
// if and else statements.
if (condition1) {
    // Code to execute if the condition1 is true
}else if (condition2) {
    // Code to execute if the condition2 is true
}else{
    // Code to execute if none of the conditions is true
}
```

Fig. 2.5: Conditional Statements

2. Switch Statement code as shown in Figure 2.6.

A switch statement allows you to evaluate an expression against multiple possible case values and execute different code blocks based on the matched case.

```
switch (expression) {
    case value1:
        //Code to execute if expression matches value1
        break;
    case value2:
        //Code to execute if expression matches value2
        break;
    default:
        //Code to execute if no case matches the expression
}
```

Fig. 2.6: Switch Statements

3. Break and Continue

The break statement is used to exit a loop prematurely, while the continue statement is used to skip the current iteration and proceed to the next iteration of a loop.

Example of break:

```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        break; // Exit the loop when i is 5
    }
    // Code to execute in each iteration
}
```

Example of continue:

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    continue; // Skip the rest of the code in this iteration when i is 5
  }
  // Code to execute in each iteration except when i is 5
}
```

Did You Know?

Flow Control in JavaScript is the order in which statements are executed in a program.

4. Loops

Loops in programming are control structures that allow you to execute a block of code repeatedly. They are essential for performing tasks that require repetitive actions or for processing collections of data. JavaScript provides several types of loops:

i. For Loop: The for loop is used when you know in advance how many times you want to repeat a block of code. It consists of three parts: initialization, condition, and iteration.

Syntax:

```
for (initialization; condition; iteration) {
  // Code to be executed in each iteration
}
```

Example

```
for (let i = 0; i < 5; i++) {
  console.log("Iteration " + i);
}
```

In the example above, the loop will execute five times, with the i variable taking on values from 0 to 4.

ii. while Loop:

The while loop is used when you want to repeat a block of code as long as a specified condition is true. It evaluates the condition before each iteration.

Syntax:

```
while (condition) {
  // Code to be executed as long as the condition is true
}
```

Example

```
let count = 0;
while (count < 3) {
  console.log("Count is " + count);
  count++;
}
```

The loop in this example will execute three times.

iii. do...while Loop:

The do...while loop is similar to the while loop but guarantees that the code block is executed at least once before checking the condition.

It evaluates the condition after each iteration.

Syntax:

```
do {
    // Code to be executed at least once
} while (condition);
```

Example

```
let x = 0;
do {
    console.log("x is " + x);
    x++;
} while (x < 3);
```

Assignment 2.2.

- List down the syntax of the following:
 - do.....while Loop
 - while Loop
 - break and continue
 - for Loop
 - for...of and for...in Loops

This loop will also execute three times.

iv. for...of and for...in Loops (Iterating Over Collections):

The for...of loop is used to iterate over elements of an iterable (e.g., arrays, strings).

The for...in loop is used to iterate over the properties of an object.

These loops simplify the process of iterating over collections of data.

Here's a brief summary of each loop's purpose and when to use it:

- for loop: Use when you know the number of iterations in advance.
- while loop: Use when you want to repeat code based on a condition.
- do...while loop: Use when you want to ensure that the code block is executed at least once.
- for...of loop: Use to iterate over elements in arrays and other iterables.
- for...in loop: Use to iterate over object properties.

Loops are powerful tools for automating repetitive tasks and processing data, making them a fundamental part of programming. All in one example of Loops as shown in Figure 2.7.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Conditional Logic Example</title>
</head>
<body>
  <h1>Conditional Logic Example</h1>

  <script>
    // Variables
    let temperature = 25
    let isSunny = true
    let isWeekend = false
    // Comparison Operators
    let isHot = temperature > 30
    // Ternary Operator
    let weatherMessage = isSunny ? "It's a sunny day!" : "It's not a sunny day today."
    // Logical Operators
    let outdoorActivity = isSunny && !isHot
    // Nested if Statements
    if (outdoorActivity) {
      if (isWeekend) {
        console.log("Enjoy outdoor activities on the weekend.")
      }else{
        console.log("Enjoy outdoor activities on weekdays.")
      }
    }else{
      console.log("Consider indoor activities")
    }

    // Switch statement
    switch (temperature) {
      case 30:
        console.log("It's 30 degree celcius")
        break
      case 25:
        console.log("It's 25 degree celcius")
      default:
        console.log("Temperatore is not 30 or 25 degree celcius")
    }
  </script>
</body>
</html>

```

Fig. 2.7: Conditional Logic of loops in JavaScript

Output

Conditional Logic Example

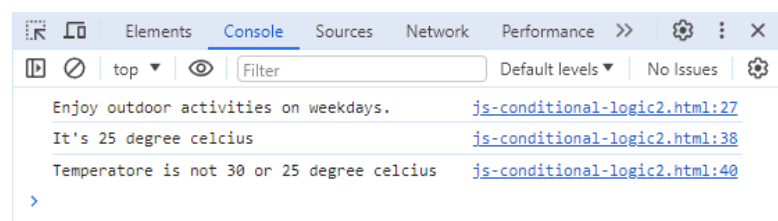


Fig. 2.7: Output for Conditional Logic of loops in JavaScript

Code Explanation:

- These lines declare and initialize three variables: temperature, isSunny, and dayOfWeek.
- The temperature is set to 28, isSunny is set to true, and dayOfWeek is set to "Saturday". These variables store information about the current temperature, weather condition, and the day of the week.
- Here, we have a series of conditional statements using if, else if, and else.
- The first if statement checks if the temperature is greater than 30. If true, it logs "It's a hot day!" to the console.
- The else if statement checks if the temperature is greater than 20. If true, it logs "It's a warm day." to the console.
- If neither of the above conditions is met, the else block is executed, and "It's a cool day." is logged.
- This is a switch statement that checks the value of dayOfWeek.
- If dayOfWeek is "Saturday" or "Sunday", it logs "It's the weekend!" to the console.
- If dayOfWeek is one of the weekdays (Monday to Friday), it logs "It's a weekday."
- If dayOfWeek doesn't match any of the cases, the default block is executed, and "Invalid day." is logged.
- These lines demonstrate different types of loops:
 - The first loop is a for loop that iterates from 0 to 4 (5 times) and logs "Iteration " followed by the current value of i.
 - The second loop is a while loop that runs as long as count is less than 3 and logs "Count is " followed by the current value of count. It increments count in each iteration.
 - The third loop is a do...while loop that executes the code block at least once and then checks the condition. It logs "x is " followed by the current value of x and increments x until it's no longer less than 3.
- This section combines conditional logic with a for loop.
- It iterates from 0 to 4 and checks if each value of i is even or odd.
- If i is even (remainder of division by 2 is 0), it logs "i is even."
- If i is odd, it logs "i is odd."

These lines of code demonstrate various flow control structures and loops, along with how they can be used to make decisions, iterate through data, and control program execution based on conditions.

SUMMARY

- Conditional logic in JavaScript involves making decisions based on conditions using if, else if, and else statements.
- The ternary operator offers a concise way to construct conditional expressions.
- Comparison operators are used to compare values in JavaScript.
- Logical operators perform logical operations on Boolean values.
- Nested if statements are used for multiple levels of conditional checks.
- Switch statements provide an organized alternative to multiple if...else if...else statements.

- Flow control in JavaScript includes conditional statements, switch statements, break and continue, and various types of loops.
- Loops, such as for, while, and do...while, allow you to repeat code based on conditions or a known number of iterations.
- for...of and for...in loops are used to iterate over collections of data and object properties.
- These control structures are fundamental for automating tasks and managing program flow in JavaScript.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What JavaScript construct is used to execute a block of code only if a specified condition holds true? (a) switch statement (b) while loop (c) if statement (d) for loop
2. Which JavaScript operator is used for strict equality, checking both value and data type? (a) == (b) === (c) != (d) !==
3. The "else if" statement is used to define and evaluate additional conditions when the preceding "if" condition is: (a) true (b) false (c) equal (d) undefined
4. What JavaScript logical operator returns true if both operands are true? (a) && (b) || (c) ! (d) &
5. What keyword is used to exit a loop prematurely in JavaScript? (a) exit (b) quit (c) break (d) skip
6. Which JavaScript loop is used when you know in advance how many times you want to repeat a block of code? (a) while loop (b) do...while loop (c) for loop (d) for...of loop
7. In a "switch" statement, which keyword is used to execute code when none of the case values match the expression? (a) default (b) none (c) other (d) unmatched
8. What does the "continue" statement do in JavaScript? (a) Exits the loop (b) Skips the rest of the code in the current iteration and proceeds to the next (c) Resets the loop (d) Restarts the loop from the beginning
9. Which of the following is not a common comparison operator in JavaScript? (a) < (b) >= (c) || (d) !==
10. When using the ternary operator in JavaScript, what symbol separates the value to return if the condition is true and the value to return if the condition is false? (a) ; (b) : (c) | (d) ,

B. Fill in the blanks

1. In JavaScript, the "if" statement is used to run a block of code only if a specified condition holds _____.
2. The ternary operator offers a concise method for constructing _____ expressions.
3. Comparison operators in JavaScript are used to _____ values and return a Boolean result.
4. Logical operators in JavaScript are used to perform logical operations on _____ values.

5. _____ if statements are used when you want to have multiple levels of conditional checks.
6. A _____ statement is used to perform different actions based on different conditions.
7. The _____ statement is used to exit a loop prematurely.
8. The _____ statement is used to skip the current iteration and proceed to the next iteration of a loop.
9. The for...of loop is used to _____ over elements of an iterable, such as arrays or strings.
10. The _____ loop is used to iterate over the properties of an object.

C. True or False

1. The "if" statement in JavaScript is used to run a block of code only if a specified condition holds true.
2. The "else if" statement is used to evaluate additional conditions only when the preceding "if" condition is true.
3. The ternary operator in JavaScript provides a concise way to construct conditional expressions.
4. Comparison operators in JavaScript are used to compare values and return a Boolean result.
5. Logical operators in JavaScript are used to perform logical operations on Boolean values.
6. Nested if statements are used to create multiple levels of conditional checks.
7. A switch statement in JavaScript can replace multiple "if...else if...else" statements.
8. The "break" statement is used to exit a loop prematurely.
9. The "continue" statement is used to skip the current iteration and proceed to the next iteration of a loop.
10. The "for" loop is used when you know in advance how many times you want to repeat a block of code.

D. Short Question Answers

1. What is conditional logic in JavaScript, and why is it important in programming?
2. Explain the basic syntax and usage of the "if" statement in JavaScript.
3. How does the "else if" statement differ from the "if" statement, and when is it used?
4. Describe the purpose and syntax of the "else" statement in JavaScript.
5. What is the ternary operator in JavaScript, and how does it work?
6. List some common comparison operators in JavaScript and explain their use cases.
7. What are logical operators in JavaScript, and give examples of their usage.
8. When is it appropriate to use nested if statements, and how do they work?
9. What is a switch statement, and how does it simplify code compared to multiple if...else if...else statements?
10. Briefly explain the types of loops in JavaScript and provide an example use case for each.

Session 3. Arrays and Functions

Ankit, a kid who loved puzzles and creating, discovered "Arrays and Functions," which were like having a treasure chest full of items and magical tools to do things with them. Arrays helped him organize items neatly, while functions were like robot friends that followed his commands. He used functions to create games where players collected items and counted scores and even made a calculator function to add numbers. Arrays and functions were his secret helpers, making the digital world more fun and allowing him to create all sorts of cool things, like a treasure chest full of digital adventures. As shown in figure 3.1.



Fig. 3.1: Ankit using Arrays and functions

In this chapter, you will learn about Arrays, Array Methods, Arrays and Loops, Functions, Function Return, Functions and Differences between Function Parameters and Arguments.

3.1 Arrays

Creating and populating arrays is a fundamental concept in programming, and it allows you to store and manage collections of data. In JavaScript, you can create arrays and populate them in several ways:

Array Literal: The most common way to create an array is by using square brackets []. You can populate it with values separated by commas.

Example: `let fruits = ["apple", "banana", "cherry"];`

Array Constructor: You can create an array using the Array constructor and passing the values as arguments.

Example: `let colors = new Array("red", "green", "blue");`

Empty Array: You can create an empty array and then populate it by assigning values to specific indices.

Example:

```
let cars = [];  
cars[0] = "Toyota";  
cars[1] = "Honda";
```

Array Methods: JavaScript provides array methods like `push()`, `unshift()`, `concat()`, and `splice()` to add elements to an existing array.

Example using `push()`:

```
let animals = ["dog", "cat"];  
animals.push("elephant");
```

Using Loops: You can use loops like for or while to populate an array dynamically.

Example using a for loop:

```
let numbers = [];
for (let i = 1; i <= 5; i++) {
  numbers.push(i);
}
```

Array Destructuring: You can destructure arrays to extract and populate variables.

Example:

```
let [firstName, lastName] = ["John", "Doe"];
```

Spread Operator: The spread operator (...) can be used to copy the elements of one array into another.

Example:

```
let fruits = ["apple", "banana"];
let moreFruits = [...fruits, "cherry", "orange"];
```

Array Comprehensions (ES6): In modern JavaScript (ES6 and later), you can use array comprehensions to create and populate arrays with concise syntax.

Example:

```
let numbers = [1, 2, 3, 4, 5];
let squaredNumbers = [x * x for (x of numbers)];
```

Using Array.from() (ES6): The Array.from() method can create an array from an iterable or array-like object.

Example:

```
let arrayFromSet = Array.from(new Set([1, 2, 2, 3, 4]));
```

3.2 Array Methods

Array methods are built-in functions in JavaScript that allow you to manipulate arrays. Here are some commonly used array methods:

Push (): Adds one or more elements to the end of an array and returns the new length of the array.

```
let fruits = ["apple", "banana"];
fruits.push("cherry");
// fruits is now ["apple", "banana", "cherry"]
```

pop(): Removes the last element from an array and returns that element.

```
let fruits = ["apple", "banana", "cherry"];
let removedFruit = fruits.pop();
// fruits is now ["apple", "banana"], and removedFruit is "cherry"
```

shift(): Removes the first element from an array and returns that element.

```
let fruits = ["apple", "banana", "cherry"];
let removedFruit = fruits.shift();
// fruits is now ["banana", "cherry"], and removedFruit is "apple"
```

unshift(): Adds one or more elements to the beginning of an array and returns the new length of the array.

```
let fruits = ["apple", "banana"];
fruits.unshift("cherry");
// fruits is now ["cherry", "apple", "banana"]
```

concat(): Combines two or more arrays and returns a new array.

```
let fruits1 = ["apple", "banana"];
let fruits2 = ["cherry", "orange"];
let combinedFruits = fruits1.concat(fruits2);
// combinedFruits is ["apple", "banana", "cherry", "orange"]
```

slice(): Returns a new array containing a portion of the original array.

```
let fruits = ["apple", "banana", "cherry", "orange"];
let slicedFruits = fruits.slice(1, 3);
// slicedFruits is ["banana", "cherry"]
```

forEach(): Executes a provided function once for each array element.

```
let numbers = [1, 2, 3];
numbers.forEach(function (number) {
  console.log(number);
});
// Outputs: 1, 2, 3
```

map(): Creates a new array with the results of calling a provided function on every element in the array.

```
let numbers = [1, 2, 3];
let doubledNumbers = numbers.map(function (number) {
  return number * 2;
});
// doubledNumbers is [2, 4, 6]
```

filter(): Creates a new array with all elements that pass a provided test.

