SGOU
Higher Education for All

Web Technology

COURSE CODE: B21CA09DC

Bachelor of Computer Applications

Discipline Core Course

Self Learning Material

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Web Technology

Course Code: B21CA09DC

Semester - V

Discipline Core Course
Undergraduate Programme
Bachelor of Computer Applications
Self Learning Material

(With Model Question Paper Sets)

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

WEB TECHNOLOGY

Course Code: B21CA09DC
Semester-V
Discipline Core Course
Bachelor of Computer Applications

Scan this QR Code for reading the SLM on a digital device.

Edition
January 2026

ISBN 978-81-996171-5-5

9 788199 617155

www.sgou.ac.in

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear

With immense joy and excitement, I extend my heartfelt greetings to all of you and warmly welcome you to Sreenarayanaguru Open University.

Established in September 2020 as a state-driven initiative, Sreenarayanaguru Open University is dedicated to advancing higher education through open and distance learning. Our vision is guided by the principle of "access and quality define equity," laying the foundation for a celebration of excellence in education. I am delighted to share that we are steadfast in our commitment to uphold the highest standards and refrain from compromising on the quality of education we offer. The university draws its inspiration from the legacy of Sreenarayana Guru, a revered figure in the Indian renaissance movement. His name serves as a constant reminder for us to prioritize quality in all our academic endeavors.

Sreenarayanaguru Open University operates within the practical framework of the widely recognized "blended format." Acknowledging the constraints faced by distance learners in accessing traditional classroom settings, we have curated a pedagogical approach centered on three main components: Self Learning Material, Classroom Counselling, and Virtual Modes. This comprehensive blend is poised to deliver dynamic learning and teaching experiences, maximizing engagement and effectiveness. Our unwavering commitment to quality ensures excellence across all aspects of our educational initiatives.

The University aims to offer you an engaging and stimulating educational environment that fosters active learning. The SLM is designed to offer a comprehensive and cohesive learning experience, fostering a deep interest in the study of technological advancements in IT. Careful consideration has been given to ensure a logical progression of topics, facilitating a clear understanding of the discipline's evolution. The curriculum is thoughtfully crafted to provide ample opportunities for students to navigate through the current trends in information technology. Furthermore, this course is designed to provide essential insights into computer hardware, software classification, and foundational HTML concepts crucial for web development.

We assure you that the university student support services will closely stay with you for the redressal of your grievances during your studentship. Feel free to write to us about anything that seems relevant regarding the academic programme.

Wish you the best.

Regards,
Dr. Jagathy Raj V.P.

01-01-2026

Contents

BLOCK 1

Advanced HTML

Linking in HTML

Learning Outcomes

After completing this unit, learners will be able to:

- define hyperlinks and their purpose in HTML
- explain the difference between external and internal links
- describe how to use an image as a clickable link
- identify the steps to create a link that opens an email application
- differentiate between linking to another page and linking within the same page

Prerequisites

Imagine visiting your favorite online store or reading an article on a website. How do you move from one page to another, or jump directly to the part you are interested in? How does clicking on a product image take you to its details, or how does clicking an email link open your email program? All these actions are possible because of hyperlinks, one of the most important features of HTML.

Before learning this unit, you are already familiar with the idea of clicking links or buttons on websites to reach information quickly. In this unit, you will learn how to create such links yourself using the <a> (anchor) tag. You will explore external links that take users to other websites, internal links that connect pages within your website or scroll to sections within the same page, image links that make pictures clickable, and email links that open the user's email application with a pre-filled recipient address. By understanding and practicing these types of links, you will be able to make web pages interactive, easy to navigate, and similar in functionality to professional websites you use every day.

Keywords

HTML links, anchor tag, external linking, internal linking, mailto link

Discussion

1.1.1 Linking in HTML

Links, or hyperlinks, are one of the most important features of HTML. They let users move from one web page to another, go to a specific section on the same page, or even open an email program to send a message. Without links, web pages would be isolated with no way to connect them.

In HTML, links are made using the anchor tag <a>. The destination of the link is specified in the href attribute, which tells the browser where to go when the link is clicked. For example: Click here. The clickable part of the link can be text, an image, or other elements on the page. This makes links very useful for creating websites that are easy to navigate.

There are different types of links in HTML. External links take the user to a page on another website. Internal links connect pages within the same website or move the user to a specific section on the same page. Image links are links where an image is clickable, instead of text. Email links open the user’s email program with a recipient address, and optionally a subject or message body. Learning how to use these links is important because they make websites interactive and help users move around easily.

1.1.2 External Linking

An external link is a hyperlink that takes the user to a web page outside of your website. These links are useful when you want to refer your visitors to other websites for information, references, or resources. To create an external link, you write the full web address (URL) in the href attribute of the anchor tag. For example, if you want to link to Wikipedia, you can write the following code:

```
<a href="https://www.wikipedia.org" target="_blank">Visit Wikipedia</a>
```

Here, the text “Visit Wikipedia” becomes clickable because it is placed between the opening <a> and closing tags. The href attribute is used to specify the destination of the link—in this case, it points to the URL of Wikipedia (https://www.wikipedia.org). The target="_blank" attribute is used so that the link opens in a new browser tab, keeping your page open in the original tab. Similarly, you can create a link to Google by using the following code:

```
<a href="https://www.google.com" target="_blank">Go to Google</a>
```

The following program combines both the Wikipedia and Google links to demonstrate how external linking works in a complete HTML page.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>External Linking Example</title>
</head>
<body>
  <h1>External Linking Example</h1>
<p><a href="https://www.wikipedia.org" target="_blank">Visit Wikipedia</a></p>
<p><a href="https://www.google.com" target="_blank">Go to Google</a></p>
</body>
</html>
```

In this program, two external links are created. The first one points to Wikipedia and the second one points to Google. Both links open in new tabs because of the target="_blank" attribute. The href attribute in each <a> tag provides the destination address.

The result of the above program in a web browser is shown below:

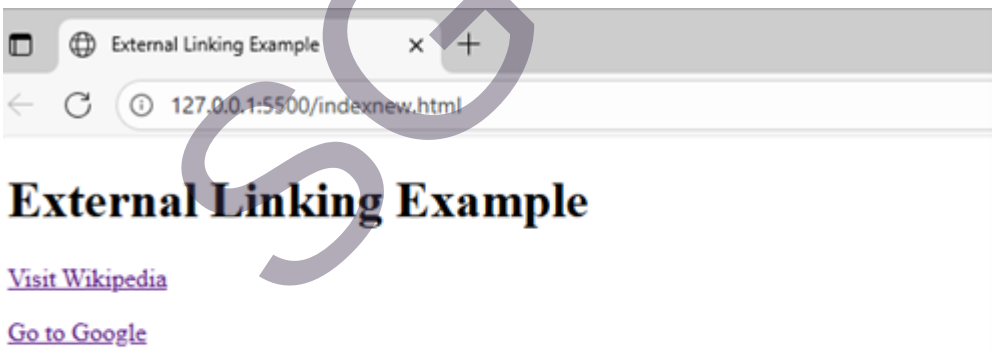


Fig. 1.1.1 Result of external linking program

When you click on the text “Visit Wikipedia”, the browser opens the Wikipedia homepage in a new tab, as shown below:

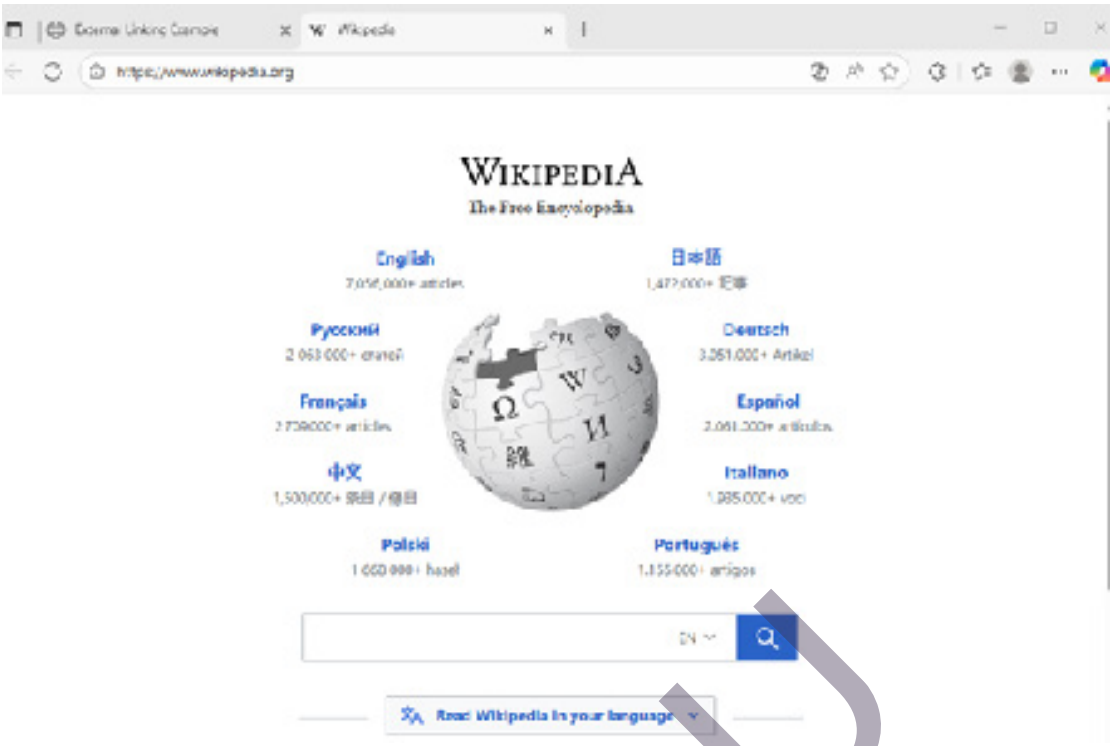


Fig. 1.1.2 Wikipedia homepage opened through external link

Similarly, if you click on the text “Go to Google”, the browser will open the Google search page in a new tab, as shown below:

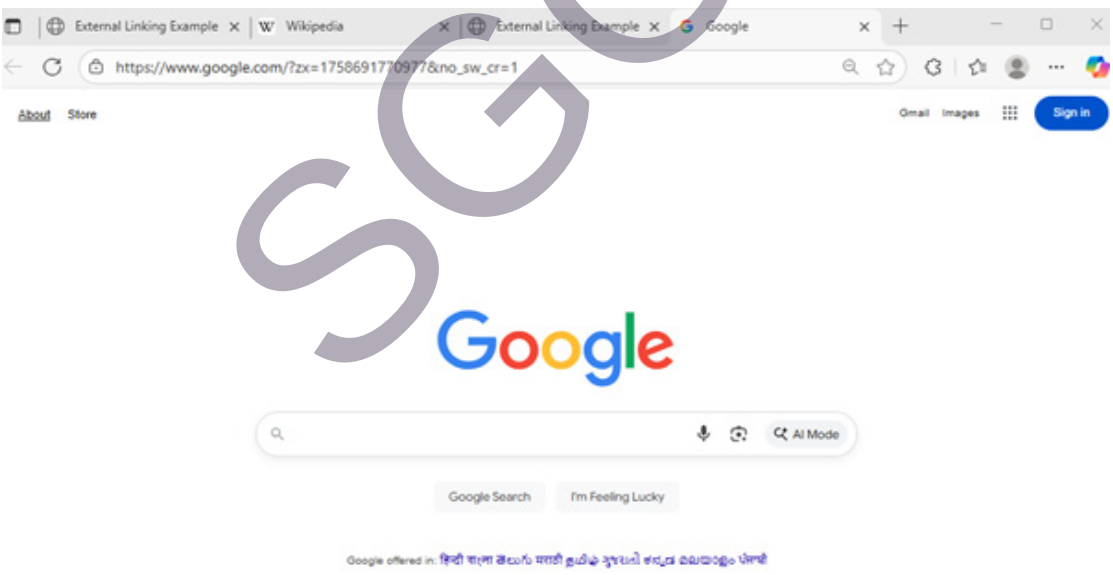


Fig. 1.1.3 Google homepage opened through external link

1.1.3 Internal Linking

Internal linking means creating links that connect one page of your website to another page within the same website or connect one part of a page to another part of the same

page. These links are important because they help visitors easily move around your website without leaving it. There are two main methods of internal linking:

Method 1: Linking to another page within the same website

This method is used to connect one page of your website to another. You just provide the file name of the target page in the href attribute of the anchor tag. For example, if you have a page named about.html, you can link to it like this:

```
<a href="about.html">Go to About Page</a>
```

Here, the text "Go to About Page" becomes clickable. When the user clicks it, the browser opens the about.html page. This method is useful for creating menus, navigation bars, or linking related content within a website.

Method 2: Linking within the same page

This method is used to jump to a specific section of a long page. First, you assign an id to the section you want to link to. Then, you create a link using the # symbol followed by the id name. For example:

```
<a href="#section2">Jump to Section 2</a>

<h2 id="section2">This is Section 2</h2>
```

Clicking "Jump to Section 2" will scroll the page directly to the heading with the id section2. This method is useful in tutorials, articles, or any long page where users may want to quickly jump to specific content.

The following program demonstrates both types of internal linking on a clothing website, including links to another page and links to sections within the same page.

```
<!DOCTYPE html>

<html>

<head>

  <meta charset="utf-8">

  <title>Cloth Store - Internal Linking</title>

</head>

<body>
```

```
<h1>Welcome to Fashion Hub</h1>
<!-- Method 1: Linking to another page -->
<nav>
  <p><a href="about.html">About Us</a></p>
  <p><a href="contact.html">Contact Us</a></p>
</nav>
<!-- Method 2: Linking within the same page -->
<h2>Shop Categories</h2>
<p><a href="#mens">Go to Men's Wear</a></p>
<p><a href="#womens">Go to Women's Wear</a></p>
 <h2 id="mens">Men's Wear</h2>
<p>Check out our latest collection of shirts, trousers, and jackets for men.</p>
 <h2 id="womens">Women's Wear</h2>
 <p>Explore our collection of dresses, tops, and skirts for women.</p>
</body>
</html>
```

In this program, internal linking is demonstrated using a simple clothing website. The homepage contains two types of links. The first type links to other pages within the website, such as about.html and contact.html. Clicking "About Us" opens the about page, which provides information about the store, while clicking "Contact Us" opens the contact page showing the store's email and phone number. The second type of internal link navigates within the same page using the id attribute. The homepage includes sections for Men's Wear and Women's Wear, each with a corresponding link at the top. Clicking on these links scrolls the page directly to the respective section, allowing users to jump quickly to the part they are interested in. This program shows how internal linking can improve website navigation, making it easier for users to access different pages or sections efficiently, similar to real-world e-commerce websites.

The code for the about.html and contact.html pages is shown below:

about.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>About Us - Fashion Hub</title>
</head>
<body>
  <h1>About Us</h1>
  <p>Welcome to Fashion Hub! We offer the latest fashion trends for men and women at affordable prices.</p>
</body>
</html>
```

contact.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Contact Us - Fashion Hub</title>
</head>
<body>
  <h1>Contact Us</h1>
  <p>Email: support@fashionhub.com</p>
  <p>Phone: +91 98765 43210</p>
</body>
</html>
```

The result of the internal linking program can be seen by opening the homepage in a web browser. The homepage displays the main navigation links “About Us” and

"Contact Us", as well as section links "Go to Men's Wear", and "Go to Women's Wear" as shown in figure 1.1.4.

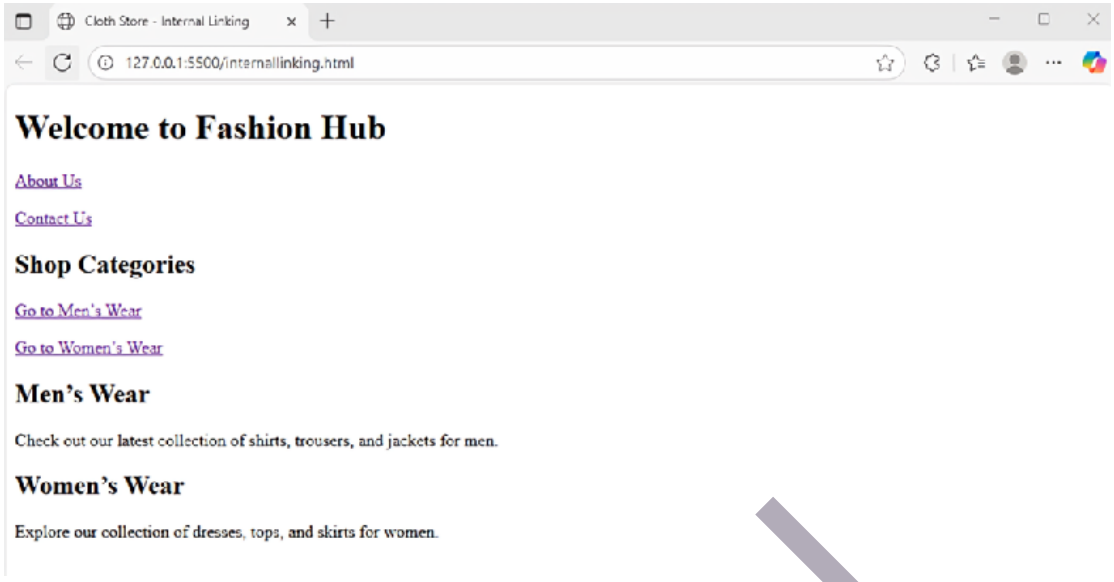


Fig. 1.1.4 Homepage showing navigation and section links

Clicking "About Us" opens the about.html page, which shows information about the clothing store. Clicking "Contact Us" opens the contact.html page, which displays the store's email and phone number are shown in Figures 1.1.5 and 1.1.6.

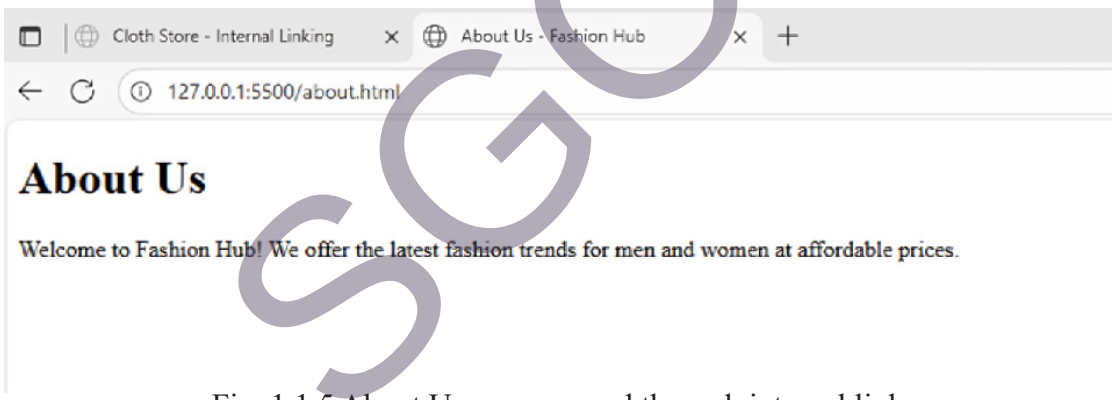


Fig. 1.1.5 About Us page opened through internal link

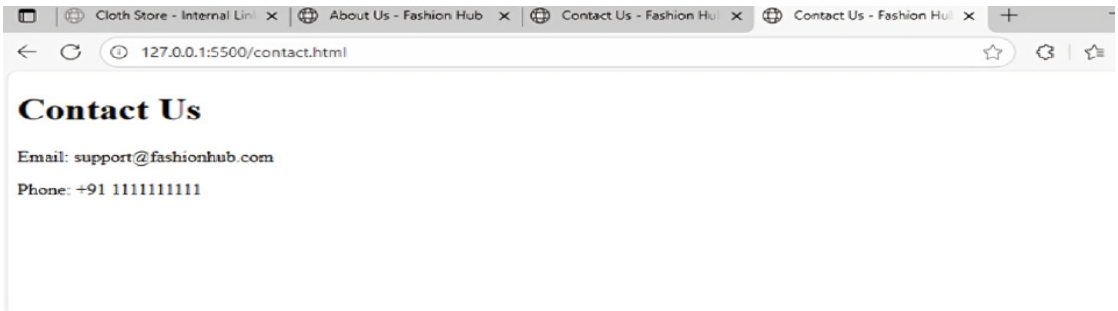


Fig. 1.1.6 Contact Us page opened through internal link

On the homepage, clicking the section links scrolls the page directly to the corresponding sections, allowing users to quickly access Men's Wear, or Women's Wear. These results demonstrate both types of internal linking navigating to other pages and navigating within the same page making the website more user friendly.

1.1.4 Using Image as a Link

In HTML, hyperlinks are not limited to text; images can also serve as clickable links. This is useful for websites that want to create a more interactive and visually appealing navigation system. For example, on e-commerce websites, users can click on product images to see more details, or on banners to visit promotional pages. Making an image clickable is done by placing the <img> tag inside an anchor <a> tag.

The anchor tag <a> works the same way as with text links. Its href attribute specifies the destination URL where the user will be taken after clicking the image. The image tag <img> specifies the image file to be displayed on the page. You can also add attributes like alt (alternative text), width, and height to improve accessibility and control the size of the image.

Syntax:

```
<a href="destination_url">
  <img src="image_file" alt="description" width="value" height="value">
</a>
```

- tells where the link will take the user. It can be another page in the same website, a different website, or any file.
- <img src="image_file" alt="description"> shows the image on the web page. The alt text is a short description that appears if the image is not loaded.
- width and height decide how big or small the image will look on the screen.
- is used to close the link tag. Everything between <a> and becomes clickable.

The following program shows how an image can be used as a hyperlink in a simple web page. In this example, an image of a T shirt is displayed on the homepage, and when the user clicks on the image, it opens another page called product.html, which can contain details about the product. This method is commonly used in online shopping websites, where clicking on a product picture takes the user to a page with more information about that item.

IMAGELINK.HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
```

```
  <title>Using Image as a Link</title>
</head>
<body>
  <h1>Product Showcase</h1>
  <!-- Image acting as a link -->
  <a href="product.html">
    <img src="shirt.jpg" alt="Men's Shirt" width="200" height="200">
  </a>
  <p>Click the image to view product details.</p>
</body>
</html>
```

The output of the above program is shown below figure 1.1.7.

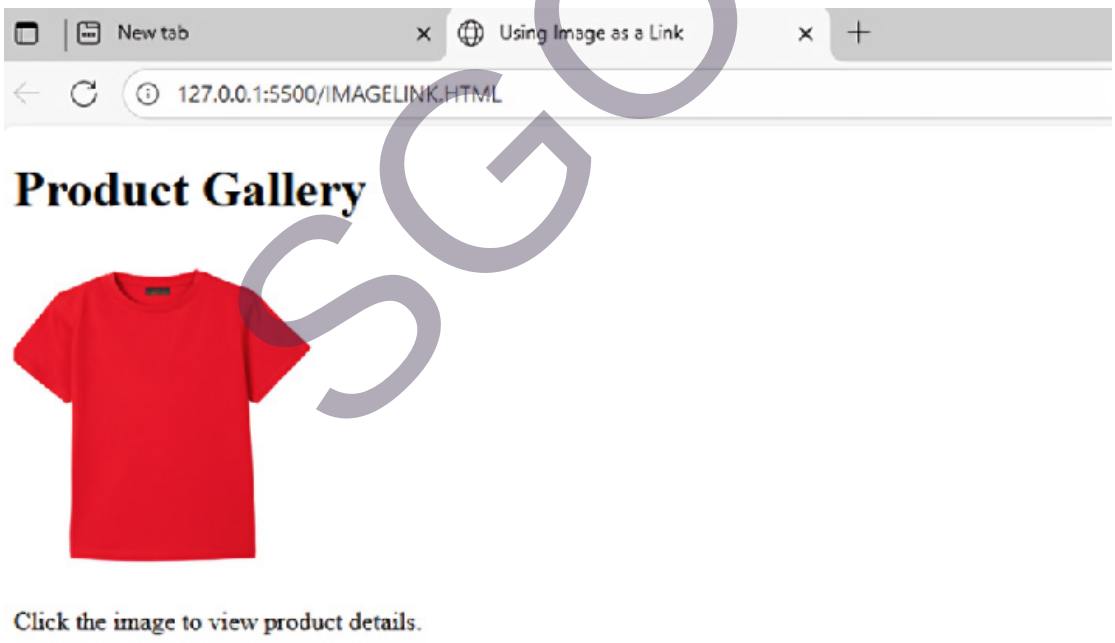


Fig. 1.1.7 Home Page with clickable shirt image

The Home Page displays the T shirt image, and when the image is clicked, it opens the Product Details Page with information about the shirt, we also need to create a separate file called product.html. This page contains the details of the product such as name, description, price, and available sizes. The following code shows the structure of the product.html page.