```
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = numbers.filter(function (number) {
  return number % 2 === 0;
});
// evenNumbers is [2, 4]
```

Assignment 3.1.

- List down the details of the following array methods:
 - push()
 - filter()
 - map()
 - concat()
 - slice()

3.3 Arrays and Loops

Arrays and loops often go hand in hand. You can use loops to iterate over the elements of an array and perform operations on them. Here are examples of how you can work with arrays using loops as shown in Figure 3.2:

for Loop:

```
let fruits = ["apple", "banana", "cherry"];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
// Outputs: "apple", "banana", "cherry"
```

forEach() Method:

```
let fruits = ["apple", "banana", "cherry"];
fruits.forEach(function (fruit) {
  console.log(fruit);
});
// Outputs: "apple", "banana", "cherry"
```

for...of Loop (ES6):

```
let fruits = ["apple", "banana", "cherry"];
for (let fruit of fruits) {
  console.log(fruit);
}
// Outputs: "apple", "banana", "cherry"
```

Did You Know?

Arrays and loops provide a versatile and efficient way to work with collections of data in JavaScript

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Array Example</title>
</head>
<body>
  <h1>Array Example</h1>
  <script>
    // Creating an array
    let fruits = ["apple", "banana", "cherry", "date", "elderberry"];
    // 1. Using forEach to log each fruit
    console.log("Using forEach:");
    fruits.forEach(function (fruit) {
      console.log(fruit);
    });
    // 2. Using map to create a new array with fruit lengths
    console.log("\nUsing map to get fruit lengths:");
    let fruitLengths = fruits.map(function (fruit) {
      return fruit.length;
    });
    console.log(fruitLengths);
    // 3. Using filter to get fruits with even-length names
    console.log("\nUsing filter to get fruits with even-length names:");
    let evenLengthFruits = fruits.filter(function (fruit) {
      return fruit.length % 2 === 0;
    });
    console.log(evenLengthFruits);
    // 4. Using reduce to concatenate all fruit names
    console.log("\nUsing reduce to concatenate all fruit names:");
    let allFruits = fruits.reduce(function (accumulator, fruit) {
      return accumulator + ", " + fruit;
    });
    console.log(allFruits);
    // Using array loops
    // 5. Using a for loop to log each fruit
    console.log("\nUsing a for loop:");
    for (let i = 0; i < fruits.length; i++) {
      console.log(fruits[i]);
    }
    // 6. Using a for...of loop to log each fruit
    console.log("\nUsing a for...of loop:");
    for (let fruit of fruits) {
      console.log(fruit);
    }
  </script>
</body>
</html>

```

Fig. 3.2: Array Example

Output

```

Array Example

Using forEach:
apple
banana
cherry
date
elderberry

Using map to get fruit lengths:
▶ Array(5)

Using filter to get fruits with even-length names:
▶ Array(4)

Using reduce to concatenate all fruit names:
apple, banana, cherry, date, elderberry

Using a for loop:
apple
banana
cherry
date
elderberry

Using a for...of loop:
apple
banana
cherry
date
elderberry

```

*Fig. 3.3: Output for Array Example***Code Explanation**

- This line declares an array named fruits and initializes it with a list of fruit names.
- This section demonstrates the use of array methods:
 - `forEach()` is used to iterate through each element (fruit) in the fruits array.
 - For each fruit, it executes the provided function, which logs the fruit to the console.
- Here, we use the `map()` method to create a new array called `fruitLengths`.
 - `map()` iterates through each element (fruit) in the fruits array.
 - For each fruit, it applies the provided function, which calculates the length of the fruit name.
 - The resulting array contains the lengths of all fruit names.
- We use the `filter()` method to create a new array called `evenLengthFruits`.
 - `filter()` iterates through each element (fruit) in the fruits array.
 - For each fruit, it applies the provided function, which checks if the length of the fruit name is even.
 - The resulting array contains only the fruits with even-length names.
- We use the `reduce()` method to concatenate all fruit names into a single string named `allFruits`.

- `reduce()` iterates through each element (fruit) in the fruits array.
- It applies the provided function, which takes an accumulator (initially an empty string) and concatenates it with each fruit name.
- The final result is a single string with all fruit names separated by commas.
- demonstrates a for loop for iterating through the fruits array:
 - The loop initializes `i` to 0 and continues as long as `i` is less than the length of the fruits array.
 - In each iteration, it logs the current fruit using the index `i`.
- Finally, we use a `for...of` loop to achieve the same iteration as the `forEach()` method:
 - It directly iterates through each element (fruit) in the fruits array.
 - In each iteration, it logs the current fruit to the console.

This example demonstrates various techniques for working with arrays, array methods for data manipulation, and different types of loops for iterating over array elements.

3.4 Functions

Functions in JavaScript are blocks of reusable code. It is designed to perform a specific task. They are a fundamental part of JavaScript programming and play a crucial role in structuring and organizing your code. Functions can take inputs, perform operations, and produce outputs. Here's a detailed explanation of functions and how to use them:

3.4.1 Function Declaration:

A function is declared using the `function` keyword, followed by a name for the function, a pair of parentheses `()`, and a pair of curly braces `{}` that enclose the function's code block.

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
```

- `function` is the keyword used to declare a function.
- `greet` is the name of the function.
- `(name)` defines a parameter named `name`. Parameters are variables that hold values passed to the function.
- `{ ... }` contains the code that the function executes.

3.5 Function Invocation (Calling a Function):

To execute a function, you "call" it by using its name followed by parentheses, and you can pass arguments (values) to the function's parameters.

```
greet("Alice"); // Calling the greet function with "Alice" as the argument.
```

- `greet("Alice")` is the function invocation. Here, "Alice" is an argument passed to the `name` parameter.

Function Return: Functions can return values using the `return` statement. A function can have zero or one return statement. When a function executes a return, it immediately exits, and the specified value is passed back to the caller.

```
function add(a, b) {
  return a + b;
}
let result = add(5, 3); // result will be 8
```

- In the add function, return a + b; returns the sum of a and b.
- The result of add(5, 3) is 8, which is stored in the variable result.

Function Scope: Variables declared inside a function are "local" to that function, meaning they can only be accessed within the function. This concept is known as "function scope."

```
function multiply(x, y) {
  let product = x * y; // product is a local variable
  return product;
}
```

console.log(product); // Error: product is not defined outside the function.

- The product variable is declared inside the multiply function and is only accessible within that function.

Differences between Function Parameters and Arguments:

Table 3.1. Differences between Function Parameters and Arguments

Parameters	Arguments
Placeholders in the function definition represent the values to be passed into the function.	Actual values or expressions provided when calling the function.
Defined in the function signature or declaration.	Supplied at the time of invoking the function.
Serve as local variables within the function's scope.	Provide the data for the function to operate on.
The number of parameters can vary, including none or multiple.	Should match the number of parameters in the function when called.
Have variable names within the function's scope.	Do not have variable names within the function's scope.
It can use default values to handle missing arguments.	Omitting arguments for required parameters results in undefined.
It can be modified within the function without affecting the original arguments.	Cannot be directly modified within the function.

3.6 Built-in functions

Built-in functions, also known as native functions or standard library functions, are functions that come pre-defined in JavaScript. They serve various purposes and provide essential capabilities for working with data, performing calculations, manipulating strings, handling dates, interacting with the DOM (Document Object Model), and more. Here are some common categories of built-in functions in JavaScript as shown in Figure 3.4:

Math Functions:

- Math.sqrt(x): Returns the square root of x.
- Math.pow(x, y): Returns x raised to the power of y.
- Math.random(): Generates a random number between 0 (inclusive) and 1 (exclusive).

String Functions:

- `str.length`: Provides the string's length.
- `str.toUpperCase()`: Changes the string to uppercase.
- `str.toLowerCase()`: Converts the string to lowercase.
- `str.indexOf(substring)`: Offers the index of the first appearance of the substring in the string.

Array Functions:

- `array.length`: Returns the number of elements in an array.
- `array.push(element)`: Adds an element to the end of an array.
- `array.pop()`: Removes and returns the last element of an array.
- `array.join(separator)`: Converts an array to a string with elements separated by separator.

Date Functions:

- `new Date()`: Creates a new Date object representing the current date and time.
- `date.getFullYear()`: Returns the year of a Date object.
- `date.getMonth()`: Returns the month (0-11) of a Date object.
- `date.getDate()`: Returns the day of the month (1-31) of a Date object.

DOM Manipulation Functions:

- `document.getElementById(id)`: Retrieves an HTML element with the specified id.
- `document.querySelector(selector)`: Retrieves the first HTML element that matches the selector.
- `element.addEventListener(event, handler)`: Attaches an event handler to an HTML element.

Type Conversion Functions:

- `parseInt(string)`: Parses a string and returns an integer.
- `parseFloat(string)`: Parses a string and returns a floating-point number.
- `String(value)`: Converts a value to a string.

Regular Expression Functions:

- `regex.test(string)`: Tests if a string matches a regular expression.
- `string.match(regex)`: Searches a string for a regular expression pattern and returns matching results.

Global Functions:

- `isNaN(value)`: Checks if a value is NaN (Not-a-Number).
- `parseInt(string, radix)`: Parses a string as an integer with the specified radix.

These are just a few examples of the many built-in functions available in JavaScript. You can use them to simplify and enhance your code by leveraging their functionality to perform common tasks efficiently.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript in-built Functions</title>
</head>
<body>
  <h1>Functions Example</h1>
  <!-- Add an HTML element with the ID "myElement" -->
  <div id="myElement">Click Me</div>

  <script>
    // Wrap your JavaScript code in the DOMContentLoaded event listener
    document.addEventListener("DOMContentLoaded", function() {
      // Math Functions
      let squareRoot = Math.sqrt(25)
      let power = Math.pow(2, 3)
      let randomNum = Math.randomNum
      // String Functions
      let str = "Hello World!"
      let length = str.length
      let uppercaseStr = str.toUpperCase()
      let index = str.indexOf("World")
      // Array Functions
      let fruits = ["apple", "banana", "cherry"]
      let arrayLength = fruits.length
      fruits.push("date")
      let lastFruit = fruits.pop()
      // Date Functions
      let currentDate = new Date()
      let year = currentDate.getFullYear()
      let month = currentDate.getMonth()
      // DOM Manipulation Functions
      let elementById = document.getElementById("myElement")
      // Type Conversion Functions
      let parsedInt = parseInt("42")
      let parsedFloat = parseFloat("3.14")
      let stringValue = String(123)

      // Print values to the console
      console.log("Square Root:", squareRoot)
      console.log("Power:", power)
      console.log("Random Number:", randomNum)
      console.log("Length:", length)
      console.log("Upper Case String:", uppercaseStr)
      console.log("Index:", index)
      console.log("Array Length:", arrayLength)
      console.log("Last Fruit:", lastFruit)

      console.log("Year:", year)
      console.log("Month:", month)
      console.log("Get Element by Id:", elementById)
      console.log("Parsed Integer:", parsedInt)
      console.log("Parsed Float:", parsedFloat)
      console.log("String Value:", stringValue)
    });
  </script>
</body>
</html>

```

Fig. 3.4: Built-in functions in JavaScript

Output

Functions Example

Click Me

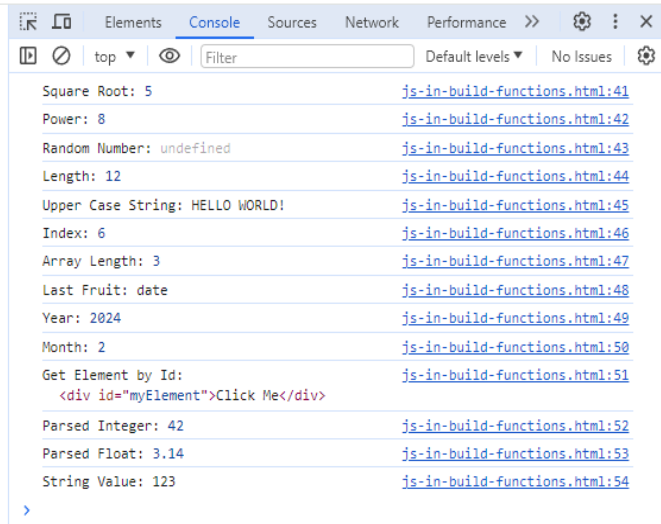


Fig. 3.4: Output for Built-in functions in JavaScript

Example of user defined function as shown in Figure 3.5.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Example of User Defined Functions</title>
</head>
<body>
  <h1>User-Defined Functions Example</h1>

  <script>
    // User-Defined function to calculate the square of a number
    function calculateSquare(number) {
      return number * number
    }

    // Call the User-Defined function and store the result in a variable
    let result = calculateSquare(5)

    // Display the result
    console.log("The square of 5 is: ", result)
  </script>
</body>
</html>
```

Fig. 3.5: user defined function Example

Output

User-Defined Functions Example

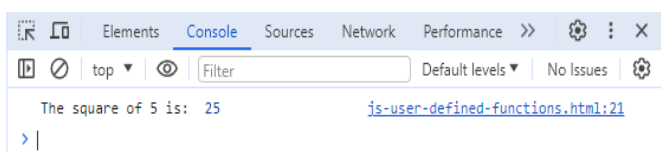


Fig. 3.6: Output for user defined function Example

Assignment 3.2.

- List down the names of built-in functions.
- List down the Differences between Function Parameters and Arguments.
- Write down the name of two functions and their syntax.

SUMMARY

- Arrays are fundamental data structures in JavaScript, allowing the storage and management of collections of data.
- Arrays can be created using array literals, the Array constructor, or by dynamically populating them using methods, loops, and spread operators.
- Array methods such as push, pop, shift, unshift, concat, slice, forEach, map, and filter offer powerful tools for array manipulation.
- Loops like for, forEach, and for...of are used to iterate over array elements and perform operations on them.
- Functions are blocks of reusable code, defined with parameters and a code block, which can be invoked with arguments.
- Functions can return values using the return statement, and variables declared inside functions are "local" to that function's scope.
- Understanding the distinction between function parameters and arguments is crucial for effective function usage.
- Built-in functions in JavaScript provide essential capabilities, including math, string manipulation, array operations, date handling, and DOM manipulation.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What is a common way to create an array in JavaScript? (a) Using parentheses () (b) Using square brackets [] (c) Using angle brackets <> (d) Using curly braces {}
2. Which method is used to add elements to the end of an array? (a) concat() (b) push() (c) unshift() (d) pop()
3. How can you remove the last element from an array and get its value? (a) pop () (b) shift () (c) remove () (d) splice ()
4. Which loop is commonly used to iterate over the elements of an array in JavaScript? (a) for-in loop (b) while loop (c) for loop (d) forEach loop
5. What is the purpose of the map() method in JavaScript arrays? (a) It modifies the original array (b) It creates a new array with transformed elements (c) It removes specific elements from the array (d) It reverses the order of the array
6. How are variables declared inside a JavaScript function scoped? (a) Globally (b) Locally to the function (c) Within the entire document (d) Only in objects
7. In JavaScript, what is the primary purpose of a function declaration? (a) To call other functions (b) To declare a variable (c) To execute a specific task (d) To create an array
8. What is the correct way to call a JavaScript function with arguments? (a) myFunction(arguments) (b) myFunction.arguments() (c) myFunction(arguments) (d) myFunction(arguments, values)
9. What keyword is used to declare a function in JavaScript? (a) func (b) declare (c) function (d) def
10. How is the return statement used in JavaScript functions? (a) It is used to terminate the function (b) It is used to define the function's name (c) It is used to return a value from the function (d) It is used to create loops

B. Fill in the blanks

1. The _____ method is used to remove the last element from an array.
2. Functions are blocks of _____ code that can be reused to perform specific tasks.
3. To execute a function, you need to call it using its name followed by a pair of _____.
4. The _____ statement in a function allows you to return a value from the function.
5. Built-in functions that come pre-defined in JavaScript are often referred to as _____ functions.
6. The _____ function is used to generate a random number between 0 (inclusive) and 1 (exclusive).
7. The _____ function converts a value to a string.
8. The method _____ adds an element to the end of an array and returns the new length of the array.
9. The method _____ removes the first element from an array and returns that element.
10. The _____ in JavaScript is designed to perform a specific task.

C. True or False

1. Arrays are used to store and manage collections of data.
2. You can create an empty array in JavaScript by assigning values to specific indices.
3. The Array constructor can be used to create an array with values.
4. Parameters are values provided when calling a function.
5. In JavaScript, a function can have multiple return statements.
6. A function can modify arguments passed to it directly.
7. The push() method adds elements to the beginning of an array.
8. The shift() method removes the last element from an array.
9. Functions are blocks of reusable code in JavaScript.
10. You can declare and initialize an array using the Array.from() method.

D. Short Question Answers

1. What is an array in JavaScript, and why is it useful?
2. How can you create an array using an array literal in JavaScript?
3. Explain how you can add elements to the end of an array using the push() method.
4. What does the pop() method do, and how is it used with arrays?
5. Describe the concept of "function scope" in JavaScript.
6. What is the difference between function parameters and arguments?
7. How can you declare a function in JavaScript, and what is its structure?
8. What is the purpose of the return statement in a function, and how does it work?
9. Give an example of a built-in function in JavaScript and explain its purpose.
10. What are built-in functions in JavaScript?

Session 4. String Manipulation

Aman, a word-loving kid, discovered "String Manipulation," which was like a word game. String manipulation was all about changing words and sentences in cool ways, like mixing up letters in a puzzle. He learned two exciting tricks: combining strings, where he could join words to create new ones, and changing characters, allowing him to replace or rearrange letters in words. With these tricks, he made up funny stories and even created his own secret language, feeling like a word magician with a magical word toolbox. It was a creative and fun way to play with words, just like solving puzzles or writing cool stories. As shown in Figure 4.1.



Fig.4.1: Aman using String Manipulation

In this chapter, you will learn about String Manipulation, Dates and Time, and Forms.

4.1 String Manipulation

String manipulation in JavaScript involves performing various operations on strings to modify or extract information from them. Here are some common string manipulation techniques and examples as shown in Figure 4.2:

Concatenation: Combining two or more strings.

```
let str1 = "Hello";
let str2 = "World";
let combinedStr = str1 + " " + str2; // "Hello World"
```

String Length: Finding the length of a string.

```
let text = "This is a sample text";
let length = text.length; // 21
```

Substring: Extracting a portion of a string.

```
let text = "Hello, World";
let substring = text.substring(0, 5); // "Hello"
```

String Case: Changing the case of a string.

```
let str = "JavaScript";
let upperCaseStr = str.toUpperCase(); // "JAVASCRIPT"
let lowerCaseStr = str.toLowerCase(); // "javascript"
```

String Search: Finding the position of a substring within a string.

```
let text = "This is a sample text";
let position = text.indexOf("sample"); // 10
```

Replacing Substrings: Replacing occurrences of a substring within a string.

```
let text = "Hello, World";
```

```
let replacedText = text.replace("World", "Universe"); // "Hello, Universe"
```

Splitting Strings: Splitting a string into an array of substrings based on a delimiter.

```
let csvData = "John,Doe,30";
```

```
let dataArray = csvData.split(","); // ["John", "Doe", "30"]
```

String Conversion: Converting other data types to strings.

```
let num = 42;
```

```
let strNum = String(num); // "42"
```

PSSCIVE Draft Study Material © Not to be Published

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>String Manipulation Examples</title>
</head>
<body>
<h1>String Manipulation Examples</h1>
<script>
  // 1. Concatenation
  let str1 = "Hello";
  let str2 = "World";
  let combinedStr = str1 + " " + str2;
  console.log("Concatenation:", combinedStr);

  // 2. String Length
  let text = "This is a sample text";
  let length = text.length;
  console.log("String Length:", length);

  // 3. Substring
  let text2 = "Hello, World";
  let substring = text2.substring(0, 5);
  console.log("Substring:", substring);
  // 4. String Case
  let str3 = "JavaScript";
  let upperCaseStr = str3.toUpperCase();
  let lowerCaseStr = str3.toLowerCase();
  console.log("Upper Case:", upperCaseStr);
  console.log("Lower Case:", lowerCaseStr);

  // 5. String Search
  let text3 = "This is a sample text";
  let position = text3.indexOf("sample");
  console.log("String Search:", position);

  // 6. Replacing Substrings
  let text4 = "Hello, World";
  let replacedText = text4.replace("World", "Universe");
  console.log("Replacing Substrings:", replacedText);

  // 7. Splitting Strings
  let csvData = "John,Doe,30";
  let dataArray = csvData.split(",");
  console.log("Splitting Strings:", dataArray);

  // 8. Trimming
  let text5 = "  Trim me  ";
  let trimmedText = text5.trim();
  console.log("Trimming:", trimmedText);

  // 9. String Conversion
  let num = 42;
  let strNum = String(num);
  console.log("String Conversion:", strNum);

  // 10. Template Literals
  let name = "Alice";
  let greeting = `Hello, ${name}!`;
  console.log("Template Literals:", greeting);
</script>
</body>
```

Fig. 4.2: String manipulation in JavaScript

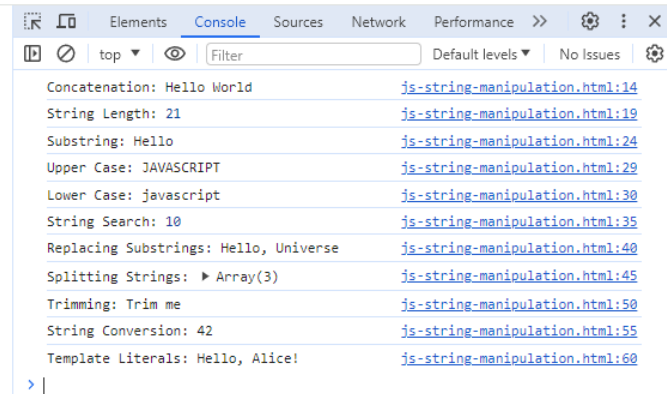
Output**String Manipulation Examples**

Fig. 4.3: Output for String manipulation in JavaScript

indexOf() : The `indexOf()` method is used to find the index of the first occurrence of a substring within a string as shown in Figure 4.4.

Example-

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>indexOf Example</title>
</head>
<body>
<h1>indexOf Example</h1>
<script>
    let text = "This is a sample text. Sample text is here.";
    // Find the index of the first occurrence of "sample"
    let firstIndex = text.indexOf("sample");
    console.log("First Index of 'sample':", firstIndex);
    // Find the index of the second occurrence of "sample"
    let secondIndex = text.indexOf("sample", firstIndex + 1);
    console.log("Second Index of 'sample':", secondIndex);
</script>
</body>
</html>
```

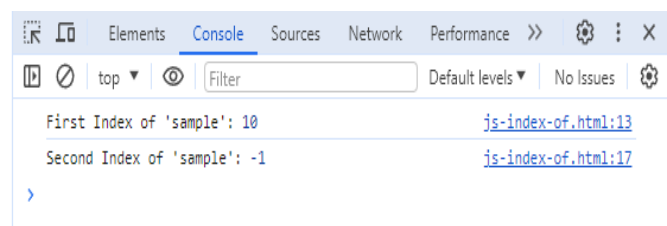
Output**indexOf Example**

Fig. 4.4: output for indexOf() method

charAt(): The charAt() method is used to retrieve the character at a specific index within a string as shown in Figure 4.5.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>charAt Example</title>
</head>
<body>
  <h1>charAt Example</h1>

  <script>
    let text = "Hello, World";

    // Get the character at index 4
    let charAtIndex4 = text.charAt(4);
    console.log("Character at Index 4:", charAtIndex4);

    // Get the character at index 8
    let charAtIndex8 = text.charAt(8);
    console.log("Character at Index 8:", charAtIndex8);
  </script>
</body>
</html>
```

Fig. 4.5: charAt() method

Output



Fig. 4.6: Output for charAt() method

substr() : The JavaScript String substr() method extracts a part of a string. The substr() method begins at a specified position, and returns a specified number of characters. The substr() method does not change the original string.

The syntax for the substr() method is:

string.substr(start, length)

Here, start: The start position, where 0 is the first character.

length: The number of characters to return.

Some examples to use the substr() method:

"Hello, world!".substr(0, 5) // "Hello"

"Hello, world!".substr(5) // "world!"

"Hello, world!".substr(-6) // "world!"

"Hello, world!".substr(-6, 5) // "world"

split() : The split() method in JavaScript is used to split a string into an array of substrings. The split() function takes one or two optional parameters, the first one being the separator, and the second the maximum number of elements to be included in the resulting array.

var str = "The quick brown fox jumps over the lazy dog.";

var substrings = str.split(" ");

Did You Know...

If (" ") is used as separator, the string is split between words.

join(): The join() method in JavaScript concatenates all elements of an array into a string, separated by a separator. The join() method can be used to combine two or more strings into a single string. For example, the following code concatenates the strings "Hello" and "World" into a single string, separated by a comma:

```
const str1 = "Hello";
const str2 = "World";
const separator = ", ";
const result = str1.concat(str2, separator);
console.log(result); // "Hello, World"
```

The join() method can also be used to combine the elements of an array into a single string. For example, the following code concatenates the elements of the array arr into a single string, separated by a comma:

```
const arr = ["Hello", "World"];
const separator = ", ";
const result = arr.join(separator);
console.log(result); // "Hello, World"
```

Assignment 4.1.

- Write down the purpose of the following methods:
 - Join()
 - charAt()
 - indexOf()
 - substr()

4.2 Dates and Time

In JavaScript, dates and times are represented using the Date object. The Date object provides information about dates and times, and also provides various methods.

Did You Know?

A JavaScript date defines the EcmaScript epoch, which represents milliseconds since 1 January 1970 UTC.

Here are some common tasks related to dates and times in JavaScript:

Creating a Date Object: You can create a Date object to represent a specific date and time.

```
let currentDate = new Date(); // Current date and time
let specificDate = new Date("2023-10-05T12:00:00"); // Specific date and time
```

Getting Date Components: You can extract various components (year, month, day, etc.) from a Date object.

```
let year = currentDate.getFullYear(); // Get the year
let month = currentDate.getMonth(); // Get the month (0-11)
let day = currentDate.getDate(); // Get the day of the month (1-31)
let hours = currentDate.getHours(); // Get the hours (0-23)
let minutes = currentDate.getMinutes(); // Get the minutes (0-59)
```



```
let seconds = currentDate.getSeconds(); // Get the seconds (0-59)
```

Formatting Dates: You can format dates as strings using various methods like `toLocaleDateString()` or libraries like `moment.js` (if used).

```
let formattedDate = currentDate.toLocaleDateString("en-US"); // Format as "MM/DD/YYYY"
```

Working with Timestamps: Timestamps represent the number of milliseconds since January 1, 1970 (UTC).

```
let timestamp = Date.now(); // Current timestamp
```

Comparing Dates: You can compare dates to check if one is before, after, or equal to another.

```
let date1 = new Date("2023-10-05");
let date2 = new Date("2023-10-10");
if (date1 < date2) {
  console.log("date1 is before date2");
} else if (date1 > date2) {
  console.log("date1 is after date2");
} else {
  console.log("date1 is equal to date2");
}
```

Working with Time Zones:

JavaScript's `Date` object works with the local time zone. You can convert between time zones using libraries or by adjusting the date manually.

```
// Adjusting for a specific time zone (e.g., UTC)
currentDate.setHours(currentDate.getHours() - 4);
```

Date Validation:

You can validate user-entered dates to ensure they are in a valid format.

```
function isValidDate(dateStr) {
  return !isNaN(Date.parse(dateStr));
}
```

Date Arithmetic:

You can perform operations like adding or subtracting days from a date as shown in Figure 4.7.

```
let futureDate = new Date();
futureDate.setDate(currentDate.getDate() + 7); // Add 7 days to the current date
```

```

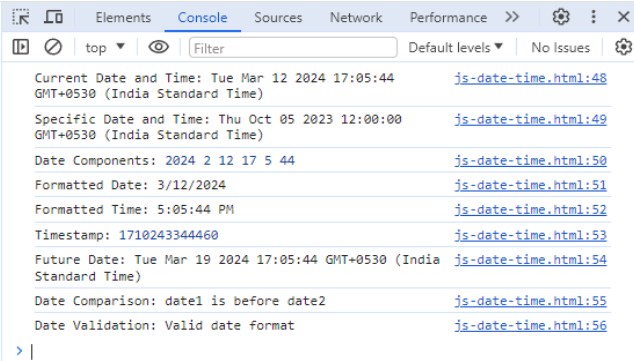
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Date and Time Example</title>
</head>
<body>
  <h1>Date and Time Example</h1>
  <script>
    // 1. Creating a Date Object
    let currentDate = new Date()
    let specificDate = new Date("2023-10-05T12:00:00")
    // 2. Getting Date Components
    let year = currentDate.getFullYear()
    let month = currentDate.getMonth()
    let day = currentDate.getDate()
    let hours = currentDate.getHours()
    let minutes = currentDate.getMinutes()
    let seconds = currentDate.getSeconds()
    // 3. Formatting Dates
    let formattedDate = currentDate.toLocaleDateString("en-US")
    let formattedTime = currentDate.toLocaleTimeString("en-US")
    // 4. Working with TimeStamps
    let timestamp = Date.now()
    // 5. Date Arithmetic
    let futureDate = new Date()
    futureDate.setDate(currentDate.getDate() + 7)
    // 6. Comparing Dates
    let date1 = new Date("2023-10-05")
    let date2 = new Date("2023-10-10")
    let comparisonResult = ""
    if (date1 < date2) {
      comparisonResult = "date1 is before date2"
    }else if (date1 > date2) {
      comparisonResult = "date1 is after date2"
    }else{
      comparisonResult = "date1 is equal to date2"
    }
    // 7. Date Validation
    function isValidDate(dateStr) {
      return !isNaN(Date.parse(dateStr))
    }
    let userInput = "2023-10-15"
    let isValid = isValidDate(userInput)
    let validationMessage = isValid ? "Valid date format" : "Invalid date format"
    // Displaying Results
    console.log("Current Date and Time:", currentDate)
    console.log("Specific Date and Time:", specificDate)
    console.log("Date Components:", year, month, day, hours, minutes, seconds)
    console.log("Formatted Date:", formattedDate)
    console.log("Formatted Time:", formattedTime)
    console.log("Timestamp:", timestamp)
    console.log("Future Date:", futureDate)
    console.log("Date Comparison:", comparisonResult)
    console.log("Date Validation:", validationMessage)
  </script>
</body>
</html>

```

Fig. 4.7: Date Arithmetic Date and Time example

Output

Date and Time Example



```

Current Date and Time: Tue Mar 12 2024 17:05:44 GMT+0530 (India Standard Time)
Specific Date and Time: Thu Oct 05 2023 12:00:00 GMT+0530 (India Standard Time)
Date Components: 2024 2 12 17 5 44
Formatted Date: 3/12/2024
Formatted Time: 5:05:44 PM
Timestamp: 1710243344460
Future Date: Tue Mar 19 2024 17:05:44 GMT+0530 (India Standard Time)
Date Comparison: date1 is before date2
Date Validation: Valid date format

```

Fig. 4.8: Output for Date Arithmetic Date and Time example

4.3 Forms

Creating and working with forms in JavaScript involves manipulating the Document Object Model (DOM) to create form elements, add event listeners, and handle user input.

The validateForm Function:

- This function is typically used as the form's onsubmit handler to validate all form fields before submission.
- It prevents form submission if any of the validation functions return false, indicating that there are validation errors.
- In this function, you can call other validation functions for specific input fields and return true only if all validations pass.

The checkEmail Function:

- This function is used to validate an email input field to ensure it has a valid email format.
- It takes an email input field as a parameter and checks if its value matches a regular expression pattern for a valid email format.
- If the input value is a valid email, the function returns true; otherwise, it returns false.

The checkRadio Function:

- This function is used to validate a group of radio buttons to ensure at least one option is selected.
- It takes a radio button group as a parameter, which is typically an array of radio button elements.
- The function loops through the radio buttons and checks if at least one of them is checked. If at least one is checked, it returns true; otherwise, it returns false.