product.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Product Details</title>
</head>
<body>
    <h1>Men's T-Shirt</h1>
    <p>This is a stylish and comfortable men's T-shirt, perfect for casual and formal wear.</p>
    <p>Price: $25</p>
    <p>Available Sizes: S, M, L, XL</p>
    <p><a href="IMAGELINK.HTML">Back to Home</a></p>
</body>
</html>
```

The output of the Product Details Page is shown in Figure 1.1.8. When the user clicks on the shirt image from the home page, this page opens and displays the product information.

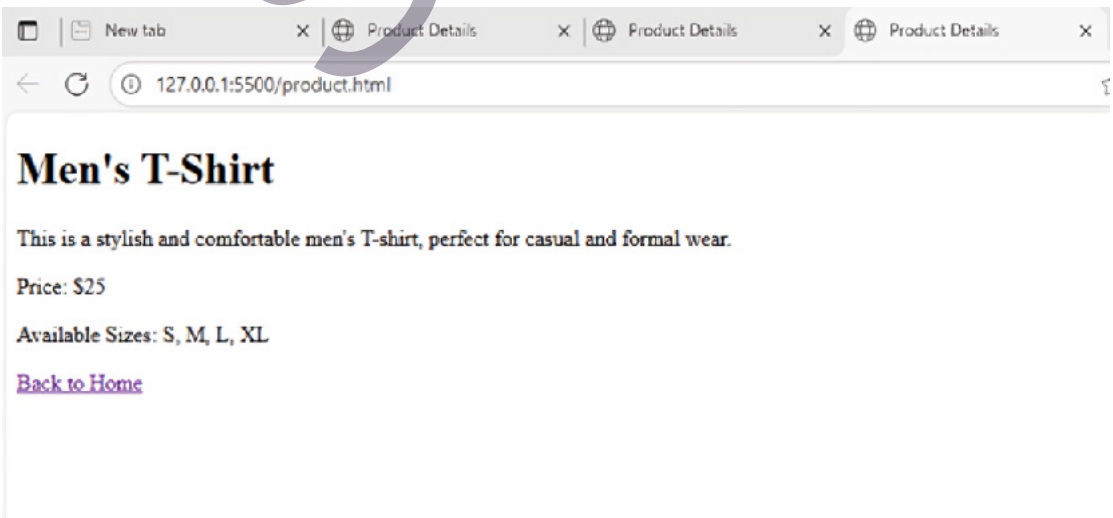


Fig. 1.1.8 Product details page displayed after clicking the image

1.1.5 Linking to Email

In addition to linking web pages and images, HTML allows creating links that can open an email application so the user can send a message directly. This is achieved using the mailto: keyword inside the href attribute of the <a> (anchor) tag. When the user clicks such a link, their default email program opens automatically with the recipient's address already filled in.

Syntax:

```
<a href="mailto:email_address">Link Text</a>
```

- mailto: tells the browser that the link should open an email program instead of a webpage.
- email_address is the email address where the message will be sent.
- Link Text is the clickable text that appears on the webpage.

The following program demonstrates how to create an email link in HTML. When the user clicks the link, their default email application opens with the recipient's address already filled in, allowing them to send a message directly from the webpage.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Linking to Email</title>
</head>
<body>
   <h1>Contact Us</h1>
   <p>If you have any queries, feel free to
      <a href="mailto:store@example.com">send us an email</a>
   </p>
</body>
</html>
```

In this program, the text "send us an email" is clickable. When the user clicks on it, the default email program opens with the recipient's address already filled in as store@example.com. The user can then type their subject and message and send it directly.

The output displays a heading “Contact Us” and a clickable link “send us an email.” Clicking the link opens the email application on the user’s system. This is shown in Figure 1.1.9.

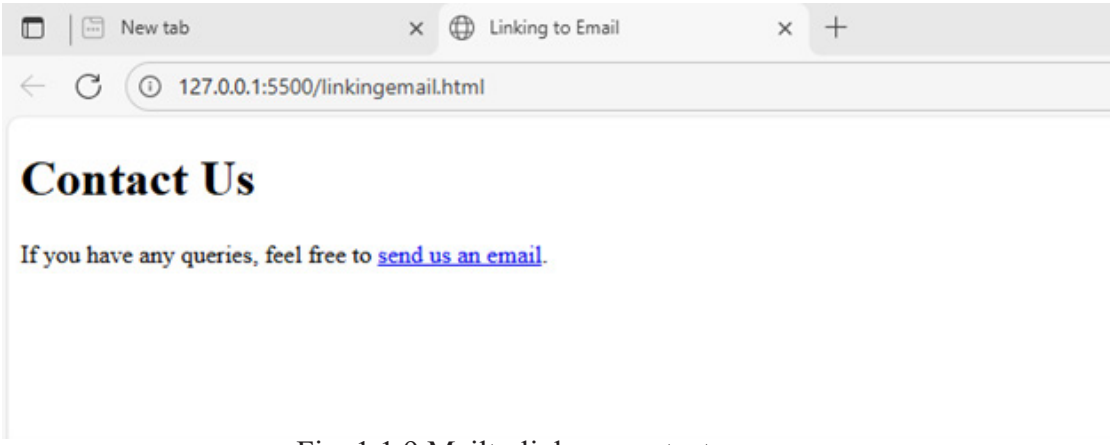


Fig. 1.1.9 Mailto link on contact us page

Recap

- Hyperlinks in HTML allow navigation between web pages, sections, images, or email programs.
- Links are created using the <a> (anchor) tag, and the href attribute specifies the destination URL or resource.
- External Linking: Links that point to pages on other websites; open in new tabs using target="_blank"
- Internal Linking:
  - Method 1: Link to another page within the same website
  - Method 2: Link to a specific section within the same page by assigning an id to the target element and using #id-name in the href.
- Using Image as a Link: Placing an <img> tag inside an <a> tag makes an image clickable; alt, width, and height attributes improve accessibility and control size.
- Linking to Email: Using mailto: in the href attribute opens the default email application with recipient pre-filled.

Objective Type Questions

1. Which HTML tag is used to create a hyperlink?
2. Which attribute specifies the destination of a link?
3. What type of link opens a webpage on another website?

4. What attribute is used to open a link in a new tab?
5. What value of target opens a link in a new tab?
6. Which type of link connects pages within the same website?
7. Which symbol is used to link to a section within the same page?
8. Which attribute is used to identify a section in a page?
9. Which HTML tag is used to display an image?
10. Which attribute in <img> provides alternative text?
11. What keyword is used in a link to open an email program?
12. Which attribute controls the width of an image?
13. Which attribute controls the height of an image?

Answers to Objective Type Questions

1. <a>
2. href
3. External
4. target
5. _blank
6. Internal
7. #
8. id
9. <img>
10. alt
11. mailto
12. width
13. height

Assignments

1. Define what a hyperlink is in HTML. Describe the purpose of the href and target attributes in the anchor tag (<a>). Provide an example of a hyperlink that opens in a new tab.
2. Differentiate between external linking and internal linking in HTML. Give one example of code for each.
3. Explain the steps and HTML code needed to create a link that jumps to a specific section on the same web page. Why is this method useful?
4. Consider the following scenario: You are designing a website for a bookstore. How would you create a navigation menu that links to other internal pages like “Home”, “Books”, and “Contact Us”? Write the HTML code.
5. How can you use an image as a hyperlink in HTML? Write a small HTML snippet where clicking on a book cover image opens a “details.html” page.
6. What is a mailto link in HTML? Write the HTML code for a contact page where clicking on the phrase “Email Customer Support” opens the default mail app with the address support@bookstore.com.

Reference

1. Negrino, T., & Smith, D. (2011). *JavaScript and Ajax for the web: Visual QuickStart guide* (8th ed.). Peachpit Press.
2. Felke-Morris, T. (2020). *Web development and design foundations with HTML5* (9th ed.). Pearson.
3. McFarland, D. S. (2015). *CSS: The missing manual* (4th ed.). O’Reilly Media.

Suggested Reading

1. Duckett, J. (2011). HTML and CSS: Design and build websites. Wiley.
2. Nixon, R. (2021). Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5 (6th ed.). O’Reilly Media.
3. Castro, E., & Hyslop, B. (2013). HTML5 and CSS: Visual QuickStart guide (8th ed.). Peachpit Press.
4. Freeman, E. (2012). Head First HTML and CSS (2nd ed.). O’Reilly Media.

Layers in HTML

Learning Outcomes

After completing this unit, learners will be able to:

- define what layers are in HTML
- identify the CSS properties used for layering elements
- recall the purpose of the z-index property
- define what an image map is in HTML
- list the main tags used to create an image map

Prerequisites

When you visit a website, you often notice that content is not just displayed in a simple top to bottom order. For example, advertisements sometimes appear above the main text, menus stay fixed at the top of the page when you scroll, and pop up windows overlap other elements. In the same way, you might have also seen websites where a single image, such as a world map or a product catalog, allows you to click on different parts to go to different pages. These everyday experiences on the web are built using concepts that extend what you already know about placing text, images, and links in HTML.

Up to this point, you have learned how a webpage shows text, pictures, and links in a simple way. But the web can do more than that. Imagine being able to place one part of the page above another, like a notice pinned over a board, or turning a single picture into an interactive map where each area leads you to new information. These ideas help transform a plain page into something more lively and engaging for the user.

By learning these concepts, you will be able to connect your existing knowledge of basic HTML structure with more advanced techniques for controlling layout and interactivity. This will prepare you to design webpages that are not only functional but also engaging and user-friendly.

Keywords

HTML layers, image maps, overlapping elements, interactive images

Discussion

1.2.1 What Are Layers in HTML?

In HTML, layers refer to placing elements on top of or below other elements just like putting one sheet of paper over another. Earlier, this was done using the <layer> tag in Netscape browsers, but that tag is now outdated and no longer supported.

Today, we use the tag with CSS properties like position, top, left, and z-index to control the position and order of elements on a webpage. This allows you to create pop-ups, banners, sticky headers, or any content that appears "above" the rest of the page.

Important CSS Properties for Layers

- position: Determines how an element is placed on the screen (static, relative, absolute, fixed, sticky).
- top, left: Set the position of the element relative to the page or its parent.
- z-index: Controls the "stacking" order. Higher values appear on top.

Simple Example of Using Layers

Let's consider one example of how layers can be created using the tag and some simple CSS. In this example, we will create two boxes one blue and one green and place them in such a way that the green box overlaps the blue one.

```
<!DOCTYPE html>
<html>
<head>
  <title>Layer Example</title>
  <style>
        #blueBox {
        position: absolute;
        top: 50px;
        left: 50px;
        width: 200px;
```

```
        height: 100px;
        background-color: lightblue;
        z-index: 1;
        }
        #greenBox {
        position: absolute;
        top: 80px;
        left: 100px;
        width: 200px;
        height: 100px;
        background-color: lightgreen;
        z-index: 2;
        }
  </style>
</head>
<body>
  <div id="blueBox">This is the Blue Box (Behind)</div>
  <div id="greenBox">This is the Green Box (On Top)</div>
</body>
</html>
```

In this program, we created two rectangular boxes using the tag. The blue box is shown first, and the green box is placed slightly lower and more to the right so that it overlaps the blue box. We used CSS properties like position: absolute; to freely place the boxes anywhere on the screen. The key part is the use of the z-index property to decide which box appears on top. The green box has a z-index: 2, and the blue box has a z-index: 1. Since 2 is higher than 1, the green box appears above the blue box.

You can use any integer value for z-index, including positive numbers (like 5, 10, 100), zero, or even negative numbers (like -1) if you want something to stay behind other elements. The higher the value, the higher the layer.

Output:

The browser displays two overlapping boxes, a blue box in the background and a green box on top of it, showing the effect of layering using z-index as shown in figure 1.2.1.

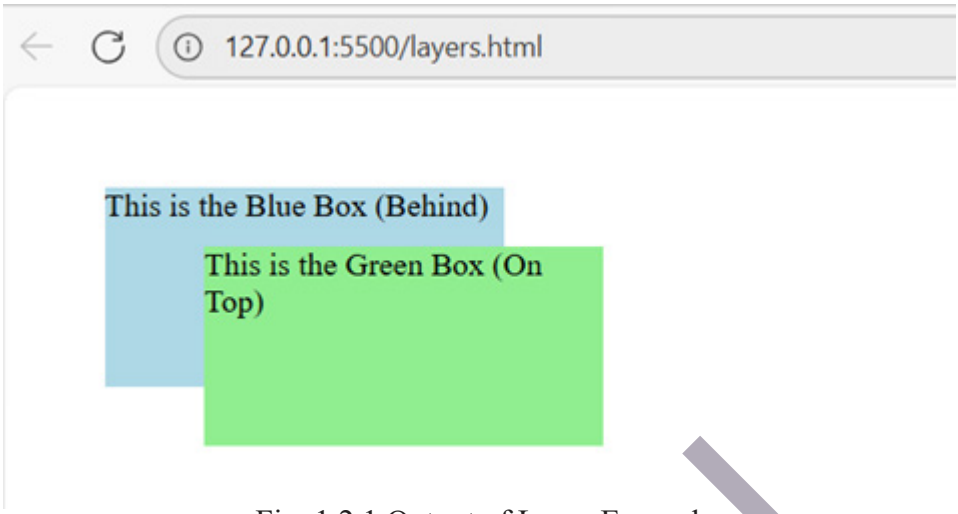


Fig. 1.2.1 Output of Layer Example

1.2.2 Image Maps in HTML

An Image Map is a feature in HTML that allows different parts of a single image to be made clickable. Normally, when you place an image on a webpage using the <img> tag, the entire image behaves as one object. But with an image map, you can divide that image into different shapes, and each shape can act as a separate clickable link. When a user clicks on a specific area of the image, they are taken to a different web page based on where they clicked.