The checkDropdown Function:

- This function is used to validate a dropdown/select element to ensure a valid option is selected.
- It takes a dropdown/select element as a parameter and checks if its selected index is greater than zero (indicating a valid selection).
- If a valid option is selected, the function returns true; otherwise, it returns false.

The checkCheckbox Function:

- This function is used to validate a checkbox input to ensure it is checked (selected).
- It takes a checkbox input element as a parameter and checks if it is checked.

- If the checkbox is checked, the function returns true; otherwise, it returns false.

These functions can be used individually or collectively within the validate Form function to perform various form validations. By combining them, you can ensure that user input adheres to the desired format and requirements before allowing form submission.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Validation Example</title>
</head>
<body>
  <h1>Form Validation Example</h1>
  <form onsubmit="return validateForm()">
    <label for="email">Email Address:</label>
    <input type="email" id="email" name="email" required>
    <span id="emailError" style="color: red;"></span><br>
    <label>Gender:</label>
    <input type="radio" id="male" name="gender" value="male">
    <label for="male">Male</label>
    <input type="radio" id="female" name="gender" value="female">
    <label for="female">Female</label>
    <input type="radio" id="others" name="gender" value="others">
    <label for="others">Others</label>
    <span id="genderError" style="color: red;"></span><br>
    <label for="country">Country:</label>
    <select name="country" id="country" required>
      <option value="">Select a country</option>
      <option value="india">India</option>
      <option value="usa">USA</option>
    </select>
    <span id="countryError"></span><br>
    <input type="checkbox" id="subscribe" name="subscribe" required>
    <label for="subscribe">Subscribe to newsletter</label>
    <span id="subscribeError"></span><br>
    <button type="submit">Submit</button>
  </form>
  <script>
    function validateForm() {
      // Reset Error Messages
      document.getElementById("emailError").textContent = ""
      document.getElementById("genderError").textContent = ""
      document.getElementById("countryError").textContent = ""
      document.getElementById("subscribeError").textContent = ""
      let isValid = true
      // Email Address Validation
      const emailInput = document.getElementById("email")
      const emailError = document.getElementById("emailError")
      const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/
      if (!emailPattern.test(emailInput.value)) {
        emailError.textContent = "Invalid email format"
        isValid = false
      }
      // Radio Button (Gender) validation
      const genderInputs = document.querySelectorAll("input[name='gender']")
      const genderError = document.getElementById("genderError")
      let isGenderSelected = false
      genderInputs.forEach(input => {
        if (input.checked) {
          isGenderSelected = true
        }
      })
      if (!isGenderSelected) {
        genderError.textContent = "Please select a gender"
        isValid = false
      }
      // Dropdown (Country) Validation
      const countrySelect = document.getElementById("country")
      const countryError = document.getElementById("countryError")
      if (countrySelect.selectedIndex === 0) {
        countryError.textContent = "Please select a country"
        isValid = false
      }
      // Checkbox (Subscribe) Validation
      const subscribeCheckbox = document.getElementById("subscribe")
      const subscribeError = document.getElementById("subscribeError")
      if (!subscribeCheckbox.checked) {
        subscribeError.textContent = "You must subscribe to continue"
        isValid = false
      }
      return isValid
    }
  </script>
</body>
</html>

```

Fig. 4.9: Form Validation Example

Output-

Form Validation Example

Email Address:

Gender: Male Female Others

Country:

Subscribe to newsletter

Fig. 4.10: Output for Form Validation Example

In this example:

- We have a form with email input, radio buttons for gender, a country dropdown, and a subscribe checkbox.
- Each input element has a corresponding error message span (e.g., emailError, genderError) to display validation errors.
- The validateForm function is called when the form is submitted.
- It validates the email format, checks if a gender option is selected, ensures a country is selected, and requires the checkbox to be checked.
- If any validation fails, the corresponding error message is displayed in red text.
- The form submission is prevented if any validation fails (return false in validateForm).

This example demonstrates how to use JavaScript to perform various form validations, providing feedback to users when they enter invalid data.

SUMMARY

- JavaScript offers a wide array of string manipulation techniques, including concatenation, substring extraction, and case changes.
- JavaScript is used to create, validate, and manipulate HTML forms.
- Functions like validateForm(), checkEmail(), checkRadio(), checkDropdown(), and checkCheckbox are used for form validation.
- Form validation ensures user input adheres to desired formats and requirements before submission.
- Handling dates and time is made easy with the Date object, offering features like formatting, comparison, and time zone adjustments.
- Dates and time in JavaScript are managed using the Date object.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. What is string manipulation in JavaScript? (a) Changing the color of strings (b) Performing operations on strings (c) Creating new strings (d) Removing strings from the document
2. Which operator is used for concatenating strings in JavaScript? (a) + (b) * (c) / (d) -
3. How do you find the length of a string in JavaScript? (a) str.len() (b) str.size() (c) str.length (d) str.count()
4. What does the substring method in JavaScript do? (a) Split a string into an array (b) Extract a portion of a string (c) Convert a string to lowercase (d) Replace substrings in a string
5. How can you change a string to uppercase in JavaScript? (a) str.toUpperCaseCase() (b) str.toUpperCase() (c) str.upper() (d) str.toUpperCase()
6. Which method is used to find the position of a substring within a string? (a) str.find() (b) str.index() (c) str.indexOf() (d) str.position()
7. What does the replace method in JavaScript do? (a) Removes all characters from a string (b) Replaces occurrences of a substring with another string (c) Splits a string into an array (d) Changes the case of the string
8. How do you split a string into an array using a delimiter in JavaScript? (a) str.break(delimiter) (b) str.split(delimiter) (c) str.divide(delimiter) (d) str.separate(delimiter)
9. What is the purpose of the charAt method in JavaScript? (a) Concatenate two strings (b) Retrieve a character at a specific index in a string (c) Convert a string to lowercase (d) Find the length of a string
10. The JavaScript method split() returns an array of substrings based on what? (a) The first character of the string (b) The number of occurrences of the delimiter (c) The position of the substring (d) A specified delimiter

B. Fill in the blanks

1. Concatenation is the process of _____ two or more strings.
2. The method used to find the length of a string is _____.
3. To change a string to lowercase, you can use the _____ method.
4. The JavaScript method for finding the position of a substring within a string is _____.
5. The JavaScript method for extracting a portion of a string is _____.
6. The validateForm function is typically used as a form's onsubmit handler for _____.
7. The JavaScript method used for splitting a string into an array of substrings is _____.
8. You can create a Date object to represent a specific date and time using new _____.

9. The JavaScript function used to validate the format of a date string is _____.
10. The JavaScript method for retrieving the character at a specific index within a string is _____.

C. True or False

1. String manipulation involves changing the gender of a string.
2. The replace method is used for finding the position of a substring within a string.
3. JavaScript does not have methods for changing the case of a string.
4. The concatenation method in JavaScript is used to split a string into an array.
5. The join method concatenates all elements of an array into a string.
6. A Date object in JavaScript can be used to represent a specific date and time.
7. Timestamps represent the number of milliseconds since January 1, 1970 (UTC).
8. The .join() method in JavaScript takes a delimiter as its parameter.
9. The .substring() method changes the original string.
10. The validateForm function is used to create form elements in JavaScript.

D. Short Question Answers

1. What is string concatenation, and how is it performed in JavaScript?
2. How can you find the length of a string in JavaScript?
3. Explain the purpose of the indexOf() method in JavaScript.
4. How do you change the case of a string to uppercase and lowercase in JavaScript?
5. What does the .split() method do, and how is it used with strings?
6. What is a Date object in JavaScript, and how can you create one?
7. What are timestamps, and how are they represented in JavaScript?
8. How do you check if a value is NaN in JavaScript?
9. What is the purpose of the .join() method in JavaScript, and how is it used?
10. How do you validate a user-entered date to ensure it's in a valid format?

Session 5. Manipulate Images using Javascript

Suresh, an explorer at heart, learned about "Manipulating Images and Geolocation in JavaScript," and it was like opening a treasure chest of adventures. With image manipulation, he could resize pictures and add cool filters, making them like magical artworks. Using geolocation, it was like having a treasure map, helping him find locations, plan adventures, and discover interesting spots. With these tools, he created amazing picture galleries and planned fantastic trips with friends, becoming a digital explorer and artist, just like finding hidden treasures and taking beautiful photos. It made the internet a more colorful and exciting place to explore! As shown in Figure 5.1.



Fig.5.1: Suresh exploring Manipulating Images and Geolocation in JavaScript

In this chapter, you will learn about Manipulate image, Creating New Images with JavaScript, Image Rollover Effect, and Timers and Images.

5.1 Manipulate image

Image manipulation in JavaScript can be achieved by using the HTML5 <canvas> element and its associated 2D drawing context. With this approach, you can load, draw, modify, and save images on a web page. Here are some common image manipulation tasks and how to perform them using JavaScript and the <canvas> element:

Loading an Image: To manipulate an image, you first need to load it. You can create an Image object and set its src attribute to the image file's URL. After the image is loaded, you can perform various operations on it.

```
const img = new Image();  
img.src = 'image.jpg';  
img.onload = function () {  
    // The image is loaded and can be manipulated here  
};
```

Drawing an Image on Canvas: To display the loaded image on a canvas, you can use the drawImage method of the 2D drawing context. This allows you to specify the image, position, and size on the canvas.

```
const canvas = document.getElementById('myCanvas');
```

```
const ctx = canvas.getContext("2d");
ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
```

Modifying Image Pixels: You can access and modify individual pixels of an image using the `getImageData` and `putImageData` methods. This allows you to apply various filters and effects to the image.

```
const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
const data = imageData.data; // Array containing pixel data (RGBA values)
```

```
for (let i = 0; i < data.length; i += 4) {
  // Manipulate pixel data here (e.g., change colors)
  data[i] = 255 - data[i]; // Invert the red channel
}
ctx.putImageData(imageData, 0, 0);
```

Resizing an Image: You can resize an image by drawing it on a canvas of a different size. You'll need to create a new canvas with the desired dimensions and draw the image onto it.

```
const newCanvas = document.createElement('canvas');
const newCtx = newCanvas.getContext("2d");
newCanvas.width = newWidth;
newCanvas.height = newHeight;
newCtx.drawImage(img, 0, 0, newWidth, newHeight);
```

Saving the Modified Image: To save the modified image, you can convert the canvas content to a data URL using the `toDataURL` method. You can then create a link that allows users to download the image.

```
const modifiedImageUrl = canvas.toDataURL('image/jpeg'); // or 'image/png'
const downloadLink = document.createElement('a');
downloadLink.href = modifiedImageUrl;
downloadLink.download = 'modified_image.jpg'; // Specify the desired file name
downloadLink.click();
```

Applying Filters and Effects: You can create custom filters and effects by manipulating pixel data. For example, you can implement grayscale, blur, sepia, or any other image filter by adjusting pixel values in the image data array.

Image manipulation in JavaScript can be quite powerful, allowing you to create interactive applications and perform real-time modifications to images displayed on a web page.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Manupulation Example</title>
</head>
<body>
  <h1>Image Manupulation Using JavaScript</h1>

  <input type="file" id="imageInput" accept="image*">
  <canvas id="canvas" width="400" height="300"></canvas>
  <button id="invertButton">Invert Colors</button>
  <a id="downloadLink" style="display: none;">Download Modified Image</a>

  <script>
    const canvas = document.getElementById("canvas")
    const ctx = canvas.getContext("2d")
    const imageInput = document.getElementById("imageInput")
    const invertButton = document.getElementById("invertButton")
    const downloadLink = document.getElementById("downloadLink")
    let img = new Image()
    // Event Listener for Image Input
    imageInput.addEventListener("change", function(event) {
      const file = event.target.files[0]
      if (file) {
        const reader = new FileReader()
        reader.onload = function(e) {
          img.src = e.target.result
        }
        reader.readAsDataURL(file)
      }
    })
    // Event Listener for image load
    img.onload = function () {
      canvas.width = img.width
      canvas.height = img.height
      ctx.drawImage(img, 0, 0, canvas.width, canvas.height)
    }
    // Event Listener for Invert Button
    invertButton.addEventListener("click", function() {
      const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height)
      const data = imageData.data
      // Invert colors by subtracting each color channel from 255
      for (let i = 0; i < data.length; i+=4) {
        data[i] = 255 - data[i] // Red
        data[i+1] = 255 - data[i+1] // Red
        data[i+2] = 255 - data[i+2] // Red
        // Alpha (data[i+3]) remains unchanged
      }
      ctx.putImageData(imageData, 0, 0)
      // Set the download link attributes
      downloadLink.href = canvas.toDataURL("image/jpeg")
      downloadLink.download = "modified_image.jpg"
      downloadLink.style.display = "block"
    })
  </script>
</body>
</html>

```

Fig. 5.2: using JavaScript and the <canvas> element

Output

After loading page as shown in Figure 5.3.

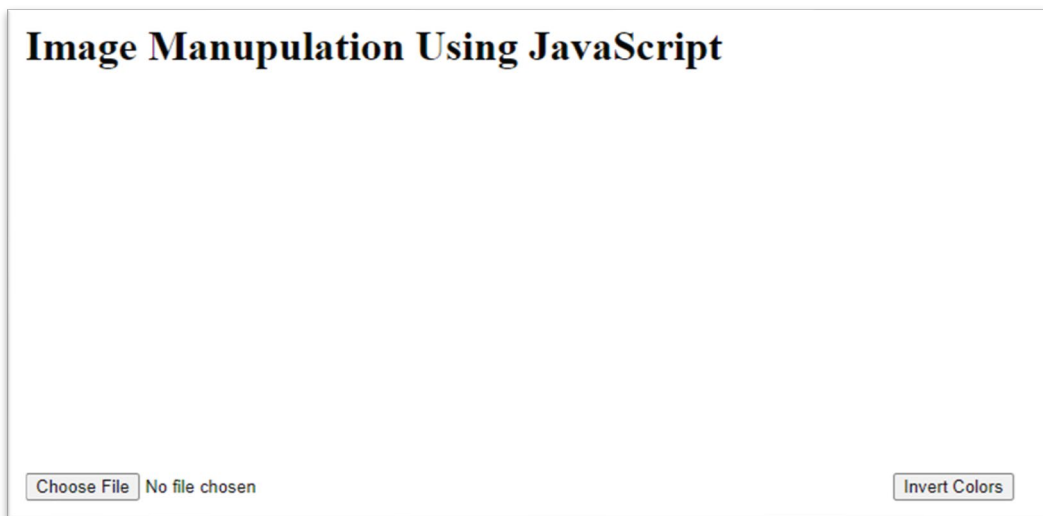


Fig. 5.3: After loading Image Manipulation using Java Script

After image select as shown in Figure 5.4.



Fig. 5.4: After Image Manipulation using Java Script

After invert as shown in Figure 5.5.



Fig. 5.5: After invert Manipulation using Java Script

Now we can download on click download button.

Code Explanation: In the modified code provided, we have an HTML webpage that allows users to upload an image, apply a color inversion filter to it, and then download the modified image with a specified name. Let's break down how each part of the code works:

HTML Structure:

- We have an `<input>` element of type "file" (imageInput) that allows users to select an image file from their device.
- A `<canvas>` element (canvas) is used to display and manipulate the image.
- A "Invert Colors" button (invertButton) triggers the color inversion filter.
- An `<a>` element (downloadLink) is initially hidden and becomes visible when the image is modified. It's used to download the modified image.

JavaScript Functionality:

- We use JavaScript to interact with the HTML elements and manipulate the image.
- When a user selects an image using the file input, an event listener (imageInput.addEventListener) is triggered. It reads the selected image file, loads it into an Image object (img), and displays the image on the canvas when it's loaded (img.onload).
- When the "Invert Colors" button is clicked (invertButton.addEventListener), it triggers an event handler that performs the following steps:
 - Retrieves the pixel data of the canvas using `ctx.getImageData`.
 - Iterates through the pixel data and inverts the colors by subtracting each color channel (red, green, blue) from 255.
 - Puts the modified pixel data back on the canvas using `ctx.putImageData`.
 - Sets the href and download attributes of the download link (downloadLink) to specify the data URL and the desired file name ("modified_image.jpg").
 - Makes the download link visible (`downloadLink.style.display = 'block'`).

Downloading the Modified Image:

- When the download link becomes visible, users can click on it to download the modified image. The download attribute of the link specifies the file name, and the href attribute contains the modified image's data URL.
- Color Inversion Filter:
 - The color inversion filter is applied by looping through the image's pixel data and subtracting each color channel's value from 255. This process inverts the colors, effectively creating a negative image.

Overall, this code demonstrates a simple image manipulation technique where users can upload an image, apply a color inversion filter to it, and then download the modified image with a specified name. The `<canvas>` element is used to draw and manipulate the image, and JavaScript handles the interaction and image processing.

Assignment 5.1.

- Write down the various image manipulation tasks perform them using JavaScript and the `<canvas>` element.
 - Loading an Image
 - Drawing an Image on Canvas
 - Modifying Image Pixels

5.2 Creating New Images with JavaScript

This concept involves using JavaScript to dynamically generate and display images on a webpage. You can create images on-the-fly by specifying their source (URL) and other attributes using JavaScript. This is often used when you need to load images dynamically based on user interactions or data from a server.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Image Creation with JavaScript</title>
</head>
<body>
<h1>Image creation with JavaScript</h1>

<!-- Create a placeholder 'div' element with the 'box' ID -->
<div id="box"></div>

<!-- Embed the JavaScript code within a <script> element -->
<script>
    // Create a new image element
    const image = document.createElement('img');

    // Set the image source (replace with your image URL)
    image.setAttribute('src', 'background.jpg');

    // Set the alternative text for the image
    image.setAttribute('alt', 'music');

    // Set the height and width of the image (in pixels)
    image.setAttribute('height', 350);
    image.setAttribute('width', 550);

    // Apply styling to the image (red border)
    image.style.border = '5px solid red';

    // Handle image load error
    image.onerror = function handleError() {
        console.log('Image could not be loaded');
        // Optionally, you can set a backup image or hide the image here
        // image.src = 'backup-image.png';
        // image.style.display = 'none';
    };
</script>
</body>
</html>
```



```

};

// Handle image loaded successfully
image.onload = function handleImageLoaded() {
console.log('Image loaded successfully');
};

// Get the element with the ID 'box' and append the image to it
const box = document.getElementById('box');
box.appendChild(image);
</script>
</body>
</html>

```

Output



Fig. 5.6: Output for Image Creation with Java Script

Code Explanation:

1. `const image = document.createElement('img');` This line creates a new `img` element and assigns it to the `image` constant.
2. `image.setAttribute('src', 'background.jpg');` It sets the `src` attribute of the `img` element to `'background.jpg'`. Make sure `'background.jpg'` is the correct path to your image file.
3. `image.setAttribute('alt', 'music');` This line sets the alternative text for the image to `'music'`.
4. `image.setAttribute('height', 350);` and `image.setAttribute('width', 550);` These lines set the height and width attributes of the image to 350 pixels and 550 pixels, respectively.
5. `image.style.border = '5px solid red';` It applies a CSS style to the image, giving it a red border with a thickness of 5 pixels.
6. `image.onerror = function handleError() { ... };` This code sets an event handler to handle image load errors. If the image fails to load, it logs an error message to the console.
7. `image.onload = function handleImageLoaded() { ... };` Here, an event handler is set to handle the successful loading of the image. It logs a success message to the console.

8. `const box = document.getElementById('box');`; This line retrieves an HTML element with the ID 'box' and assigns it to the box constant.
9. `box.appendChild(image);`; Finally, it appends the image element to the box element in the HTML document, making the image visible on the webpage.

5.3 Image Rollover Effect

An image rollover effect is a common interaction on websites where the appearance of an image changes when a user hovers their mouse pointer over it. JavaScript and CSS are typically used to achieve this effect. When the mouse hovers over an image, it can change its source or apply visual effects like changing colors or adding borders.

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Image Rollover Effect</title>
</head>
<body>
<h1>Image Rollover Effect</h1>

<!-- Create an <img> element with the initial image -->

<script>
    // JavaScript function to change the image source on hover
    function changeImage() {
        const imageElement = document.getElementById('rolloverImage');
        imageElement.src = 'modified_image.jpg'; // Replace 'hover-image.jpg' with the path to
your hover image
    }

    // JavaScript function to restore the original image on mouseout
    function restoreImage() {
        const imageElement = document.getElementById('rolloverImage');
        imageElement.src = 'initial-image.jpg'; // Replace 'initial-image.jpg' with the path to your
initial image
    }
</script>
</body>
</html>
```

Output

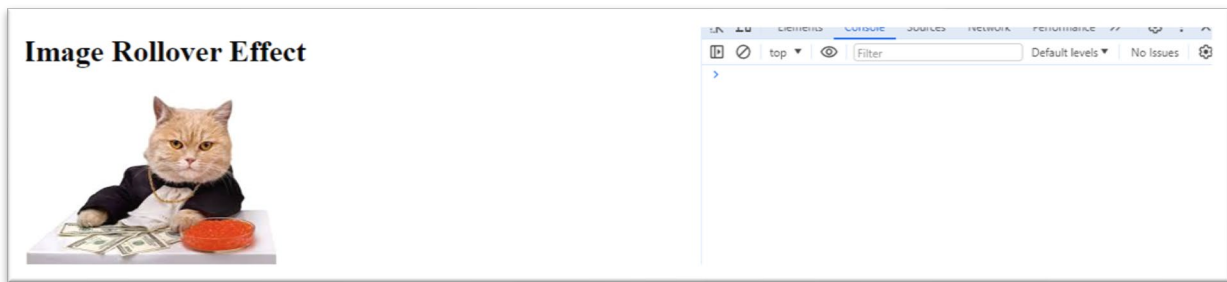


Fig. 5.7: Output for Image Rollover Effect

After hover as shown in Figure 5.8.

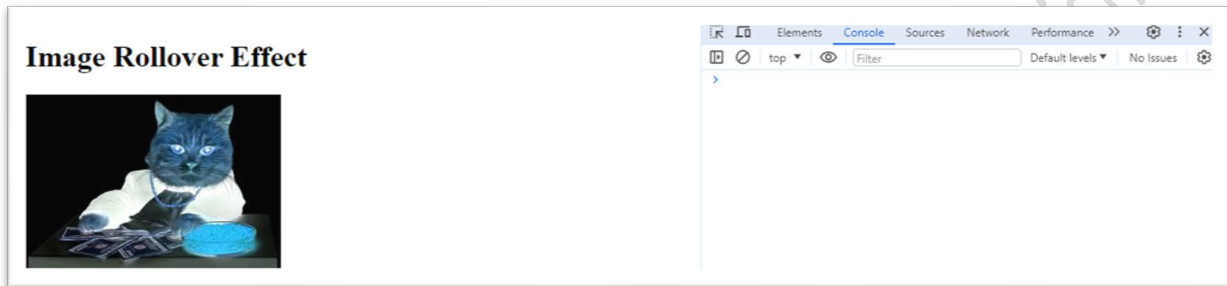


Fig. 5.8: Output for After hover Image Rollover Effect

Code Explanation

- `<!DOCTYPE html>`: This line declares the document type and version of HTML being used.
- `<html lang="en">`: The opening tag for the HTML document, specifying the document's language as English.
- `<head>`: This section contains metadata about the document, such as the character encoding and page title.
- `<meta charset="UTF-8">`: Specifies the character encoding for the document as UTF-8, which supports a wide range of characters.
- `<title>Image Rollover Effect</title>`: Sets the title of the web page, which appears in the browser's title bar or tab.
- `<body>`: The main content of the web page is contained within this tag.
- `<h1>Image Rollover Effect</h1>`: A heading element that displays "Image Rollover Effect" as the page's main heading.
- ``: This line creates an `img` element with the following attributes:
 - `id="rolloverImage"`: Assigns the ID "rolloverImage" to the image element, which is used to identify the element in JavaScript.
 - `src="initial-image.jpg"`: Sets the initial source of the image to "initial-image.jpg." This is the image displayed initially.
 - `alt="Initial Image"`: Specifies alternative text for the image, which is displayed if the image cannot be loaded or for accessibility purposes.
 - `onmouseover="changeImage();"`: Specifies that the JavaScript function `changeImage` should be called when the mouse pointer is over the image.

- `onmouseout="restoreImage()";`: Specifies that the JavaScript function `restoreImage` should be called when the mouse pointer leaves the image.
- `<script>`: Begins a JavaScript script section within the HTML document.
- `function changeImage() { ... }`: Defines a JavaScript function named `changeImage`. This function will be called when the mouse hovers over the image.
- `const imageElement = document.getElementById('rolloverImage');`: Retrieves the image element with the ID "rolloverImage" and assigns it to the `imageElement` constant, allowing us to manipulate the image in JavaScript.
- `imageElement.src = 'hover-image.jpg';`: Changes the `src` attribute of the `imageElement` to 'hover-image.jpg' when the `changeImage` function is called. This swaps the image to the "hover" state.
- `function restoreImage() { ... }`: Defines a JavaScript function named `restoreImage`. This function will be called when the mouse moves away from the image.
- `imageElement.src = 'initial-image.jpg';`: Restores the `src` attribute of the `imageElement` to 'initial-image.jpg' when the `restoreImage` function is called, reverting the image to its initial state.
- `</script>`: Ends the JavaScript script section.

This code creates a basic image rollover effect, allowing the image to change when the mouse hovers over it and returns to its original state when the mouse moves away.

5.4 Timers and Images

Timers in JavaScript allow you to execute code at specified intervals. When it comes to images, timers can be used to create various effects. For example, you can create a slideshow that automatically changes images after a certain time interval or implement animations by updating the image's position or properties over time as shown in Figure 5.9.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Timers and Image Example</title>
</head>
<body>
  <h1>Timers and Image Example</h1>
  <!-- Create an <img> element with an initial image -->
  
  <script>
    // Get the image element by ID
    const imageElement = document.getElementById('timerImage');
    // Array of image URLs to cycle through
    const imageUrls = ['modified-image.jpg', 'sample-image.jpg', 'pause.jpg', 'play.jpg'];
    let currentIndex = 0; // Index of the currently displayed image
    // Function to change the image
    function changeImage() {
      // Change the image source to the next URL in the array
      imageElement.src = imageUrls[currentIndex];
      // Increment the index and wrap around if needed
      currentIndex = (currentIndex + 1) % imageUrls.length;
      // Set a timer to call the function again after 3 seconds (3000 milliseconds)
      setTimeout(changeImage, 3000);
    }
    // Start the timer to change the image initially
    setTimeout(changeImage, 3000);
  </script>
</body>
</html>

```

Fig. 5.9: Timers in JavaScript

Output

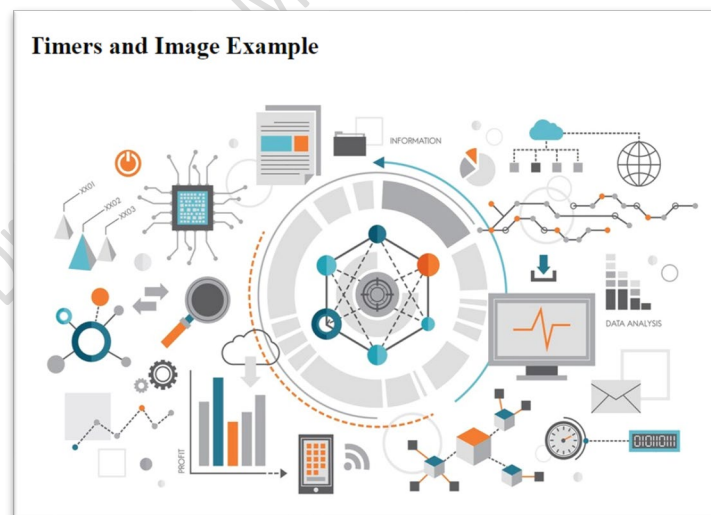


Fig. 5.10: Output for Timers in JavaScript

After 3 second as shown in Figure 5.11.

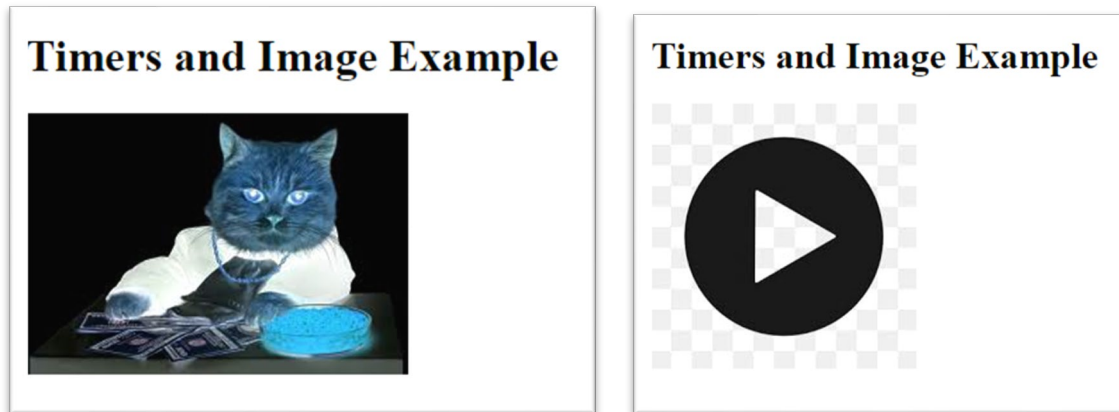


Fig. 5.11: Output for After 3 second Timers in JavaScript

It continues.

Code Explanation:

HTML Structure: We start with the usual HTML5 document structure, including the `<!DOCTYPE>` declaration, the opening `<html>` tag with the language attribute, and the `<head>` and `<body>` sections.

Page Title and Heading: Inside the `<head>` section, we set the document's character encoding and provide a title for the page.

In the `<body>` section, there's an `<h1>` element with the text "Timers and Image Example," serving as the page heading.

Image Element: We create an `` element with the ID "timerImage."

The `src` attribute is initially set to "background.jpg," which is the image displayed when the page loads. The `alt` attribute provides alternative text for the image, useful for accessibility.

JavaScript Code: The JavaScript code is embedded within a `<script>` element, and it starts by getting a reference to the image element with the ID "timerImage" using `document.getElementById('timerImage')`.

An array called `imageUrls` is defined. It contains the URLs of the images that we want to cycle through in the slideshow. In this example, there are four images: "modified_image.jpg," "datascience.jpg," "pause.jpg," and "play.jpg."

`currentIndex` is a variable that keeps track of the currently displayed image's index within the `imageUrls` array. It is initialized to 0, representing the first image in the array.

The `changeImage` function is defined. This function is responsible for changing the displayed image and setting up the timer for the next image.

Inside changeImage: The `src` attribute of the image element (`imageElement`) is set to the URL of the next image in the `imageUrls` array. This effectively changes the displayed image.

`currentIndex` is incremented, and the modulo operation (%) is used to ensure that it wraps around to 0 when it reaches the end of the array, allowing the slideshow to cycle through all images.

`setTimeout` is used to schedule the next call to `changeImage` after a 3-second delay (3000 milliseconds). This creates an automatic image slideshow effect with a 3-second interval between image changes.

Finally, we initiate the timer by calling `setTimeout(changeImage, 3000)` outside the `changeImage` function. This starts the image slideshow when the page initially loads.

The result is an automatic image slideshow that cycles through the specified images, changing the displayed image every 3 seconds. This example demonstrates the use of timers to create dynamic content on a webpage.

5.5 Manipulate HTML Elements with JavaScript

Manipulating HTML elements with JavaScript involves using JavaScript to dynamically change the content, style, and structure of HTML elements on a web page. This is commonly done to create interactive and responsive web applications. Here are some common tasks and techniques for manipulating HTML elements with JavaScript:

Accessing Elements: You can access HTML elements in JavaScript using various methods such as `getElementById`, `getElementsByClassName`, `getElementsByTagName`, and `querySelector`. These methods return references to DOM (Document Object Model) elements.