Let's take a simple example to understand this better. Suppose you have a world map image on your website. Using an image map, you can define specific regions on the map like Europe, Asia, and Africa. When someone clicks on Europe, they can be taken to a page that contains information about European countries. If they click on Asia, they can go to a different page that talks about Asia. This makes the image interactive and useful for guiding users through content visually.

This feature is extremely helpful in many real-world scenarios:

- In interactive maps, where users can click on different countries, cities, or regions to view more details.
- In diagrams or technical charts, clicking on a specific part (like a machine part or organ in a human body) can open more information.
- In floor plans of buildings, where clicking on different rooms or sections opens detailed info or photos.
- In product catalogs, where clicking on different parts of an image (like different clothes or accessories) leads to product pages.

Image maps are helpful because they make websites easier to use and more visually appealing. Instead of displaying many text links, you can use one image that users can click on to go to different pages. This makes the website look cleaner and more interactive.

To create an image map in HTML, you need three main tags:

1. The <img> tag is used to display the image.
2. The <map> tag is used to define the clickable areas on that image.
3. The<area>tagisusedinsidethemaptodefineeachclickableregion.Eachareahas:
    - A shape (like rectangle, circle, or polygon)
    - Coordinates to define where the shape is on the image
    - Alink(href)thattellsthebrowserwhichpagetoopenwhentheareaisclicked

The image and the map are connected by the usemap attribute inside the <img> tag, which points to the name of the map.

Syntax of Image Map:

```
<img src="image.jpg" usemap="#mapname">
<map name="mapname">
  <area shape="rect" coords="x1,y1,x2,y2" href="page1.html">
  <area shape="circle" coords="x,y,radius" href="page2.html">
  <area shape="poly" coords="x1,y1,x2,y2,x3,y3..." href="page3.html">
</map>
```

Where,

- <img> – displays the image on the page
- usemap="#mapname" – connects the image to the map (you can name it anything)
- <map> – contains all clickable areas
- <area> – defines one clickable shape:
    - shape can be rect (rectangle), circle, or poly (polygon)
    - coords are numbers to tell the browser where on the image the clickable area is
    - href is the page the user goes to when clicking that area

Let's Consider One Example of Image Map in HTML:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>World Map Image Map</title>
</head>
<body>
  <h1>Click a Region on the Map</h1>
    <img src="worldmap.jpg" usemap="#worldmap" alt="World Map" width="800" height="400">
  <map name="worldmap">
   <area shape="rect" coords="100,100,200,200" href="northamerica.html" alt="North America">
          <area shape="circle" coords="600,150,50" href="europe.html" alt="Europe">
  <area shape="poly" coords="580,230,590,240,600,245,610,240,600,230" href="india.html" alt="India">
  </map>
</body>
</html>
```

This HTML program demonstrates the use of an image map, a feature in HTML that allows different parts of a single image to be made clickable. In this example, a world map image is displayed on a web page using the <img> tag, and several interactive regions (North America, Europe, Africa, and India) are defined using the <map> and <area> tags.

Each clickable region is assigned a specific shape:

- Rectangle for North America
- Circle for Europe

- Polygon for and India

The coords attribute defines the exact position and size of each clickable area based on pixel values. When the user clicks on a particular region, they are redirected to a new HTML page with information about that region.

This program helps demonstrate how HTML image maps can be used to create interactive diagrams, geographic maps, or navigation interfaces using a single image, making the website more visually intuitive and user-friendly.

The figure 1.2.2 given below shows the world map image used in the HTML program. Various regions such as North America, Europe,, and India are marked as clickable areas using the <area> tag inside a <map>. Each of these areas is linked to a separate HTML page that displays content specific to the selected region.

The image is mapped using pixel-based coordinates, and the clickable areas are defined using different shapes such as rectangle, circle, and polygon. This allows users to interact with different parts of the image and navigate accordingly.

Fig. 1.2.2 World map image

The following HTML file is opened when the user clicks on the India region in the image map defined in the main program. This page simply displays a heading and some text related to India, along with a link to go back to the world map.

india.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>India</title>
</head>
<body>
  <h1>This is India</h1>
  <p>You clicked on the India region of the map.</p>
  <a href="index.html">← Back to World Map</a>
</body>
</html>
```

When the index.html file is executed in a web browser, it displays a world map image along with a heading that says “Click a Region on the Map” as shown in figure 1.2.3. The image is interactive and contains multiple clickable regions, defined using the HTML <map> and <area> tags.

Fig. 1.2.3 Output of the Main Page Showing Clickable World Map

Fig. 1.2.4 User clicks on india

The figure 1.2.5 below shows the output displayed when the user clicks on the India region of the image map. As defined in the index.html file, clicking on India navigates to the india.html page. This page displays a heading and a short message, confirming that the user has selected India from the map.

Fig. 1.2.5 Output When Clicking on India

Recap

- HTML Layers: Placing elements on top of or behind others.
- Old <layer> tag: Used in old Netscape browsers, now not supported.
- Using and CSS: Control position with position, top, left, and z-index.
- CSS properties for layers:
  - position (static, relative, absolute, fixed, sticky)
  - top and left (position on page)
  - z-index (stacking order; higher value = on top)
- Image Maps in HTML: Make parts of an image clickable.
- Uses of image maps: Interactive maps, diagrams, floor plans, product catalogs.
- HTML tags for image maps: <img> (image), <map> (define areas), <area> (define shape, position, link).
- Shapes in image maps: Rectangle (rect), Circle (circle), Polygon (poly).
- Connecting image to map: usemap="#mapname" in <img>.
- Interactive result: Clicking a region opens the related page.

Objective Type Questions

1. Which CSS property controls the stacking order of elements?
2. Which HTML tag is used to create layers today?
3. Which CSS property sets the vertical position of an element?
4. Which CSS property sets the horizontal position of an element?
5. Which CSS property defines how an element is positioned on a page?
6. Which value of position keeps an element fixed when scrolling?
7. Which value of position allows free placement anywhere on the page?
8. Which HTML tag displays an image on a webpage?
9. Which HTML tag defines clickable areas on an image?

10. Which HTML tag contains all clickable areas of an image map?

11. Which attribute connects an image to a map?

12. Which shape attribute in <area> defines a rectangle?

13. Which shape attribute in <area> defines a circle?

14. Which shape attribute in <area> defines a polygon?

Answers to Objective Type Questions

1. z-index
2. div
3. top
4. left
5. position
6. fixed
7. absolute
8. <img>
9. <area>
10. <map>
11. <usemap>
12. rect
13. circle
14. poly

Assignments

1. Explain the concept of layers in HTML and why they are used.
2. Describe the purpose of the z-index property in CSS.
3. List and explain the different values of the CSS position property.
4. What is an image map in HTML? Explain its uses.
5. Name and explain the HTML tags used to create an image map.
6. Describe the different shapes that can be used in an image map and their purposes.
7. Explain how the usemap attribute connects an image to a map in HTML.

Reference

1. Duckett, J. (2011). *HTML and CSS: Design and build websites*. Wiley.
2. Robbins, J. N. (2018). *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics* (7th ed.). O'Reilly Media.
3. Powell, T. (2010). *HTML & CSS: The complete reference.* McGraw-Hill.

Suggested Reading

1. Keith, J. (2010). *HTML5 for web designers.* A Book Apart.
2. Freeman, E., & Freeman, E. (2012). *Head First HTML and CSS: A learner's guide to creating standards-based web pages*. O'Reilly Media.

Frames, Audio and Embed in HTML

Learning Outcomes

After completing this unit, learners will be able to:

- explain the purpose and use of frames, audio, and embed tags in HTML
- identify the main attributes of <frameset>, <frame>, <audio>, and <embed> tags
- create a simple webpage using frames to display multiple sections at once
- add audio and multimedia content to a webpage using the <audio> and <embed> elements
- compare the advantages and disadvantages of using frames, audio, and embed in modern web design

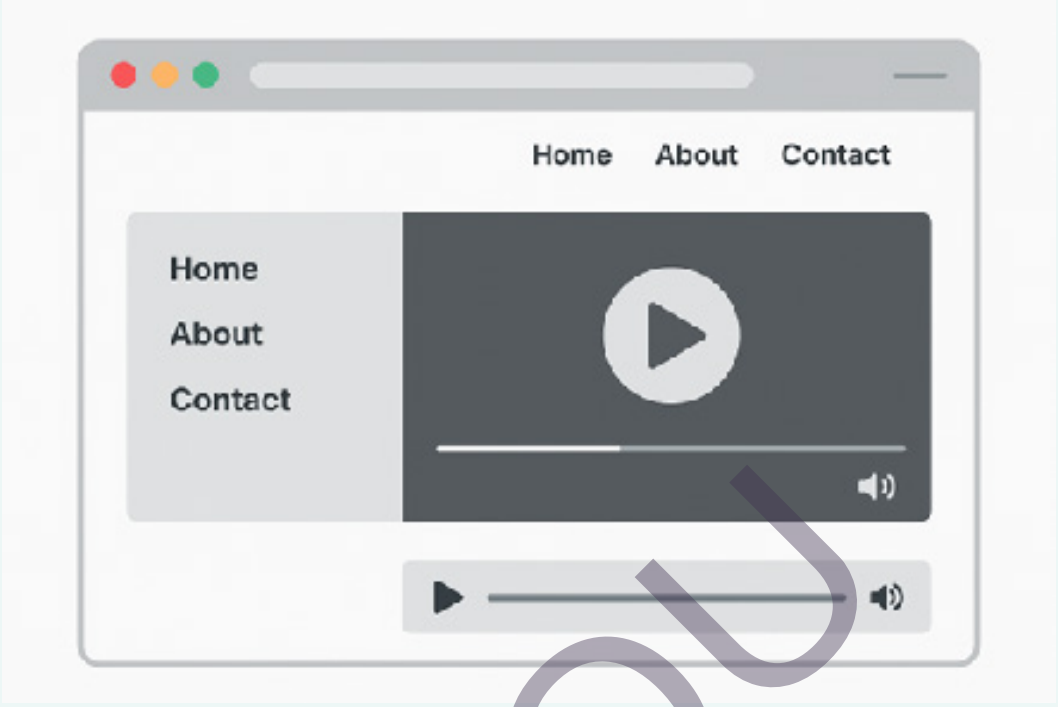
Prerequisites

Imagine visiting a news website where the top section displays live updates, the left panel shows quick navigation links, and the main area changes as you click without refreshing the whole page. In another example, when a learning website plays background music or embeds a video lesson directly on the screen, it uses frames, audio, and embed elements in HTML. These features make websites lively, organized, and interactive, providing users with a more engaging browsing experience.

The web development, as modern websites depend greatly on multimedia features and well-structured layouts. Before beginning this topic, learners should have a basic understanding of HTML tags, webpage structure, and common attributes. Learning how to divide a webpage, insert multimedia elements, and organize content effectively helps in creating professional and user-friendly websites. It also bridges traditional web design methods, such as frames, with modern techniques like iframes and embedded media, enabling learners to gain a complete understanding of the evolution and improvement of web design practices.

This unit is important for learners because it helps them learn how to design webpages that are more attractive and interactive. Studying frames, audio, and embed elements improves creative thinking, builds technical skills, and enables students to present

information in appealing ways. It also helps them understand how multimedia and structured layouts can enhance the usability and visual quality of modern websites.



Keywords

Frames, embed, src, Attributes, iframe, noframes, Frameset, Audio tag, Height and Width, HTML, CSS

Discussion

1.3.1 Introduction

In the early days of the World Wide Web, web pages were mostly static, simple collections of text and hyperlinks with very little multimedia or layout control. As the web evolved, designers and developers sought more dynamic and interactive ways to present content. HTML, the backbone of every webpage, began to include new tags and attributes that allowed richer presentation, such as frames for dividing the browser window into multiple sections, audio elements for embedding sound, and the <embed> tag for integrating external media and interactive applications. These elements revolutionized how users experience websites, transforming them from static text-based documents into immersive multimedia environments.

The use of frames in HTML allowed developers to split a single browser window into multiple independent sections, each capable of displaying different HTML documents simultaneously. This approach helped in creating persistent menus, navigation panels,

or advertisements that remained static while the main content changed. Although modern web design has largely moved toward CSS layouts and responsive designs, the concept of dividing content into manageable sections remains influential in web development. Understanding frames helps students appreciate the historical evolution and structure of complex page layouts.

On the other hand, the inclusion of audio and embed capabilities in HTML brought a new dimension to web interactivity. The <audio> tag made it possible to include background music, voice recordings, and sound effects directly within web pages without requiring external plugins. Similarly, the <embed> tag provided a versatile way to insert multimedia elements such as videos, animations, PDFs, and other interactive content from external sources. Together, these features transformed the web into a platform capable of delivering rich multimedia experiences, which are now an integral part of modern web design.

1.3.2 Frames in HTML

Frames in HTML were introduced to allow a single browser window to be divided into multiple, independent sections each displaying a different document. This concept enabled web designers to create pages with static elements such as navigation menus, headers, or advertisements that remain visible while other sections of the page change dynamically. For example, a website could have a frame at the top displaying a logo and navigation links, while the lower frame shows content that updates as users click on different links. This made browsing more seamless and improved user navigation in the early web era.