```
const elementById = document.getElementById('myId');
const elementsByClass = document.getElementsByClassName('myClass');
const elementsByTag = document.getElementsByTagName('div');
const elementByQuery = document.querySelector('.mySelector');
```

Modifying Content: You can change the content of elements using the `innerHTML` property or the `textContent` property. The `innerHTML` property allows you to set HTML content, while `textContent` sets plain text content.

```
const element = document.getElementById('myElement');
element.innerHTML = '<strong>New Content</strong>';
element.textContent = 'New Text Content';
```

Changing Styles: You can manipulate the style of elements by accessing their `style` property. You can change properties like `backgroundColor`, `fontSize`, `color`, and more.

```
const element = document.getElementById('myElement');
element.style.backgroundColor = 'blue';
element.style.fontSize = '16px';
```

Adding/Removing Classes: You can add or remove CSS classes to/from elements using the `classList` property. This is useful for applying predefined styles or toggling classes for animations.

```
const element = document.getElementById('myElement');
element.classList.add('newClass');
element.classList.remove('oldClass');
```

Creating and Appending Elements: You can create new HTML elements and append them to the DOM using the `createElement` and `appendChild` methods.

```
const newElement = document.createElement('div');
newElement.textContent = 'New Element';
document.body.appendChild(newElement);
```

Event Handling: You can attach event listeners to elements to respond to user interactions (e.g., click, hover). This allows you to create interactive features.

```
const button = document.getElementById('myButton');
button.addEventListener('click', function () {
```

```
alert('Button Clicked!');
});
```

Removing Elements: You can remove elements from the DOM using the `removeChild` method.

```
const elementToRemove = document.getElementById('toBeRemoved');
elementToRemove.parentNode.removeChild(elementToRemove);
```

Data Attributes: You can store custom data in HTML elements using `data-*` attributes, which can be accessed and modified using JavaScript.

```
<div id="myElement" data-info="some data"></div>
const element = document.getElementById('myElement');
const data = element.getAttribute('data-info');
```

Example-

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>HTML Element Manipulation</title>
<style>
.highlighted {
    background-color: yellow;
}
</style>
</head>
<body>
<h1 id="myHeading">HTML Element Manipulation</h1>

<!-- Accessing Elements -->
<p>Accessing Elements:</p>
<ul>
<li id="accessExample">Access me!</li>
<li class="accessExample">Access me too!</li>
</ul>

<!-- Modifying Content -->
<p>Modifying Content:</p>
<div id="contentExample">Initial Content</div>
<div id="innerHtmlExample">InnerHTML Example</div>

<!-- Changing Styles -->
<p>Changing Styles:</p>
<div id="styleExample">Style me</div>
```



```

<!-- Adding/Removing Classes -->
<p>Adding/Removing Classes:</p>
<div id="classExample">Class Example</div>

<!-- Creating and Appending Elements -->
<p>Creating and Appending Elements:</p>
<button id="createElementBtn">Create Element</button>
<div id="container"></div>

<!-- Event Handling -->
<p>Event Handling:</p>
<button id="clickMeBtn">Click Me</button>

<!-- Removing Elements -->
<p>Removing Elements:</p>
<div id="removeExample">Remove me</div>
<!-- Data Attributes -->
<p>Data Attributes:</p>
<div id="dataExample" data-info="Custom Data Attribute">Data Example</div>
<script>
  // Accessing Elements
  const elementById = document.getElementById('accessExample');
  const elementsByClass = document.getElementsByClassName('accessExample');

  // Modifying Content
  const contentElement = document.getElementById('contentExample');
  contentElement.innerHTML = '<strong>New Content</strong>';

  // InnerHTML Example
  const innerHtmlExample = document.getElementById('innerHtmlExample');
  innerHtmlExample.innerHTML = '<em>Modified with InnerHTML</em>';

  // Changing Styles
  const styleElement = document.getElementById('styleExample');
  styleElement.style.backgroundColor = 'blue';
  styleElement.style.fontSize = '20px';

  // Adding/Removing Classes
  const classElement = document.getElementById('classExample');

```

```
classElement.classList.add('highlighted');
classElement.classList.remove('highlighted');

// Creating and Appending Elements
const createElementBtn = document.getElementById('createElementBtn');
const container = document.getElementById('container');
createElementBtn.addEventListener('click', function () {
    const newElement = document.createElement('p');
    newElement.textContent = 'Newly Created Element';
container.appendChild(newElement);
});

// Event Handling
const clickMeBtn = document.getElementById('clickMeBtn');
clickMeBtn.addEventListener('click', function () {
alert('Button Clicked!');
});

// Removing Elements
const removeExample = document.getElementById('removeExample');
removeExample.parentNode.removeChild(removeExample);

// Data Attributes
const dataExample = document.getElementById('dataExample');
const dataInfo = dataExample.getAttribute('data-info');
console.log('Data Attribute:', dataInfo);
</script>
</body>
</html>
```

Output

HTML Element Manipulation

Accessing Elements:

- Access me!
- Access me too!

Modifying Content:

New Content

Modified with InnerHTML

Changing Styles:

Style me

Adding/Removing Classes:

Class Example

Creating and Appending Elements:

Create Element

Newly Created Element

Event Handling:

Click Me

Removing Elements:

Data Attributes:

Data Example

Fig. 5.12: HTML Element Manipulation

Explanation

HTML Structure: The HTML file sets up a basic structure with various sections to demonstrate HTML element manipulation using JavaScript. It includes headings, lists, buttons, and div elements with unique IDs and classes.

Styles: The code includes a simple CSS style to highlight elements with a yellow background when they have the class "highlighted."

JavaScript: The JavaScript code is included within a <script> tag. It performs various operations to demonstrate how to manipulate HTML elements.

- **Accessing Elements:** It starts by accessing HTML elements using `getElementById` and `getElementsByClassName` methods. These elements are stored in variables like `elementById` and `elementsByClass` for further manipulation.
- **Modifying Content:** It demonstrates how to modify the content of elements. For instance, it changes the content of a div with the ID `contentExample` to "New Content" and another div with the ID `innerHTMLExample` using the `innerHTML` property.
- **Changing Styles:** It showcases changing the styles of elements. It sets the background color and font size of a div with the ID `styleExample` using the `style` property.
- **Adding/Removing Classes:** The code adds and removes classes from the `classElement` using the `classList` property. It adds the "highlighted" class and then removes it.
- **Element Creation and Addition:** This section illustrates the dynamic generation and addition of elements. Upon clicking the button identified as `createElementBtn`, a fresh `p` element containing the text "Newly Created Element" is generated and incorporated into a container div marked with the ID `container`.
- **Managing Events:** Event management is exemplified through the attachment of a click event listener to the button having the ID `clickMeBtn`. When this button is activated, it triggers an alert displaying the message "Button Clicked!"
- **Removing Elements:** The code removes the div with the ID `removeExample` by using the `removeChild` method on its parent.
- **Data Attributes:** It accesses a data attribute of the `dataExample` div using the `getAttribute` method and logs it to the console.

Execution: When the HTML page is loaded, the JavaScript code runs, and various HTML element manipulations and event handling take place as described above.

Overall, this code serves as a practical example of how to interact with and manipulate HTML elements using JavaScript.

Assignment 5.2.

- Write down the code for accessing HTML elements in JavaScript.
- Write down the code for add or remove CSS classes from elements using the classList property.
- Write down the code for remove elements from the DOM using the removeChild method.

SUMMARY

- HTML's <canvas> element and JavaScript's 2D drawing context enable image manipulation.
- Loading images involves creating an Image object and setting its src attribute.
- Images can be drawn on a canvas using the drawImage method.
- Modifying image pixels is achieved with getImageData and putImageData.
- Resizing images is possible by drawing them on a canvas of the desired dimensions.
- Saving modified images is done by converting canvas content to data URLs.
- Custom filters and effects can be created by manipulating pixel data.
- Geolocation in JavaScript involves checking support and requesting user location.
- Geolocation options like accuracy and timeout can be configured.
- Continuous location tracking is possible using watchPosition.
- Handling user permissions and browser security is crucial for geolocation.
- Google Maps integration requires an API key and initialization functions.
- JavaScript can dynamically access, modify, and style HTML elements.
- Event handling in JavaScript enables interactive web applications.
- Elements can be added, removed, and their data attributes accessed in HTML manipulation.

Check Your Progress

A. MULTIPLE CHOICE QUESTIONS

1. How can you load an image in JavaScript for manipulation? (a) Using the element (b) Using the document.createElement method (c) Using the <canvas> element (d) Using the navigator.geolocation object
2. Which method allows you to access and modify individual pixels of an image in JavaScript? (a) getImageData (b) drawImage (c) putImageData (d) createElement
3. To save a modified image in JavaScript, you can convert the canvas content to a data URL using: (a) getContext('2d') (b) toDataURL() (c) img.src (d) getImageData
4. What is one purpose of using geolocation in JavaScript? (a) To manipulate images (b) To create custom filters (c) To access a user's geographical position (d) To create HTML elements
5. When requesting a user's location with navigator.geolocation, which method is used to handle successful location retrieval? (a) onerror (b) getCurrentPosition (c) addEventListener (d) getElementsByClassName
6. Which JavaScript method allows you to change the style of an HTML element? (a) innerHTML (b) appendChild (c) style (d) textContent
7. What attribute is used to specify the source URL of an image element in HTML? (a) alt (b) src (c) class (d) href
8. To add a class to an HTML element in JavaScript, you can use: (a) classList.add() (b) style.backgroundColor (c) innerHTML (d) getElementById
9. What is the purpose of the <canvas> element in image manipulation in JavaScript? (a) To create an image rollover effect (b) To create data URLs (c) To load and modify images (d) To access geolocation data
10. Which attribute in the element specifies the alternative text for an image? (a) src (b) alt (c) title (d) class

B. Fill in the blanks

1. To manipulate an _____, first load it using an image object.
2. To display a loaded image on a canvas, you can use the _____ method of the 2D drawing context.
3. You can access and modify individual pixels of an image using the _____ and putImageData methods.
4. To save a modified image, you can convert the canvas content to a data URL using the _____ method.
5. _____ in JavaScript allows you to access a user's geographical position through their device's GPS, IP address, or other sources.
6. You can check if geolocation is supported in the user's _____ using the navigator.geolocation object in JavaScript.
7. The function that handles successful retrieval of a user's _____ is called getCurrentPosition.

8. To continuously _____ a user's location, you can use the watchPosition method.
9. In JavaScript, you can access and _____ an element's style properties using the style property.
10. You can remove an HTML element from the _____ using the removeChild method.

C. True or False

1. Image manipulation in JavaScript is primarily achieved using the HTML element.
2. To create a data URL for a canvas, you can use the getElementById method.
3. Geolocation in JavaScript allows you to access a user's geographical position through their device's GPS, IP address, or other location data sources.
4. The timeout property in geolocation options specifies the maximum age of location data.
5. To change the style of an HTML element in JavaScript, you can use the classList property.
6. You can remove an HTML element from the DOM using the createElement method.
7. When manipulating content using JavaScript, the textContent property allows you to set plain text content.
8. The watchPosition method in geolocation allows you to get a one-time location fix.
9. To change the content of an HTML element in JavaScript, you can use the appendChild method.
10. The alt attribute in the element specifies the source URL of an image.

D. Short Question Answers

1. What is the purpose of the <canvas> element in HTML?
2. How do you load an image in JavaScript for manipulation?
3. What method allows you to access individual pixels of an image in JavaScript?
4. Explain the purpose of the alt attribute in the element.
5. How can you add a class to an HTML element using JavaScript?
6. What does the watchPosition method in geolocation do?
7. Which property is used to change the content of an HTML element in JavaScript?
8. Describe the function of the timeout property in geolocation options.
9. How do you create a new HTML element in JavaScript?
10. What method can be used to save a modified image in JavaScript?

Session 6. HTML5 Canvas

Joe, a creative kid, discovered "HTML5 Canvas," a magical drawing board on the computer that allowed him to draw, paint, and create beautiful artwork right on the screen. With HTML5 Canvas, he could draw shapes, add colors, create animations, and even design simple games, becoming a digital artist. It was like a world of endless creativity on his computer, making the internet a more colorful and entertaining place to explore. As shown in figure 6.1.



Fig. 6.1: Joe using HTML5 Canvas

In this chapter, you will learn about HTML5 Canvas tag in JavaScript, Drawing Shapes, Gradients, and using Images with the Canvas Tag

6.1 HTML5 Canvas tag in JavaScript

The HTML5 <canvas> element allows you to draw graphics, animations, and interactive content directly in a web page using JavaScript.

Did You Know?

The <canvas> element is only a container for graphics.

Here's a step-by-step guide to use the <canvas> tag in JavaScript:

Create an HTML File: Start by creating an HTML file, for example, canvas.html, and open it in your favorite text editor.

Add the <canvas> Element: Inside the HTML file, add a <canvas> element where you want to draw your graphics. Provide it with an id attribute for easy JavaScript access and set the width and height attributes to define the size of the canvas as shown in Figure 6.2.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Canvas Example</title>
</head>
<body>
  <canvas id="myCanvas" width="400" height="200"></canvas>
</body>
</html>
```

Fig. 6.2: use the <canvas> tag in JavaScript

Access the Canvas in JavaScript: To work with the canvas in JavaScript, you need to obtain a reference to it using its id. Place the following code inside a `<script>` tag just before the closing `</body>` tag or in an external JavaScript file linked to your HTML.

```
const canvas = document.getElementById('myCanvas');  
const context = canvas.getContext('2d');
```

The `getContext('2d')` method is used to get a 2D rendering context, which provides methods for drawing on the canvas.

Drawing on the Canvas: Now that you have the canvas and its context, you can use various drawing methods provided by the context. For instance, you can draw a rectangle like this:

```
context.fillStyle = 'blue'; // Set fill color to blue  
context.fillRect(50, 50, 100, 80); // Draw a filled rectangle
```

This code sets the fill color to blue and draws a filled rectangle on the canvas.

Adding More Graphics: You can add more graphics, shapes, text, images, and even create animations by repeatedly using the context's drawing methods. Explore functions like `fillRect`, `strokeRect`, `fillText`, `drawImage`, etc., to create your desired visuals.

Interactivity: Make your canvas interactive by adding event listeners to respond to user actions. For example, you can use the `canvas.addEventListener` method to capture mouse clicks or key presses and update the canvas accordingly.

Run the HTML Page: Open your HTML file in a web browser to see your canvas and the graphics you've drawn on it. open `canvas.html`

Explore Advanced Features: The `<canvas>` element and 2D context provide a powerful toolset for creating graphics. You can explore more advanced topics like animations, transformations, gradients, and pixel manipulation as you become more familiar with canvas programming.

Remember that HTML5 `<canvas>` is suitable for 2D graphics. For more advanced 3D graphics, WebGL is a popular technology built on top of HTML5 that provides hardware-accelerated 3D rendering capabilities for web applications.

Assignment 6.1.

Write down a javascript code for the following:

- Add the `<canvas>` Element
- Access the Canvas in JavaScript
- Drawing on the Canvas

6.2 Drawing Shapes

6.2.1 Drawing Shapes – Rectangles:

- To draw a rectangle on an HTML5 canvas, you can use the `fillRect()` and `strokeRect()` methods of the 2D rendering context.
- `fillRect(x, y, width, height)`: Draws a filled rectangle starting at (x, y) with the specified width and height.
- `strokeRect(x, y, width, height)`: Draws the outline (stroke) of a rectangle with the specified parameters.

6.2.2 Drawing Shapes – Circles and Arcs:

- To draw circles and arcs, you can use the `arc()` method.
- `arc(x, y, radius, startAngle, endAngle, anticlockwise)`: Draws a circular arc with a center at (x, y) and the specified radius. The `startAngle` and `endAngle` define the start and end points

of the arc, and the anticlockwise parameter determines the direction (clockwise or anticlockwise) to draw the arc.

6.2.3 Drawing Paths and Lines:

- The canvas API allows you to create custom paths and draw lines using the `beginPath()`, `moveTo()`, `lineTo()`, and `closePath()` methods.
- `beginPath()`: Starts a new path.
- `moveTo(x, y)`: Moves the path drawing position to (x, y) without drawing a line.
- `lineTo(x, y)`: Draws a straight line from the current path position to (x, y).
- `closePath()`: Closes the path by connecting the current position to the starting position.
- You can use these methods to create complex shapes and paths by connecting multiple lines.

6.2.4 Line Caps and Line Joins:

- Line caps and joins are properties that affect the appearance of the ends of lines and the intersections of lines.
- **Line Caps:**
 - `context.lineCap`: This property can be set to "butt", "round", or "square". It defines how the ends of lines should appear.
 - "butt": The default value. Ends the line with no added decoration.
 - "round": Rounds off the ends of the line with a semicircular shape.
 - "square": Squares off the ends of the line with a square shape.
- **Line Joins:**
 - `context.lineJoin`: This property can be set to "round", "bevel", or "miter". It determines how lines that meet at an angle should be joined.
 - "round": Joins lines with a rounded corner.
 - "bevel": Joins lines with a beveled (flat) corner.
 - "miter": Joins lines with an extended point (miter) at the corner.
- `context.miterLimit`: This property sets the limit at which miter joins are cut off. It is only applicable when `lineJoin` is set to "miter".

These concepts are fundamental for creating various shapes and graphical elements on an HTML5 canvas. By combining these methods and properties, you can create intricate drawings and visual effects as shown in Figure 6.3 and 6.4.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Canvas Example</title>
</head>
<body>
  <canvas id="myCanvas" width="400" height="200"></canvas>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Canvas Drawing Example</title>
  <style>
    canvas {
      border: 1px solid ■ black;
    }
  </style>
</head>
<body>
</body>
</html>

```

Fig. 6.3: Drawing Shapes in HTML5

Assignment 6.2.

- Write down the name of method to draw a circle and arcs.
- Write down the name of method to create custom paths and draw lines.
- Write down the name of property set to "butt", "round", or "square" in Line Caps.

```

<body>
  <h1>Canvas Drawing Example</h1>
  <canvas id="myCanvas" width="400" height="300"></canvas>

  <script>
    const canvas = document.getElementById('myCanvas');
    const ctx = canvas.getContext('2d');

    // Drawing a filled rectangle
    ctx.fillStyle = 'blue';
    ctx.fillRect(20, 20, 100, 50);

    // Drawing the outline of a rectangle
    ctx.strokeStyle = 'red';
    ctx.lineWidth = 2;
    ctx.strokeRect(150, 20, 100, 50);

    // Drawing a circle (arc)
    ctx.beginPath();
    ctx.arc(75, 150, 50, 0, Math.PI * 2, false); // Full circle
    ctx.fillStyle = 'green';
    ctx.fill();

    // Drawing a path with lines
    ctx.beginPath();
    ctx.moveTo(200, 150); // Starting point
    ctx.lineTo(250, 100); // Line to point
    ctx.lineTo(300, 150); // Line to point
    ctx.closePath(); // Close the path
    ctx.strokeStyle = 'purple';
    ctx.lineWidth = 3;
    ctx.stroke();

    // Setting line caps and line joins
    ctx.lineWidth = 10;

    // Line with butt cap
    ctx.lineCap = 'butt';
    ctx.beginPath();
    ctx.moveTo(50, 250);
    ctx.lineTo(150, 250);
    ctx.stroke();

    // Line with round cap
    ctx.lineCap = 'round';
    ctx.beginPath();
    ctx.moveTo(200, 250);
    ctx.lineTo(300, 250);
    ctx.stroke();

    // Line with square cap
    ctx.lineCap = 'square';
    ctx.beginPath();
    ctx.moveTo(350, 250);
    ctx.lineTo(450, 250);
    ctx.stroke();

    // Line joins
    ctx.lineJoin = 'round';
    ctx.beginPath();
    ctx.moveTo(100, 300);
    ctx.lineTo(150, 350);
    ctx.lineTo(200, 300);
    ctx.stroke();

    ctx.lineJoin = 'bevel';
    ctx.beginPath();
    ctx.moveTo(250, 300);
    ctx.lineTo(300, 350);
    ctx.lineTo(350, 300);
    ctx.stroke();

    ctx.lineJoin = 'miter';
    ctx.miterLimit = 2; // Miter limit for sharp miter joins
    ctx.beginPath();
    ctx.moveTo(400, 300);
    ctx.lineTo(450, 350);
    ctx.lineTo(500, 300);
    ctx.stroke();
  </script>
</body>
</html>

```

Fig. 6.4: Create intricate drawings and visual effects in HTML5

Output

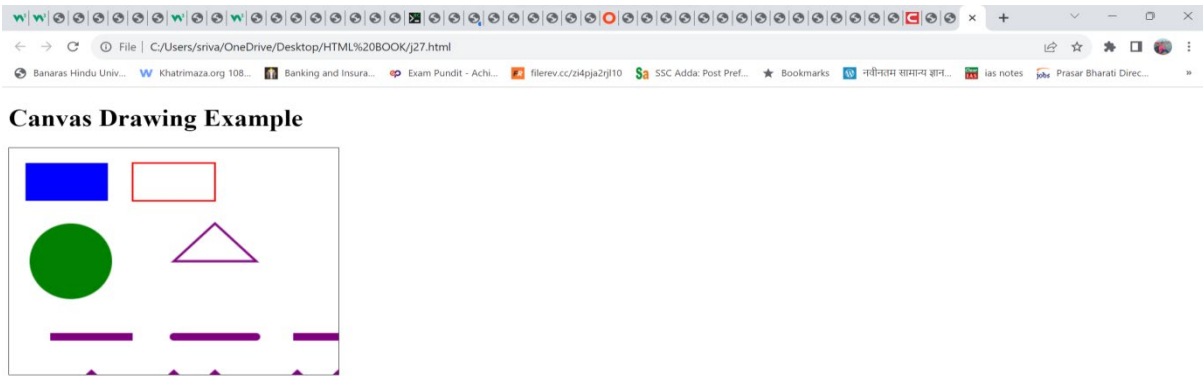


Fig. 6.5: Output for Canvas Drawing Example

Code Explanation-

Drawing a Filled Rectangle: The `fillRect` method is used to draw a filled blue rectangle at coordinates (20, 20) with a width of 100 pixels and a height of 50 pixels.

Drawing the Outline of a Rectangle: The `strokeRect` method is used to draw the outline (border) of a red rectangle at coordinates (150, 20) with the same width and height as the filled rectangle. The border has a line width of 2 pixels.

Drawing a Circle (Arc): The `arc` method is used to draw a green circle (arc) centered at (75, 150) with a radius of 50 pixels. It draws a full circle by specifying a start angle of 0 radians and an end angle of $\text{Math.PI} * 2$ radians (360 degrees). The `fill` method is used to fill the circle.

Drawing a Path with Lines: The `beginPath`, `moveTo`, `lineTo`, and `closePath` methods are used to create a path consisting of connected lines. It starts at (200, 150), moves to (250, 100), and then to (300, 150), creating a triangle. The `stroke` method outlines the path with purple color and a line width of 3 pixels.

Setting Line Caps and Line Joins:

- The code demonstrates different line cap styles (butt, round, and square) using the `lineCap` property. Three lines are drawn with different line caps.
- It also demonstrates different line join styles (round, bevel, and miter) using the `lineJoin` property. Three paths with sharp miter joins, beveled joins, and round joins are drawn. The `miterLimit` property controls the sharpness of miter joints.

You can see the results of these drawing operations on the HTML5 canvas element. Each section of the code is explained to show how it achieves different drawing effects on the canvas.

6.3 Gradients

Gradients in HTML5 canvas allow you to create smooth transitions of color or style within a shape or along a path. Gradients can be linear or radial, and they are often used to add depth and visual appeal to drawings. Here's an explanation of the two main types of gradients in canvas:

6.3.1 Linear Gradients

A linear gradient is a smooth transition of colors or styles along a straight line.

To create a linear gradient, you need to define at least two color stops (points with specific colors) and the start and end points of the gradient line.

You can use the `createLinearGradient` method to create a linear gradient object and then set the color stops and gradient direction.

Color stops are defined using the `addColorStop` method of the gradient object. Each color stop has a position (a value between 0 and 1) and a color value.

Once the gradient is defined, you can use it as a fill or stroke style when drawing shapes.

6.3.2 Radial Gradients:

A radial gradient is a smooth transition of colors or styles from the center of a circle outward in all directions.

To create a radial gradient, you need to define at least two color stops and the center and radius of the inner and outer circles.

Similar to linear gradients, you use the `createRadialGradient` method to create a radial gradient object and set the color stops and circles' positions and radii.

Radial gradients can be used to create effects like soft glows, shiny surfaces, or radial backgrounds as shown Figure 6.6.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Canvas Gradients Example</title>
</head>
<body>
  <h1>Canvas Gradients Example</h1>
  <!-- Create a canvas element -->
  <canvas id="myCanvas" width="400" height="200"></canvas>
  <script>
    // Get the canvas element and its 2D rendering context
    const canvas = document.getElementById('myCanvas');
    const ctx = canvas.getContext('2d');
    // Create a linear gradient (left to right)
    const linearGradient = ctx.createLinearGradient(0, 0, 400, 0); // (x0, y0, x1, y1)
    // Add color stops to the linear gradient
    linearGradient.addColorStop(0, 'red'); // Start color at the left
    linearGradient.addColorStop(1, 'blue'); // End color at the right
    // Fill a rectangle with the linear gradient
    ctx.fillStyle = linearGradient;
    ctx.fillRect(10, 10, 380, 80);
    // Create a radial gradient
    const radialGradient = ctx.createRadialGradient(200, 100, 20, 200, 100, 100);
    // (x0, y0, r0, x1, y1, r1)
    // Add color stops to the radial gradient
    radialGradient.addColorStop(0, 'yellow'); // Inner circle color
    radialGradient.addColorStop(1, 'green'); // Outer circle color
    // Fill a circle with the radial gradient
    ctx.fillStyle = radialGradient;
    ctx.beginPath();
    ctx.arc(200, 100, 80, 0, Math.PI * 2);
    ctx.fill();
  </script>
</body>
</html>

```

Fig. 6.6: Canvas Radial gradients example

Output

Canvas Gradients Example

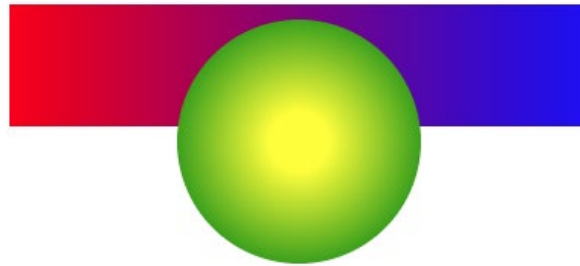


Fig. 6.7: Output for Canvas Radial gradients example

Code Explanation

We start with an HTML5 document structure and create a canvas element with an id of "myCanvas" and set its width and height.

In the JavaScript section:

- We obtain the canvas element and its 2D rendering context (ctx) using `getElementById` and `getContext`.
- Create a linear gradient (linearGradient) using `ctx.createLinearGradient` with coordinates (0, 0) (start) to (400, 0) (end).
- Add color stops to the linear gradient using `linearGradient.addColorStop`. In this case, we start with red at the left (0) and transition to blue at the right (1).
- Set the fill style of the context (`ctx.fillStyle`) to the linear gradient.
- Fill a rectangle (using `fillRect`) with the linear gradient.
- Next, we create a radial gradient (radialGradient) using `ctx.createRadialGradient` with specific coordinates and radii.
- Add color stops to the radial gradient, transitioning from yellow at the center (0) to green at the outer circle (1).
- Set the fill style to the radial gradient.
- Use `ctx.beginPath` to begin drawing a circle.
- Define a circle using `ctx.arc` with coordinates (200, 100) as the center and a radius of 80.

Finally, we fill the circle with the radial gradient, creating a visually appealing gradient-filled circle on the canvas.

6.4 Using Images with the Canvas Tag:

The HTML5 `<canvas>` tag allows you to draw and manipulate graphics, including images, directly on a web page as shown in Figure 6.8.

You can load, display, and interact with images within the canvas, making it a versatile tool for creating interactive web applications.


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Canvas Image Example</title>
</head>
<body>
  <h1>JavaScript Canvas Image Example</h1>
  <canvas id="canvas" width="400" height="200"></canvas>
  <script>
    const canvas = document.getElementById("canvas")
    const ctx = canvas.getContext("2d")
    // Create an image object
    const img = new Image()
    img.src = 'sample-image.jpg'
    // Draw the image on the canvas when it's loaded
    img.onload = function() {
      ctx.drawImage(img, 0, 0)
    }
  </script>
</body>
</html>

```

Fig. 6.8: JavaScript Canvas Image example

Output



Fig. 6.9: Output for JavaScript Canvas Image example

6.4.2 Image Patterns:

Image patterns involve using an image as a repeating pattern or texture within the canvas.

You can create a pattern using the create Pattern method, which takes an image as input and returns a pattern object as shown in Figure 6.10.