Frames in HTML are created using the <frameset> tag, which is used in place of the <body> tag. The <frameset> element is responsible for defining how the web browser window will be divided into multiple sections, either vertically into columns or horizontally into rows. Inside this <frameset> element, separate <frame> tags are included to specify the individual HTML documents that will appear in each section. Every frame functions as an independent window within the main browser window and is capable of loading a completely separate web page. This means that different parts of a single screen can display different content at the same time.

The introduction of frames made it easier for web designers to create pages with static areas, such as navigation menus, advertisements, or headers, that remain visible while other parts of the screen change according to user interaction. However, despite their usefulness, frames had several drawbacks that limited their long-term adoption. For instance, web pages using frames were difficult to bookmark correctly, as the browser could not save the state of individual frame content. In addition, printing framed web pages often produced disorganized results because each frame was treated as a separate document. Search engines also faced difficulties in indexing frame-based websites since each frame represented a distinct file rather than a single, unified page.

With the advancement of web standards, modern layout techniques such as cascading style sheets (CSS), responsive web design, and the introduction of the <iframe> tag gradually replaced traditional frames. These newer technologies offered greater

flexibility, better accessibility, and improved compatibility across devices. As a result, frames are now considered an outdated feature and are no longer supported in HTML5. Nevertheless, understanding how frames worked remains important for students, as it helps them appreciate the historical development of web layout techniques and the evolution of web design practices.

1.3.2.1 Basic Syntax of Frames in HTML

The general syntax of frames in HTML is as follows:

```
<frameset cols="50%,50%">
    <frame src="page1.html">
    <frame src="page2.html">
</frameset>
```

In this example, the browser window is divided into two vertical frames, each occupying 50% of the width. The src attribute of each <frame> tag specifies the HTML file that will appear within that section.

Example: Creating a Web Page with Frames

```
<!DOCTYPE html>
<html>
<head>
    <title>Frame Example</title>
</head>
<frameset cols="30%,70%">
    <frame src="menu.html" name="menuFrame">
    <frame src="content.html" name="contentFrame">
</frameset>
</html>
```

menu.html might contain navigation links such as:

```
<a href="home.html" target="contentFrame">Home</a><br>
<a href="about.html" target="contentFrame">About</a><br>
<a href="contact.html" target="contentFrame">Contact</a>
```

content.html would display the corresponding page content when links are clicked.

1.3.2.2 Attributes of Frames

Table 1.3.1 Important attributes used in <frameset> and <frame> tags.

Attribute	Description
cols	Specifies the number and size of columns in a frameset. Example: cols="30%,70%"
rows	Specifies the number and size of rows in a frameset. Example: rows="50%,50%"
src	Defines the source (URL or file name) of the document to be displayed in the frame.
name	Assigns a name to the frame, used for targeting links or scripts.
scrolling	Controls the appearance of scroll bars: values can be yes, no, or auto.
noresize	Prevents the user from resizing the frame.
frameborder	Defines whether the frame should have a border (1) or not (0).
bordercolor	Specifies the color of the frame border (not supported in HTML5).
marginwidth and marginheight	Set the margins inside the frame in pixels.

1.3.2.3 Nested Frames

Frames can also be nested, meaning that one frame can contain another set of frames. For example, a top frame might contain two columns that display different sections.

```
<frameset rows="25%,75%">
   <frame src="header.html">
   <frameset cols="30%,70%">
      <frame src="menu.html">
      <frame src="content.html">
   </frameset>
</frameset>
```

This divides the screen into two rows — the top for the header and the bottom divided into two columns for navigation and content.

1.3.2.4 The <noframes> Tag

Since not all browsers originally supported frames, HTML provided the <noframes> tag to define fallback content. Any text or message placed within this tag is displayed to users whose browsers cannot render frames.

```
<noframes>
   <body>
      Your browser does not support frames.
   </body>
</noframes>
```

1.3.2.5 Advantages and Disadvantages of Frames

Advantages:

- Allows persistent navigation menus or banners.
- Reduces bandwidth by loading only part of the page when content changes.
- Enables independent scrolling within different sections of the page.

Disadvantages:

- Difficult to bookmark or share specific frame content.
- Poor support for printing and search engine optimization.
- Complex maintenance and navigation synchronization.
- Deprecated in HTML5 (replaced by CSS and <iframe>).

1.3.2.6 The <iframe> Tag

The <iframe> element, short for *inline frame*, is a modern replacement for frames. It allows a separate HTML document to be embedded within the main page without using a <frameset>. Unlike traditional frames, an <iframe> exists inside the normal flow of the document and is supported by all modern browsers and HTML5 standards.

An iframe is often used to embed external content such as YouTube videos, Google Maps, advertisements, or other web pages within a site.

<iframe src="about.html" width="600" height="400"></iframe>

This code displays the content of about.html inside a 600x400 pixel frame on the webpage.

Properties of <iframe>:

1. **src** : Specifies the URL of the page to display within the iframe.
2. **width** and **height** : Define the size of the iframe in pixels.
3. **name** : Assigns a name to the iframe, useful for targeting links.
4. **frameborder** : Determines whether a border is displayed around the iframe.
5. **allowfullscreen** : Enables full-screen mode for embedded videos.
6. **sandbox** : Restricts what actions the iframe can perform (enhances security).
7. **loading="lazy"** : Delays loading until the iframe is visible, improving performance.

Example:

<iframe width="560" height="315"

src="https://www.youtube.com/embed/tgbNymZ7vqY"

allowfullscreen>

</iframe>

1.3.3 Audio in HTML

In the early web, adding sound to websites required external plugins like Flash, QuickTime, or Windows Media Player. This made web pages dependent on third-party software, often leading to compatibility issues. With the introduction of HTML5, developers gained the ability to embed audio directly into web pages using the <audio> tag. This tag allows web designers to include music, narration, or sound effects without relying on any external plugin, making websites more interactive and accessible across all modern browsers.

The <audio> element is versatile. It supports multiple audio formats, including MP3, OGG, and WAV, ensuring compatibility with different browsers. Developers can control playback using built-in controls, autoplay options, loops, and even integrate audio playback with JavaScript to create dynamic web experiences. Audio can be used in many applications, from simple background music on a website to interactive learning tools, podcasts, online games, and notifications.

Using audio effectively improves the user experience, making websites more engaging and interactive. For example, e-learning platforms use narration for lessons, online stores use sound effects for notifications, and blogs or news sites may include interviews or podcasts. Understanding how to implement the <audio> tag is essential for modern web development, as it provides the foundation for multimedia-rich websites without relying on external software.

1.3.3.1 Basic Syntax of Audio

The <audio> tag is simple to implement. The most basic form is:

```
<audio src="audiofile.mp3" controls>
    Your browser does not support the audio element.
</audio>
```

- src specifies the path to the audio file.
- controls adds a built-in player with play, pause, and volume options.
- The text inside the <audio> tag provides a fallback message for browsers that do not support HTML5 audio.

Example: Multiple Source Formats

To ensure that your audio works across all browsers, you can provide multiple source formats:

```
<audio controls>
  <source src="song.mp3" type="audio/mpeg">
  <source src="song.ogg" type="audio/ogg">
  Your browser does not support the audio element.
</audio>
```

- The browser will try to play the first compatible audio file it recognizes.
- type specifies the MIME type of the audio file.

1.3.3.2 Attributes of the <audio> Tag

Table 1.3.2 Attributes of the < audio > Tag

Attribute	Description
src	Path to the audio file (MP3, OGG, WAV, etc.)
controls	Adds a visible audio player with play, pause, and volume controls
autoplay	Automatically plays the audio when the page loads

loop	Repeats the audio indefinitely
muted	Starts the audio in a muted state
preload	Specifies if the audio should be loaded when the page loads. Values: auto, metadata, none

1.3.3.3 Advantages of Using HTML Audio

- No external plugins are required.
- Works across all modern browsers.
- Supports multiple audio formats for maximum compatibility.
- Can be controlled with HTML attributes or JavaScript for dynamic interaction.
- Enhances user engagement on websites, making them more interactive and professional.

1.3.4 The <embed> Tag in HTML

The <embed> tag in HTML allows developers to integrate external content or applications directly into a web page. This can include multimedia files such as videos, audio, animations, or even documents like PDFs. Unlike <iframe>, which embeds an entire HTML page, <embed> is primarily used to include content that is not inherently HTML, such as Flash objects, PDFs, or media files.

Initially, <embed> was not part of the official HTML specification, but due to its widespread use, it became standardized in HTML5. One of the main advantages of the <embed> tag is its simplicity. By specifying the source file and its type, developers can include complex content with minimal coding effort. It is especially useful for displaying content that would otherwise require a plugin or external application, while still keeping it integrated within the webpage layout.

1.3.4.1 Basic Syntax of <embed>

<embed src="filename" type="media/type" width="width" height="height">

src → Path to the file you want to embed.

type → MIME type of the content (e.g., application/pdf, audio/mpeg, video/mp4).

width and height → Size of the embedded object on the web page.

Example1: Embedding a PDF Document

<!DOCTYPE html>

```
<html>
<head>
   <title>Embed PDF Example</title>
</head>
<body>
   <h2>View PDF Document</h2>
    <embed src="document.pdf" type="application/pdf" width="700" height="500">
</body>
</html>
```

- This code embeds a PDF file directly into the webpage.
- Users can scroll through the PDF without leaving the page.
- The width and height define the visible area of the embedded document.

Example2: Embedding a Video File

```
<!DOCTYPE html>
<html>
<head>
   <title>Embed Video Example</title>
</head>
<body>
   <h2>Watch Video</h2>
<embed src="video.mp4" type="video/mp4" width="600" height="400">
</body>
</html>
```

- This embeds a video directly into the page.
- Users can view the video without opening it in a separate player.
- The type attribute ensures the browser recognizes the file format.

1.3.4.2 Attributes of <embed> Tag

Table 1.3.3 Attributes of <embed> Tag

Attribute	Description
src	Specifies the URL or path of the external file to embed
type	Defines the MIME type of the embedded content
width	Sets the width of the embedded object (pixels or %)
height	Sets the height of the embedded object (pixels or %)
autostart	If supported, determines whether media starts automatically
loop	If supported, allows the media to play repeatedly
align	Aligns the embedded content relative to surrounding elements

1.3.4.3 Advantages of Using <embed>

- Simple and easy to use for embedding non-HTML content.
- Reduces the need for external plugins or applications.
- Allows integration of multimedia files, PDFs, and interactive objects directly into a webpage.
- Enhances user experience by keeping all content within a single page.
- Works across modern browsers and supports various MIME types.

Recap

- HTML evolved from static text pages to interactive multimedia websites.
- Frames, audio, and embed elements allow richer web content.
- Frames divide the browser window into multiple independent sections.
- Frames are created using the <frameset> and <frame> tags.
- Each frame can display a different HTML document simultaneously.
- Frames are useful for creating persistent menus, headers, or advertisements.
- The <frameset> tag replaces the <body> tag to define rows or columns.
- Frames have drawbacks: difficult bookmarking, poor printing, and SEO issues.
- Frames are now deprecated in HTML5, replaced by <iframe> and CSS layouts.

- Nested frames allow complex layouts by combining rows and columns.
- The <noframes> tag provides fallback content for unsupported browsers.
- The <iframe> tag embeds a separate HTML page inline in a webpage.
- <iframe> supports modern features like sandbox, allowfullscreen, and lazy loading.
- Audio in HTML is added using the <audio> tag, introduced in HTML5.
- The <audio> tag supports multiple formats: MP3, OGG, and WAV.
- Audio can be controlled using controls, autoplay, loop, muted, and preload attributes.
- The text inside <audio> provides a fallback message for unsupported browsers.
- JavaScript can control audio dynamically using play(), pause(), and volume.
- Audio improves user engagement on websites, e.g., e-learning, podcasts, notifications.
- The <embed> tag allows embedding external content like PDFs, videos, audio, or animations.
- <embed> is simple, specifying src, type, width, and height.
- <embed> can autoplay or loop media if supported.
- It enhances user experience by keeping multimedia integrated in a webpage.
- <embed> is widely supported across modern browsers, though <audio> and <video> are preferred for HTML5 media.

Objective Type Questions

1. What tag is used to divide a browser window into multiple sections?
2. Where do you place the <frameset> tag in an HTML document?
3. How can a frame be prevented from being resized by the user?
4. Which attribute of <frameset> specifies the number and size of rows?
5. Why was the <noframes> tag used in HTML?
6. What modern HTML tag replaced <frameset> for embedding pages?

7. When does the autoplay attribute in <audio> work?
8. Who controls audio dynamically using JavaScript?
9. What attribute adds the play/pause controls in <audio>?
10. Which audio formats are commonly supported in HTML5?
11. How can audio be repeated indefinitely on a webpage?
12. What tag is used to embed external content like PDFs or videos?
13. Where do you specify the MIME type for embedded content?
14. Why is <iframe> preferred over <frameset> in modern web design?
15. Which attribute in <embed> defines the width of the embedded object?
16. What attribute starts an audio file in a muted state?
17. How can you provide multiple audio sources for browser compatibility?
18. Which attribute of <iframe> delays loading until visible?
19. What problem did frames cause for bookmarking web pages?
20. Who benefits from using audio and embed tags on websites?

Answers to Objective Type Questions

1. <frameset>
2. In place of the <body> tag
3. noresize
4. rows
5. To provide fallback content for browsers that do not support frames
6. <iframe>
7. When the page loads
8. JavaScript (or the web developer using JS)
9. controls

10. MP3, OGG, WAV

11. loop

12. <embed>

13. type

14. Greater flexibility, accessibility, and HTML5 support

15. width

16. muted

17. Using multiple <source> tags inside <audio>

18. loading="lazy"

19. Users could not save or share the state of individual frame content

20. Website users, learners, and general audience

Assignments

1. Explain the difference between <frame> and <iframe> in HTML with examples.
2. Write a short note on the advantages and disadvantages of using frames in HTML.
3. Write an HTML code to embed an audio file on a webpage
4. Create a web page using <embed> to display a PDF of your choice.
5. Create an HTML page with fallback content for unsupported browsers: use <noframes> for frames and fallback text for <audio> or <embed>.

Reference

1. Freeman, A., & Robson, E. (2019). *Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages*. Sebastopol, CA: O'Reilly Media.

2. Robbins, J. N. (2018). *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics* (5th ed.). Sebastopol, CA: O'Reilly Media.

3. Morris, T. F. (2017). *Web Development and Design Foundations with HTML5* (8th ed.). Pearson.

4. Powers, S. (2011). *HTML5 Media*. O'Reilly Media.

Suggested Reading

1. https://www.w3.org/TR/html5/

2. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio

3. Bos, B., Çelik, T., Hickson, I., & Lie, H. W. (2012). *Cascading Style Sheets, level 2 revision 1 (CSS 2.1)*

4. https://www.w3schools.com/tags/tag_embed.asp

5. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/audio

HTML Forms

Learning Outcomes

After completing this unit, learners will be able to:

- explain the purpose of HTML forms and describe how different form elements are used to collect user input
- create simple web forms using various form controls and attributes such as text boxes, checkboxes, and buttons
- compare the GET and POST methods and choose the suitable one for different form submission needs
- evaluate the importance of form validation and accessibility in making web forms more user-friendly and inclusive
- design a complete and well-structured HTML form that includes validation and accessibility features

Prerequisites

Imagine you are trying to create an online movie ticket booking system. Users can see the list of available movies and showtimes, but without a way to enter their details, select seats, or confirm payment, the website cannot process their bookings. Similarly, on an e-commerce site, customers could browse products but would not be able to place orders, provide delivery information, or submit feedback. HTML forms are important because they form the foundation of user interaction on the web. Without forms, websites would only display information but could not collect any data from users.

Imagine you are applying for a job online. You fill in your personal details, upload your resume, and click the “Submit” button. The data you entered is collected by the website and sent to a server for processing, all through an HTML form.Similarly, when you log in to your social media account, order food online, or subscribe to a newsletter, you are interacting with forms. They silently perform the crucial task of collecting, validating, and transferring user information behind every web-based action. HTML forms are, therefore, an essential part of modern web development. They provide structure and control for gathering different types of data such as text, numbers, files, and selections.

Familiarize the topic the learners gain the ability to design interactive, responsive, and user-friendly web pages that communicate effectively with web servers. It also builds the groundwork for advanced areas like web programming, client-side scripting, and database connectivity.

Keywords

Input Controls, Text Field, Radio Button, Checkbox, Select Box, Form Validation, Client-Side Accessibility.

Discussion

1.4.1 Introduction to HTML Forms

HyperText Markup Language is the backbone of every webpage. While most HTML elements are used for displaying content such as text, images, and multimedia, HTML forms are special because they allow user interaction, collecting input, sending data, and performing actions. A form in HTML is a structured area on a webpage that allows users to enter data such as their name, email, password, or feedback. The data entered can then be sent to a web server for processing. For example, when you log into a website, sign up for a newsletter, or fill out a contact form, you are using HTML forms.

An HTML form is an essential element of web development that allows users to interact with a website by entering and submitting data to a server. It acts as a communication bridge between the user and the web application, enabling functions such as user registration, login, feedback submission, search queries, and online transactions. Defined using the <form> tag, an HTML form can contain various input elements like text fields, radio buttons, checkboxes, drop-down lists, file uploads, and buttons. The action attribute of the <form> tag specifies where (i.e., which server-side script or URL) the form data will be sent, while the method attribute defines how the data is transmitted, commonly using GET (appending data to the URL) or POST (sending data securely in the request body). Forms also support accessibility and validation features to ensure that users input correct and complete data.

1.4.2 The <form> Tag

The <form> tag in HTML defines a form section in a webpage. All input fields, labels, buttons, and other form elements are placed inside this tag.

Syntax:

```
<form>
form elements
</form>
```

1.4.2.1 Attributes of the <form> Tag

Table 1.4.1 Attributes of the <form> Tag

Attribute	Description	Example
action	Specifies where to send the form data after submission (the URL of the server-side script).	<form action="submit.php">
method	Defines how data is sent to the server — either GET or POST.	<form method="post">
enctype	Specifies how the form data should be encoded when submitting. Used especially for file uploads.	<form enctype="multipart/form-data">
name	Gives a name to the form, useful for JavaScript access.	<form name="loginForm">
target	Determines where to display the response after submitting the form (_blank, _self, _parent, _top).	<form target="_blank">
autocomplete	Enables or disables browser auto-fill.	<form autocomplete="on">

1.4.2.2 GET and POST Methods

When a form is submitted, the data entered by the user must be sent to a web server for processing. The way this data is sent is determined by the method attribute of the <form> tag. The two most common methods are GET and POST. The GET method appends the form data to the URL in the form of key-value pairs and sends it to the server. The POST method sends the form data inside the body of the HTTP request, not in the URL. This makes it more secure and suitable for sensitive information.

```
<form action="search.html" method="get">
  <label for="query">Search:</label>
 <input type="text" id="query" name="q" placeholder="Type something...">
  <input type="submit" value="Search">
</form>
```

Search: Type something... Search

```
<form action="register.html" method="post">
  <input type="text" name="username" placeholder="Enter Username">
      <input type="password" name="password" placeholder="Enter Password">
  <input type="submit" value="Register">
</form>
```

1.4.3 Common Form Controls

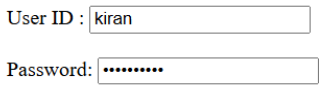
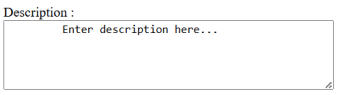
HTML provides a wide range of form controls (input elements) for different types of data entry. Each control has a specific tag and attributes that define how it behaves and looks. The <input> tag is the most versatile form element.

1.4.3.1 Text Input Controls

In HTML forms, controls used for capturing text from the user can be divided into three primary categories based on how the text is displayed and the intended security level.

Table 1.4.2 Types of text input controls

Types	Description	Examples	Output
Single-line Text Input Control	Used for short, general-purpose text entries, such as names, search terms, or single-line answers.	<!DOCTYPE html> <html> <head> <title>Text Input Control</title> </head> <body> <form > First name: <input type = "text" name = "first_name" /> Last name: <input type = "text" name = "last_name" /> </form> </body> </html>	First name: Last name:

Password Input Control	Designed for sensitive information like passwords. The key feature is that the characters entered by the user are masked for privacy.	< ! D O C T Y P E html> <html> <head> <title>Password Input Control</ title> </head> <body> <form > User ID : <input type = "text" name = "user_id" /> Password: <input type = "password" name = "password" /> </form> </body> </html>	
Multi-line Text Input Control	Used when the user needs to enter a larger amount of text, such as comments, descriptions, or messages. The input area is typically resizable.	< ! D O C T Y P E html> <html> <head> <title>Multiple-Line Input Control</title> </head> <body> <form> Description : <textarea rows = "5" cols = "50" name = "description"> Enter description here... </textarea> </form> </body> </html>	

1.4.3.2 Checkbox Control

The Checkbox Control allows users to select zero, one, or multiple options from a list of choices.

Example	Output
<!DOCTYPE html> <html> <head> <title>Checkbox Control</title> </head> <body> <form> <input type = "checkbox" name = "Single" value = "on"> Single <input type = "checkbox" name = "Double" value = "on"> Double <input type = "checkbox" name = "Mixed" value = "on"> Mixed </form> </body> </html>	☐ Single ☐ Double ☐ Mixed

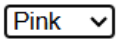
1.4.3.3 Radio Buttons Control

The Radio Buttons Control is used when you need the user to select exactly one option from a predetermined set of choices.

Example	Output
<!DOCTYPE html> <html> <head> <title>Radio Box Control</title> </head> <body> <form> <input type = "radio" name = "Male" value = "maths"> Male <input type = "radio" name = "Female" value = "physics"> Female </form> </body> </html>	○ Male ○ Female

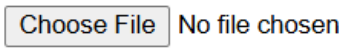
1.4.3.4 Select Box Control(Drop-Down List)

The Select Box control (created using the HTML <select> tag) displays a list of choices to the user in a compact, drop-down list format.

Example	Output
<!DOCTYPE html> <html> <head> <title>Select Box Control</title> </head> <body> <form> <select name = "dropdown"> <option value = "Pink" selected>Pink</option> <option value = "Yellow">Yellow</option> <option value = "Green">Green</option> </select> </form> </body> </html>	

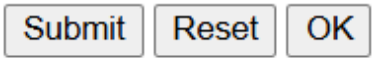
1.4.3.5 File Select Box Control (File Upload)

The File Select Box is a control that allows a user to upload a file from their local computer to be submitted along with the form data. It's often referred to as a File Upload control.

Example	Output
<!DOCTYPE html> <html> <head> <title>File Upload Box</title> </head> <body> <form> <input type = "file" name = "fileupload" accept = "image/*" /> </form> </body> </html>	

1.4.3.6 Button Control

A Button Control creates a standard clickable button element on a web page. Buttons are used to trigger an action, often executed by a script (like JavaScript), such as opening a dialog, calculating a result, or initiating a request, without automatically submitting the form.

Example	Output
<!DOCTYPE html> <html> <head> <title>File Upload Box</title> </head> <body> <form> <input type = "submit" name = "submit" value = "Submit" /> <input type = "reset" name = "reset" value = "Reset" /> <input type = "button" name = "ok" value = "OK" /> </form> </body> </html>	

1.4.3.7 Datetime controls

HTML provides several dedicated input types for handling date and time values

Example	Output
<!DOCTYPE html> <html> <body> <form action = "/cgi-bin/html5.cgi" method = "get"> Date and Time : <input type = "datetime" name = "newinput" /> <input type = "submit" value = "submit" /> </form> </body> </html>	

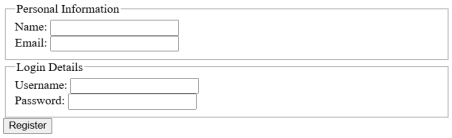
1.4.3.8 Email Control

The Email Control is a specialized single-line input field designed exclusively for collecting email addresses. It ensures the data submitted is in a standard email format.

Example	Output
<!DOCTYPE html> <html> <body> <form action = "/cgi-bin/html5.cgi" method = "get"> Enter email : <input type = "email" name = "newinput" /> <input type = "submit" value = "submit" /> </form> </body> </html>	

1.4.4 Fieldsets and Legends

When a form contains multiple sections, it is a good practice to group related form controls together using the <fieldset> element. The <legend> tag provides a caption for the fieldset, improving readability and accessibility.

Example	Output
<form> <fieldset> <legend>Personal Information</legend> Name: <input type="text" name="name"> Email: <input type="email" name="email"> </fieldset> <fieldset> <legend>Login Details</legend> Username: <input type="text" name="username"> Password: <input type="password" name="password"> </fieldset> <input type="submit" value="Register"> </form>	

1.4.5 Form Validation (Client-Side)

Client-side validation ensures that user inputs are checked in the browser before the data is submitted to the server. It improves user experience by providing instant feedback and reduces server load.

HTML5 Validation Attributes:

- required → Ensures the field is not empty.
- min and max → Define numerical limits.
- minlength and maxlength → Define text length limits.
- pattern → Defines a regular expression that input must match.
- type → Automatically validates based on input type (e.g., email, url, etc.).

Example	Output
<form> <input type="text" name="name" required> <input type="email" name="email" required> <input type="password" name="pwd" minlength="6" maxlength="12"> <input type="submit" value="Submit"> </form>	

1.4.6 Form Accessibility and Usability

Forms are one of the most commonly used elements on a website, they allow users to register, log in, provide feedback, make purchases, or share information. However, for a form to be truly effective, it must be both accessible and usable.

- Accessibility means that the form can be easily used by everyone, including people with disabilities such as visual, auditory, motor, or cognitive impairments.
- Usability means that the form is designed in a way that makes it simple, clear, and efficient for all users to complete.