This pattern can then be used as a fill style when drawing shapes or paths on the canvas.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Canvas Image Pattern Example</title>
</head>
<body>
  <canvas id="canvas" width="400" height="200"></canvas>
  <script>
    const canvas = document.getElementById('canvas');
    const ctx = canvas.getContext('2d');

    // Create an image pattern
    const img = new Image();
    img.src = 'lamp.jpg';

    img.onload = function() {
      const pattern = ctx.createPattern(img, 'repeat');
      ctx.fillStyle = pattern;
      ctx.fillRect(0, 0, canvas.width, canvas.height);

      // Draw a circle with a different style
      ctx.beginPath();
      ctx.arc(canvas.width / 2, canvas.height / 2, 40, 0, Math.PI * 2);
      ctx.strokeStyle = 'red'; // Set the stroke color to red
      ctx.lineWidth = 5; // Set the line width to 5 pixels
      ctx.stroke(); // Draw the circle border
    };
  </script>
</body>
</html>

```

Fig. 6.10: Image as input and returns a pattern object

Output

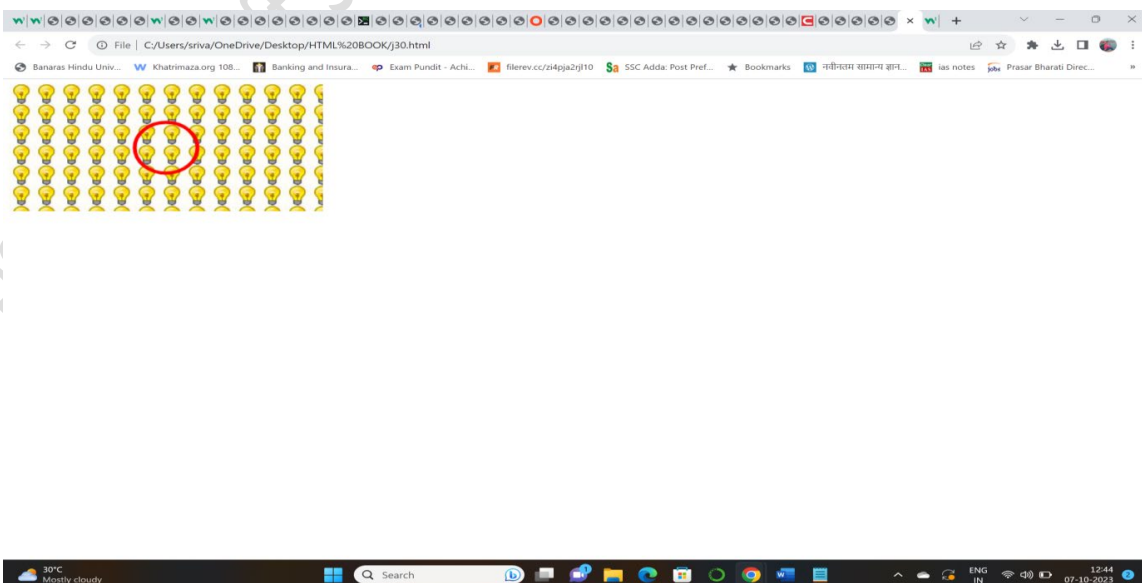


Fig. 6.11: Output for Image as input and returns a pattern object

Check Your Progress

CHECK YOUR PROGRESS

A. MULTIPLE CHOICE QUESTIONS

1. What is the purpose of the HTML5 <canvas> element? (a) To play videos (b) To draw graphics and animations (c) To create forms (d) To display text
2. To obtain a 2D rendering context for a canvas in JavaScript, you would use: (a) document.getContext('canvas') (b) document.canvas.get2DContext() (c) document.querySelector('<canvas>') (d) canvas.getContext('2d')
3. Which method is used to draw a filled rectangle on an HTML5 canvas? (a) drawRect (b) fillRect (c) strokeRect (d) drawFilledRect
4. What's the purpose of the createLinearGradient method in canvas programming? (a) To create a repeating pattern (b) To draw a circle (c) To create a linear gradient (d) To clear the canvas
5. Which property is used to set the style of the ends of lines in canvas drawing? (a) lineCap (b) lineJoin (c) lineStyle (d) lineEnds
6. What's the primary function of timers in canvas animations? (a) To draw objects on the canvas (b) To handle keyboard input (c) To clear the canvas (d) To control the timing of animation frames
7. How can you create a simple animation using requestAnimationFrame in JavaScript? (a) By using the clearRect method (b) By handling mouse events (c) By calling requestAnimationFrame in a loop (d) By changing the canvas size
8. What is the purpose of the keydown event in canvas animation with keyboard input? (a) To start the animation (b) To handle key presses (c) To handle key releases (d) To clear the canvas
9. Which event is used to capture mouse clicks in canvas-based animations? (a) mousedown (b) mousemove (c) mouseup (d) keydown
10. What is the main purpose of touch events in canvas programming? (a) To detect keyboard input (b) To capture interactions on touch-enabled devices (c) To clear the canvas (d) To create complex shapes

B. Fill in the blanks

1. The HTML5 <canvas> element allows you to draw graphics, animations, and interactive content directly in a web page using _____.
2. To work with the canvas in JavaScript, you need to obtain a reference to it using its _____.
3. The method used to get a 2D rendering context in canvas programming is _____.
4. To draw a filled rectangle on an HTML5 canvas, you can use the _____ method.
5. The fillRect method takes parameters for the _____ position, width, and height of the rectangle.
6. The lineCap property can be set to "butt," "round," or "square" to define how the ends of lines should appear, such as "_____" for a square shape.

7. The canvas API allows you to create custom paths and draw lines using methods like _____.
8. The method that starts a new path in canvas drawing is _____.
9. The method that moves the path drawing position to a specified point without drawing a line is _____.
10. The method used to draw an image on the canvas is _____.

C. True or False

1. The HTML5 <canvas> element is primarily used to create forms on web pages.
2. The getContext('2d') method is used to obtain a 3D rendering context for the canvas in JavaScript.
3. The strokeRect method is used to draw the filled area of a rectangle on an HTML5 canvas.
4. The arc method is used to draw straight lines on the canvas.
5. Line caps and joins are properties that affect the appearance of the ends of lines and the intersections of lines in canvas drawing.
6. The beginPath method is used to close a path in canvas programming.
7. Radial gradients in HTML5 canvas are used for creating smooth transitions of colors along a straight line.
8. The drawImage method is used to draw text on the canvas.
9. Image scaling involves changing the width and height of an image when drawing it on the canvas.
10. In simple canvas animations, timers are used to control the timing of animation frames.

D. Short Question Answers

1. What is the purpose of the HTML5 <canvas> element?
2. How do you obtain a 2D rendering context for the <canvas> element in JavaScript?
3. Which method is used to draw a filled rectangle on the canvas?
4. What attributes are used to define the size of a <canvas> element?
5. How can you create a linear gradient in HTML5 canvas?
6. What is the purpose of the lineCap property in canvas drawing?
7. How do you clear the canvas before each frame of animation?
8. What event listeners can be used to detect mouse interactions in canvas animations?
9. How do you create a radial gradient in HTML5 canvas?
10. What is the purpose of the arc() method in canvas drawing?

ANSWER KEY**MODULE 1.WEB DEVELOPMENT ESSENTIALS****Session1. Information Technology (IT) and Information Technology Enabled Services (ITeS)****A. Multiple Choice Questions (MCQ)**

1. (b) 2. (c) 3. (c) 4. (d) 5. (b) 6. (d) 7. (c) 8. (c) 9. (d) 10. (a)

B. Fill in the blanks

1. Information 2. Business process 3. Storage devices 4. Devices, data 5. Computing 6. Automation 7. Cloud-based 8. Web 9. Video games 10. Blockchain

C. True or False

1. (T) 2. (T) 3. (F) 4. (F) 5. (T) 6. (F) 7. (T) 8. (F) 9. (F) 10. (F)

Session2. Web Design and Development**A. Multiple Choice Questions (MCQ)**

1. (c) 2. (c) 3. (b) 4. (a) 5. (b) 6. (c) 7. (a) 8. (d) 9. (a) 10. (c)

B. Fill in the blanks

1. Front-end 2. Website 3. Design 4. Stages 5. User interface 6. Store 7. Website 8. Static 9. Browsers 10. Web design

C. True or False

1. (F) 2. (T) 3. (F) 4. (F) 5. (T) 6. (T) 7. (F) 8. (F) 9. (F) 10. (T)

Session3. Web Design and Development Tools**A. Multiple Choice Questions (MCQ)**

1. (c) 2. (c) 3. (c) 4. (b) 5. (c) 6. (b) 7. (c) 8. (c) 9. (d) 10. (c)

B. Fill in the blanks

1. Structure 2. Responsive 3. Interactivity 4. Data security 5. Websites 6. HTML 7. Web 8. Javascript 9. High-level 10. SQL

C. True or False

1. (F) 2. (T) 3. (F) 4. (F) 5. (T) 6. (F) 7. (F) 8. (F) 9. (T) 10. True

Session4. Application Development Models of IT**A. Multiple Choice Questions (MCQ)**

1. (b) 2. (b) 3. (a) 4. (b) 5. (b) 6. (c) 7. (b) 8. (b) 9. (a) 10. (c)

B. Fill in the blanks

1. Cycles or iterations 2. Shared 3. Plans 4. Risk Management 5. Change 6. Verification 7. Repository 8. Workflow 9. Feedback 10. Lifecycle

C. True or False

1. (F) 2. (F) 3. (T) 4. (F) 5. (F) 6. (F) 7. (F) 8. (T) 9. (F) 10. (F)

Session5. Web Designing Specifications**A. Multiple Choice Questions (MCQ)**

1. (b) 2. (c) 3. (a) 4. (b) 5. (b) 6. (b) 7. (c) 8. (d) 9. (b) 10. (b)

B. Fill in the blanks

1. Risks 2. End-user 3. Response 4. Blueprint 5. Operating 6. URS 7. Security 8. Purpose 9. Non-Functional 10. screen sizes, devices

C. True or False

1. (F) 2. (F) 3. (F) 4. (T) 5. (T) 6. (F) 7. (T) 8. (F) 9. (F) 10. (F)

Session6. Low-level and High-level Design for Programming**A. Multiple-Choice Questions (MCQs)**

1. (b) 2. (b) 3. (a) 4. (a) 5. (b) 6. (b) 7. (c) 8. (b) 9. (c) 10. (b)

B. Fill in the blanks

1. Conceptual 2. Methods 3. Redundancy 4. Abuse 5. Low-level 6. Requirements 7. Fair usage 8. Design 9. Data flow 10. Breaking

C. True or False

1. (T) 2. (T) 3. (F) 4. (T) 5. (F) 6. (T) 7. (T) 8. (T) 9. (T) 10. (F)

Session7. Test and identify Design Defects**A. Multiple-Choice Questions (MCQs)**

1. (c) 2. (a) 3. (b) 4. (c) 5. (b) 6. (d) 7. (d) 8. (c) 9. (c) 10. (b)

B. Fill in the blanks

1. Arithmetic Defects 2. Wrong 3. Style 4. Needs 5. Components 6. Unauthorized 7. Regression 8. Scalability 9. Browsers 10. Identification

C. True or False

1. (F) 2. (F) 3. (F) 4. (T) 5. (T) 6. (F) 7. (T) 8. (F) 9. (T) 10. (F)

Session8. Resolve Design Defects**A. Multiple-Choice Questions (MCQs)**

1. (c) 2. (b) 3. (b) 4. (a) 5. (c) 6. (c) 7. (c) 8. (b) 9. (c) 10. (c)

B. Fill in the blanks

1. Product 2. Identify 3. First 4. Resolution 5. Functionality 6. Monitoring 7. Development 8. Experiment 9. Addressed 10. Revisions

C. True or False

1. (F) 2. (F) 3. (F) 4. (F) 5. (T) 6. (F) 7. (T) 8. (F) 9. (F) 10. (F)

MODULE 2. HTML AND CSS**Session9. HTML****A. Multiple-Choice Questions (MCQs)**

1. (a) 2. (a) 3. (b) 4. (c) 5. (c) 6. (b) 7. (b) 8. (c) 9. (c) 10. (a)

B. Fill in the blanks

1. Hypertext Markup 2. HTML5 3. Angle 4. <p> 5. Element 6. Hyperlinks 7. <blockquote> 8. <div> 9. 10. HTML

C. True or False

1. (T) 2. (F) 3. (T) 4. (F) 5. (T) 6. (F) 7. (F) 8. (T) 9. (F) 10. (T)

Session 10. Concept of lists – Unordered lists, Ordered lists, Definition list

A. Multiple-Choice Questions (MCQs)

1. (a) 2. (b) 3. (b) 4. (d) 5. (c) 6. (b) 7. (c) 8. (b) 9. (b) 10. (b)

B. Fill in the blanks

1. 2. Ordered 3. 4. Definition 5. <dl> 6. 7. Image 8. Mapping 9. Hyperlinks 10. <tbody>

C. True or False

1. (F) 2. (T) 3. (F) 4. (F) 5. (F) 6. (T) 7. (T) 8. (T) 9. (F) 10. (T)

Session 11. CSS

A. Multiple-Choice Questions (MCQs)

1. (b) 2. (b) 3. (c) 4. (a) 5. (b) 6. (d) 7. (b) 8. (b) 9. (c) 10. (b)

B. Fill in the blanks

1. Presentation 2. asterisk (*) 3. Elements 4. text-align 5. Parent 6. letter-spacing 7. Descendant 8. Responsive 9. Head 10. Internal

C. True or False

1. () 2. () 3. () 4. () 5. () 6. () 7. () 8. () 9. () 10. ()

Session 12. CSS Box Model

A. Multiple-Choice Questions (MCQs)

1. (c) 2. (c) 3. (a) 4. (a) 5. (c) 6. (a) 7. (b) 8. (b) 9. (c) 10. (a)

B. Fill in the blanks

1. CSS 2. Space 3. Dimensions 4. Max-width 5. Position 6. Float 7. Hidden 8. Inline 9. Overflows 10. Right

C. True or False

1. (F) 2. (T) 3. (T) 4. (F) 5. (F) 6. (F) 7. (F) 8. (F) 9. (T) 10. (F)

MODULE 3. WEB DEVELOPMENT USING JAVA SCRIPT

Session 13. JavaScript

A. Multiple Choice Questions (MCQ)

1. (c) 2. (b) 3. (b) 4. (b) 5. (b) 6. (a) 7. (d) 8. (d) 9. (a) 10. (b)

B. Fill in the blanks

1. Const 2. JavaScript 3. Node.js 4. Sensitive 5. alert () 6. Browser 7. History 8. Cookies 9. Location 10. Execution

C. True or False

1. (T) 2. (F) 3. (T) 4. (F) 5. (T) 6. (T) 7. (F) 8. (T) 9. (F) 10. (T)

Session 14. Conditional Logic and Flow Control**A. Multiple Choice Questions (MCQ)**

1. (c) 2. (b) 3. (a) 4. (a) 5. (c) 6. (c) 7. (a) 8. (b) 9. (c) 10. (b)

B. Fill in the blanks

1. True 2. Conditional 3. Compare 4. Boolean 5. Nested 6. Switch 7. Break 8. Continue 9. Iterate 10. for...in

C. True or False

1. (T) 2. (T) 3. (F) 4. (T) 5. (F) 6. (F) 7. (T) 8. (T) 9. (F) 10. (T)

Session 15. Arrays and Functions**A. Multiple Choice Questions (MCQ)**

1. (b) 2. (b) 3. (a) 4. (c) 5. (b) 6. (b) 7. (c) 8. (c) 9. (c) 10. (c)

B. Fill in the blanks

1. pop() 2. Reusable 3. Parentheses 4. Return 5. Native 6. Math.random() 7. String() 8. push() 9. shift() 10. function

C. True or False

1. (T) 2. (F) 3. (T) 4. (T) 5. (F) 6. (T) 7. (F) 8. (F) 9. (T) 10. (T)

Session 16. String Manipulation**A. Multiple Choice Questions (MCQ)**

1. (b) 2. (a) 3. (c) 4. (b) 5. (d) 6. (c) 7. (b) 8. (b) 9. (b) 10. (d)

B. Fill in the blanks

1. Combining 2. length() 3. toLowerCase() 4. indexOf() 5. substring() 6. Validation 7. split() 8. date() 9. isValidDate 10. charAt()

C. True or False

1. (F) 2. (F) 3. (F) 4. (F) 5. (T) 6. (T) 7. (T) 8. (T) 9. (F) 10. (F)

Session 17. Manipulate Images using Javascript**A. Multiple Choice Questions (MCQ)**

1. (b) 2. (a) 3. (b) 4. (c) 5. (b) 6. (c) 7. (b) 8. (a) 9. (c) 10. (b)

B. Fill in the blanks

1. Image 2. DrawImage 3. GetImageData 4. Todataurl 5. Geolocation 6. Browser 7. Location 8. Track 9. Modify 10. DOM

C. True or False

1 (F) 2. (F) 3. (T) 4. (F) 5. (F) 6. (F) 7. (T) 8. (F) 9. (F) 10. (F)

Session 18. HTML5 Canvas**A. Multiple Choice Questions (MCQ)**

1. (b) 2. (d) 3. (b) 4. (c) 5. (a) 6. (d) 7. (c) 8. (b) 9. (a) 10. (b)

B. Fill in the blanks

1. JavaScript 2. Id 3. getContext("2d") 4. fillRect 5. x, y, width, and height 6. "square" 7. beginPath, moveTo, and.lineTo 8. beginPath 9. moveTo 10. drawImage

C. True or False

1. (F) 2. (F) 3. (F) 4. (F) 5. (T) 6. (F) 7. (F) 8. (F) 9. (T) 10. (T)

PSSCIVE Draft Study Material © Not to be Published