1.4.6.1 Form Accessibility

Form accessibility refers to the practice of designing web forms that can be used effectively by people who rely on assistive technologies, such as screen readers, keyboard navigation, voice input tools, and switch devices.

For example, a visually impaired user depends on a screen reader to read out the labels and descriptions of form fields. If the form is not properly labeled or structured, that person will not understand what input is required. Some essential practices to make forms more accessible.

Use <label> Elements for Inputs	<label for="email">Email Address:</label> <input type="email" id="email" name="email" required> This ensures screen readers can correctly announce what each field represents.

Provide Group Labels with <fieldset> and <legend>	<fieldset> <legend>Gender</legend> <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> </fieldset> 
Ensure Keyboard Accessibility	Many users navigate forms using the Tab and Enter keys instead of a mouse. A well-designed form should: ♦ Allow users to move between fields using the Tab key. ♦ Highlight the active field clearly. ♦ Allow submission using the Enter key when appropriate.
Use Clear and Descriptive Placeholders	<input type="text" name="phone" placeholder="e.g., +91-9876543210"> 
Maintain Proper Color Contrast	Color contrast between text and background should be high enough for users with low vision to read easily.

1.4.6.2 Form Usability

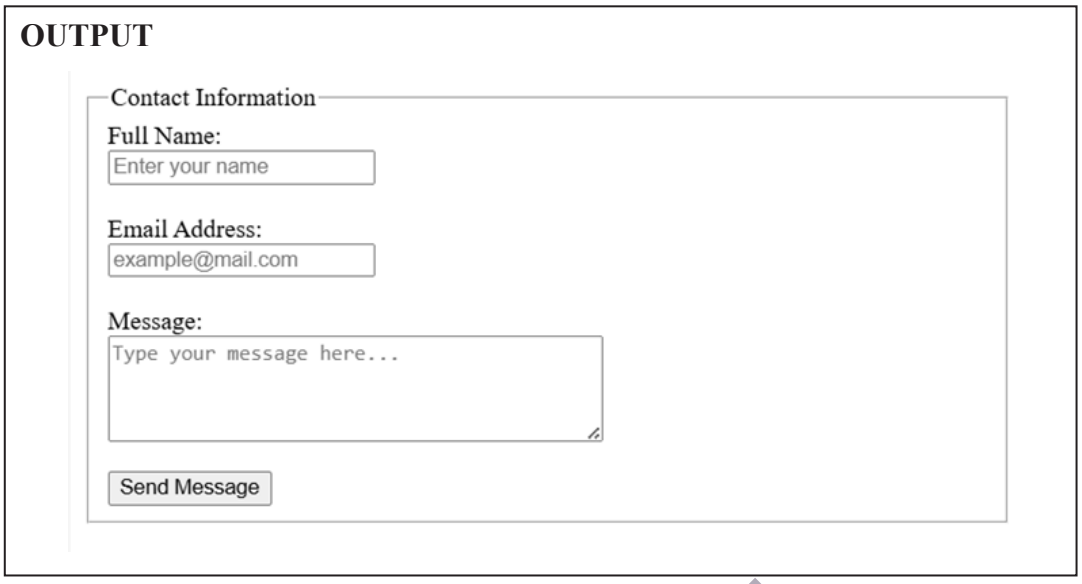
To make forms more user-friendly, designers should follow these usability guidelines.

a) Keep Forms Simple- Ask only for necessary information
b) Organize Information Logically
c) Provide Inline Validation- Show errors or confirmations as soon as users move away from a field. This helps them correct mistakes quickly.
d) Indicate Required Fields Clearly- Use an asterisk (*) or label text to show which fields are required.
e)Use Proper Field Sizes and Alignments-Field sizes should match expected input lengths. For example, a PIN field should be small, while an address field should be larger.

Example: Accessible and Usable Form:

```
<form>
  <fieldset>
    <legend>Contact Information</legend>
    <label for="name">Full Name:</label><br>
      <input type="text" id="name" name="name" placeholder="Enter your name" required><br><br>
    <label for="email">Email Address:</label><br>
     <input type="email" id="email" name="email" placeholder="example@mail.com" required><br><br>
    <label for="msg">Message:</label><br>
 <textarea id="msg" name="message" rows="4" cols="40" placeholder="Type your message here..." required></textarea><br><br>
    <input type="submit" value="Send Message">
  </fieldset>
</form>
```

OUTPUT



1.4.7 Form Submission and Handling

When the Submit button is pressed:

1. The browser gathers all form data as name-value pairs.
2. It sends the data to the location specified in the action attribute.
3. If no action is specified, the data is submitted to the same page.
4. The method determines how it's sent:
   - GET: Appends data to the URL (for simple searches or short data).
   - POST: Sends data securely in the body (used for login, registration, etc.).

Recap

- HTML forms allow user interaction by collecting input data like names, emails, passwords, and sending them to a server for processing.
- Common examples include login pages, registration forms, feedback forms, and search boxes on websites.
- All input fields, buttons, and controls are placed inside the <form> tag, which defines the beginning and end of a form.
- Form Attributes:

  action: Defines where the form data is sent

  method: Defines how the data is sent (GET or POST).

enctype: Defines encoding type (used for file uploads).

target: Specifies where to open the response page.

autocomplete: Enables or disables browser autofill.

- GET Method: Sends data appended to the URL; suitable for simple searches and visible data.
- POST Method: Sends data securely in the request body; suitable for sensitive information like passwords or registration forms.
- HTML provides various form elements for user input such as text boxes, checkboxes, radio buttons, dropdowns, file uploads, and buttons.
- Text Input Controls:
  - Single-line: <input type="text"> for names or short text.
  - Password field: <input type="password"> masks characters.
  - Multi-line: <textarea> for long messages or comments.
- Checkbox Control: Allows selecting multiple options (e.g., hobbies, interests).
- Radio Button Control: Used to select only one option from a group (e.g., gender selection).
- Select Box (Dropdown): <select> provides a compact list of options for easy selection.
- File Upload Control: <input type="file"> lets users upload images, documents, or other files.
- Button Controls:
  - Submit: Sends data to server.
  - Reset: Clears all fields.
  - Button: Triggers custom actions via scripts.
- Form Validation: HTML5 provides attributes like required, min, max, pattern, and type to check user input before submission (client-side validation).
- Accessibility & Usability: Use <label>, <fieldset>, and <legend> for clarity.
  - Ensure keyboard navigation works.
  - Keep forms simple, organized, and visually clear for all users, including those with disabilities.

Objective Type Questions

1. What is the purpose of using the <form> element in HTML?
2. Which attribute of the <form> tag specifies the URL to send form data?
3. Which two methods are most commonly used to send data to the server?
4. What tag is used to create a single-line text input field?
5. Which tag is used to create a multi-line text area for long messages?
6. Which HTML tag is used to create a drop-down list in a form?
7. What input type is used to enter hidden or masked characters like passwords?
8. Which attribute defines how the form data is encoded during submission?
9. What type of form control allows users to select multiple options?
10. Which form control allows the user to select only one option from a group?
11. What input type allows users to upload files from their local system?
12. Which attribute ensures that a form field cannot be left empty?
13. Which tag is used to group related form controls together?
14. What tag provides a title or heading for a group of related fields?
15. In which part of the web process does client-side validation take place?

Answers to Objective Type Questions

1. To collect user input
2. action
3. GET and POST
4. input
5. textarea
6. select
7. password

8. enctype
9. checkbox
10. radio
11. file
12. required
13. fieldset
14. legend
15. browser

Assignments

1. Write an HTML program to create a simple login form with username and password fields and a submit button.
2. Design a registration form that includes text fields for name, email, password, gender (radio buttons), and hobbies (checkboxes).
3. Create an HTML form that allows a user to upload a profile picture.
4. Write an HTML program using the <select> tag to display a list of courses (e.g., BCA, MCA, MSc, MBA).
5. Design a feedback form that uses a <textarea> element for user comments and a submit button.
6. Create an HTML form using both GET and POST methods and observe the difference in the browser's address bar.
7. Write an HTML code that uses the required attribute to make certain fields mandatory.
8. Develop a form that collects a user's date of birth using the <input type="date"> control.
9. Design a form that includes a <fieldset> and <legend> to group personal and login information separately.
10. Create a contact form that includes name, email, phone number, and a message box with form validation.

Reference

1. Packt Publishing. (2025). *Responsive Web Design with HTML5 and CSS* (5th ed.). Packt Publishing.

2. Ruvalcaba, Z., Boehm, A., & Delamater, M. (2024). *Murach's HTML and CSS* (6th ed.). Mike Murach & Associates Inc.

3. Włodarczyk, A. (2023). *HTML and CSS for Beginners: The First Step to Your Coding Career*. Packt Publishing.

4. Frain, B. (2022). *Responsive Web Design with HTML5 and CSS* (4th ed.). Packt Publishing

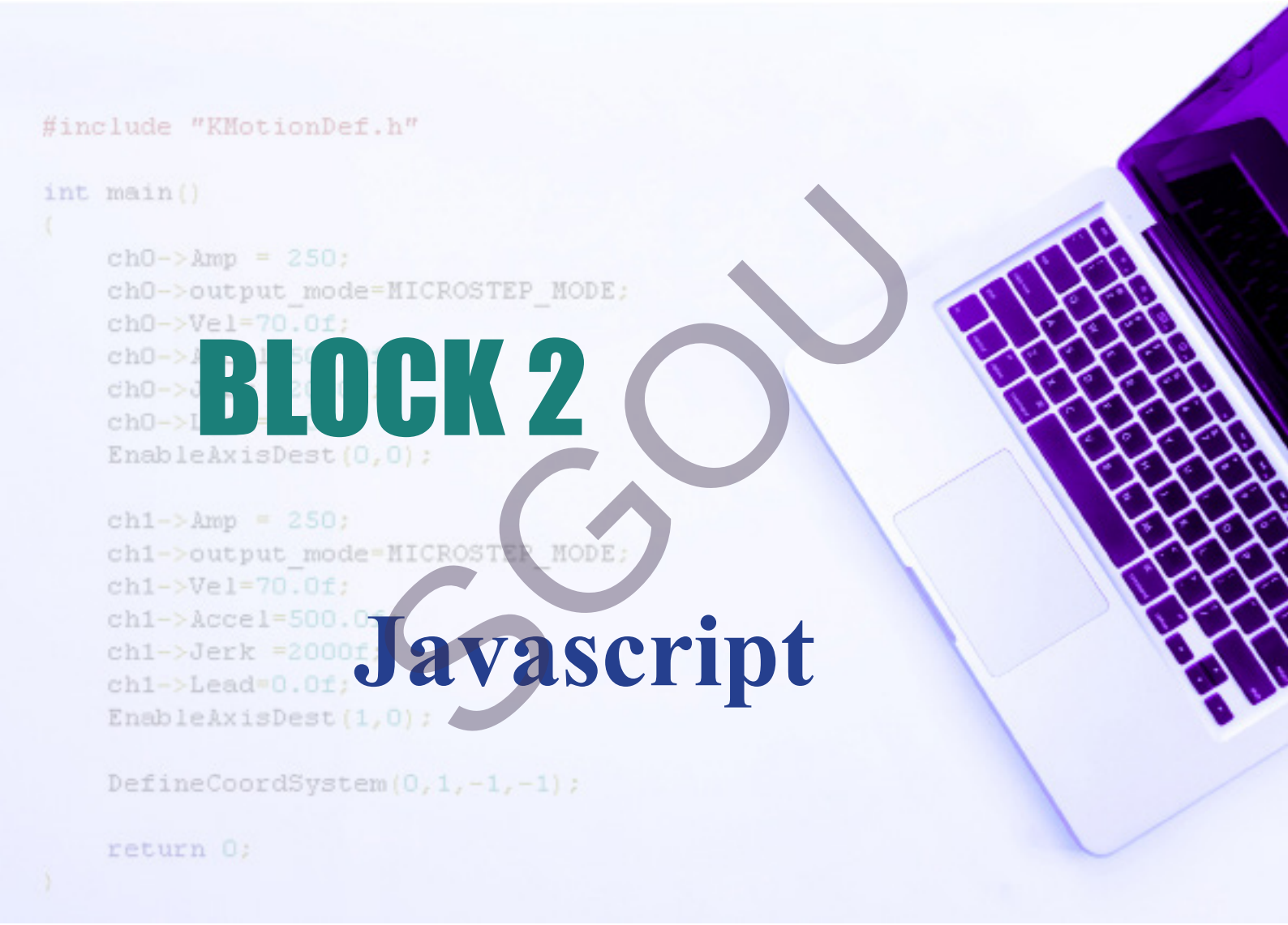
Suggested Reading

1. Tittel, E., & Minnick, C. (2013). *HTML5 for dummies.* John Wiley & Sons.

2. Duckett, J. (2011). *HTML and CSS: Design and build websites.* John Wiley & Sons.

3. W3Schools. (2024). *HTML Forms tutorial.* Retrieved from https://www.w3schools.com/html/html_forms.asp

4. Mozilla Developer Network (MDN). (2024). *HTML forms guide.* Retrieved from https://developer.mozilla.org/en-US/docs/Learn/Forms



BLOCK 2

Javascript

Introduction to JavaScript

Learning Outcomes

After completing this unit, learners will be able to:

- recall the basic features of JavaScript
- identify the different ways JavaScript can be included in an HTML document
- recognize the components of the client-server model
- list the steps involved in how a browser interacts with a server
- familiarise the key functions of the JavaScript engine

Prerequisites

Imagine you are browsing an online shopping website. You find a product you like-a pair of shoes - and click the "Add to Cart" button. Instantly, the item count in your cart updates from "0" to "1," all without the page reloading. Later, you scroll down to sign up for the newsletter. As you begin typing your email address into the form, a message pops up in real time, warning you for example,if it is missing the "@" symbol. Everything feels fast, responsive, and smooth.

Have you ever wondered how websites manage to do all this without making you wait or refresh the page? The answer is JavaScript. JavaScript is a powerful and lightweight scripting language that brings interactivity and dynamic behavior to otherwise static web pages. While HTML provides the structure of a webpage and CSS handles its appearance, JavaScript is what makes the page respond to user actions and inputs in real time.

Understanding JavaScript means gaining control over how web pages behave and respond. It is the tool that empowers developers to go beyond static content and create interactive experiences that users expect from today's web applications.

Keywords

Client-Server Model, Dynamic Typing, Inline JavaScript, Internal JavaScript, External JavaScript

Discussion

2.1.1 Introduction to Javascript

JavaScript is a versatile, and dynamically typed programming language that plays a vital role in modern web development. It brings life to static web pages by enabling interactivity and dynamic behavior. JavaScript is widely used for building interactive web applications,JavaScript supports both client-side and server-side development. It integrates seamlessly with HTML and CSS, allowing developers to create rich, responsive and engaging user interfaces.

As a lightweight and interpreted language, JavaScript is commonly embedded in web pages, enabling client-side scripts to interact with users and respond to their actions in real time. Its object-oriented features and flexible syntax make it a powerful and adaptable tool for a wide range of use cases, from simple animations to complex,full-scale web systems.

> JavaScript was originally introduced under the name LiveScript by Netscape. However, due to the popularity and hype around Java at the time, Netscape decided to rename it to JavaScript.

2.1.1.1 Features of Javascript

JavaScript offers a wide range of features that make it powerful and adaptable for web development. Some of its key characteristics include:

1. Lightweight

JavaScript is designed to be a lightweight language, optimized for handling data and functionality on the client side. It operates with minimal runtime overhead and requires relatively few system resources, making it efficient for web-based applications.

2. Dynamic Typing

One of the defining features of JavaScript is its support for dynamic typing. This means that the type of a variable is determined by the value it holds, rather than being explicitly declared. Developers can assign different types of data to the same variable without the need for prior type specification.

3. Object-Oriented Programming

JavaScript supports object-oriented programming (OOP) by allowing the creation and manipulation of objects that contain properties and methods. This enables developers to build complex data models and implement algorithms essential for creating modern, interactive web applications.

4. Prototype-Based Inheritance

Unlike class-based languages, JavaScript uses prototype-based inheritance. Instead of defining classes, developers can create object prototypes, which can then be used to generate new objects that inherit properties and behaviors from the original prototype.

5. Interpreted Language

JavaScript is an interpreted language, meaning its code is executed line by line at runtime, rather than being compiled beforehand. Every modern web browser includes a built-in JavaScript interpreter, which processes the code directly, allowing for faster development and testing without a separate compilation step.

2.1.2 Client-Server Model of Javascript

The Client-Server Model is a type of network architecture in which two main components are the client and the server, communicate over a network to exchange data and perform tasks.

In this model, the client is typically a user-facing application that collects input or requests from the user. It then sends these requests to the server, which is responsible for processing the requests, performing any necessary operations (like database queries or computations), and returning the appropriate response back to the client as in Fig 2.1.1.

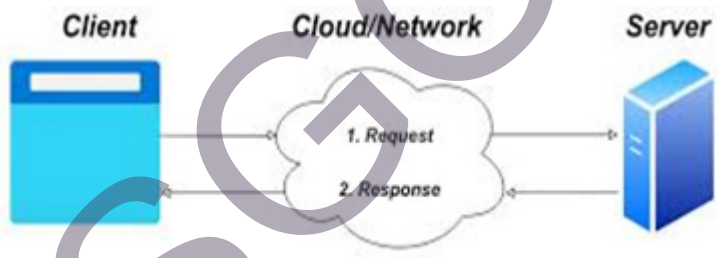


Fig. 2.1.1 Client Server model

This model is distributed, meaning the client and server typically operate on separate systems connected through a network, such as the internet or a local area network (LAN).

1. Node

A node refers to any device or endpoint connected to a computer network. Devices such as laptops, PDAs, internet-enabled mobile phones, and similar components are all considered nodes within a network. These nodes communicate with one another to share data and resources.

2. Client

A client is a node (a computer or device) that connects to a server within a network. It is responsible for collecting input or data from the user, sending that data to the

server, and then receiving the server's response. Finally, the client presents the received information back to the user in a usable format.

3. Server

In the client-server model, the server functions as the counterpart to the client. It is a computer that processes and responds to requests sent by clients. The software programs that handle these requests are known as server applications, and the hardware or machine running these applications is referred to as a server machine.

Common types of servers include:

- Web servers – handle website content and HTTP requests
- Database servers – manage and provide access to databases
- Mail servers – send, receive, and store email messages

2.1.2.1 How a Browser Interacts with Servers

When a user accesses a website through a browser, a series of steps take place behind the scenes to retrieve and display the webpage as in Fig 2.1.2.

1. Entering the Website URL

The user begins by typing a web address (for example, www.example.com) into the browser's address bar.

2. DNS (Domain Name System) Resolution

The browser communicates with a DNS server to translate the human-readable domain name into its corresponding IP address, which identifies the server on the internet.

3. Making a Connection

With the IP address in hand, the browser sends an HTTP or HTTPS request to the web server to request the necessary resources.

4. Server Sends a Response

The server processes the request and returns the required website files, including:

- HTML (structure of the page)
- CSS (styling and layout)
- JavaScript (functionality and interactivity)
- Images or other media

5. Rendering the Webpage in the Browser

Once the browser receives the files, it processes them using various engines:

- **DOM Interpreter:** Reads and organizes the HTML to form the page's structure.
- **CSS Engine:** Applies styles and layouts defined in CSS.
- **JavaScript Engine:** Executes JavaScript code to enable interactivity; uses Just-In-Time (JIT) compilation for better performance.

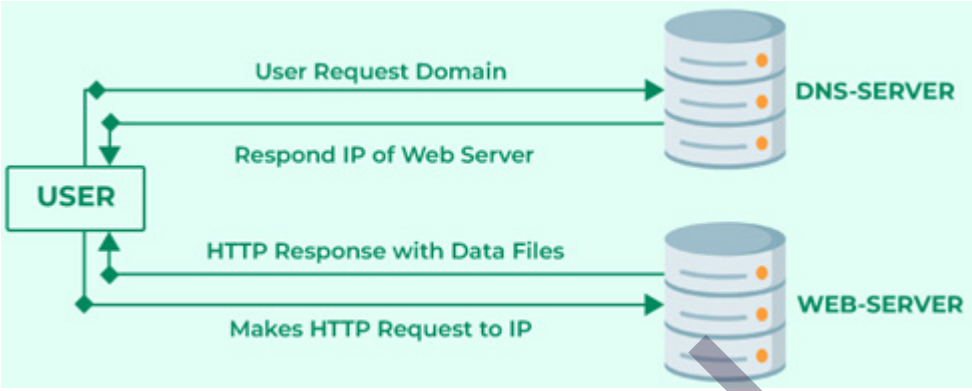


Fig. 2.1.2 Client Server Request and Response

2.1.3 Javascript with HTML

There are several ways to include JavaScript in an HTML page. You can place the JavaScript code in a separate file and link it wherever required, or you can embed the code directly within the HTML document. JavaScript is added to an HTML page using the <script> tag, and it can be included in various ways.

1. Inline JavaScript
2. Internal JavaScript
3. External JavaScript

2.1.3.1 Inline JavaScript

JavaScript code can be embedded directly into an HTML element by using its event attributes, such as onclick, onload, or similar.

```
<html>
<head></head>
<body>
   <h2>

Adding JavaScript in HTML Document

   </h2>
   <button onclick="alert('Button Clicked!')">
```

```
        Click Here
    </button>
</body>
</html>
```

2.1.3.2 Internal JavaScript (Within <script> Tag)

JavaScript code can be included inside the <script> tag within an HTML file. This is referred to as Internal JavaScript and is typically added in the <head> or <body> section of the document.

1. JavaScript Inside the <head> Tag

Placing JavaScript in the <head> section of an HTML page makes sure the script is loaded and executed as the page begins to load. This approach is useful for scripts that must run or initialize before the page content is displayed.

```
<html>
  <head>
    <script>
      function myFun() {
       document.getElementById("demo").innerHTML = "Content changed!";
      }
    </script>
  </head>
  <body>
    <h2>Add JavaScript Code inside Head Section</h2>
    <h3 id="demo" style="color: green;">
      GeeksforGeeks
    </h3>
    <button type="button" onclick="myFun()">
      Click Here
    </button>
  </body>
</html>
```

2. JavaScript Code Inside the <body> Tag

JavaScript can also be included within the <body> section of an HTML document. Placing scripts near the end of the <body> ensures they load after the page content, which is helpful when the script relies on the DOM being fully loaded.

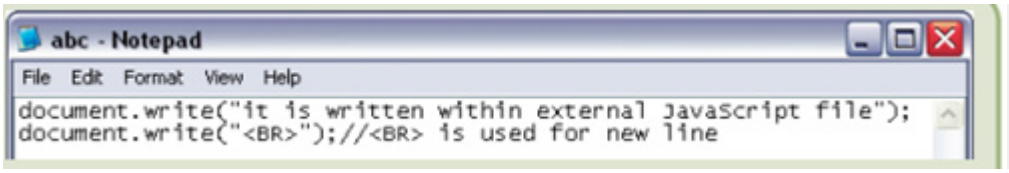
```
<html>
 <head></head>
 <body>
  <h2>Add JavaScript Code inside Body Section</h2>
  <h3 id="demo" style="color: green;">
   GeeksforGeeks
  </h3>
  <button type="button" onclick="myFun()">
   Click Here
  </button>
  <script>
   function myFun() {
    document.getElementById("demo").innerHTML = "Content changed!";
   }
  </script>
 </body>
</html>
```


2.1.3.3 External JavaScript (Using External File)

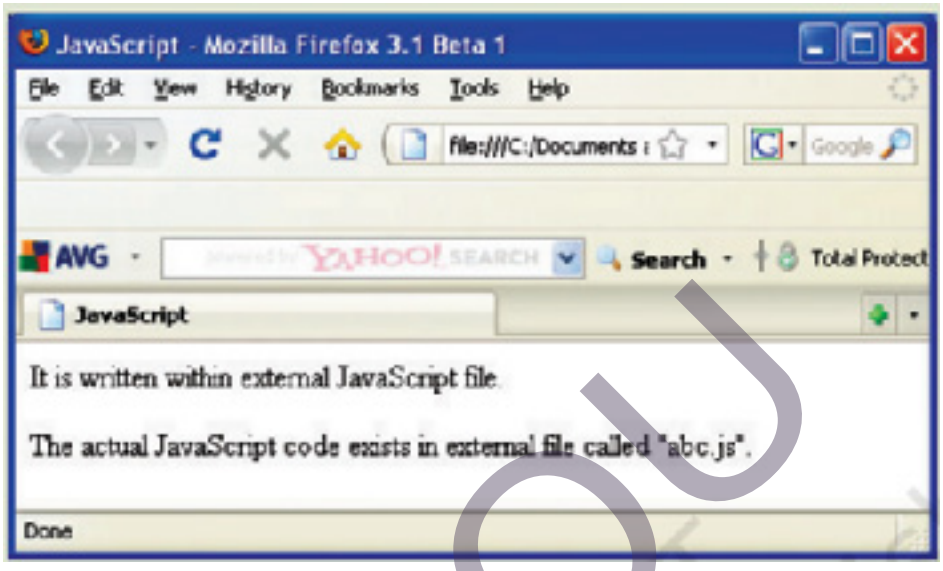
If the same JavaScript code has to be used across multiple HTML documents, the most efficient way is to store it in an external file with the ".js" extension. This can be done by using the src attribute of the <script> tag to link to the external JavaScript file.

```
<html>
<head>
<title>Using External Javascript</title>
</head>
<body>
<script language= "JavaScript" src= "abc.js">
</script>
<p>The actual javascript code exists in external file called "abc.js" . </
p>
</body>
</html>
```

Actual JavaScript file “abc.js”



Output



Recap

- JavaScript is a versatile, lightweight, and dynamically typed scripting language used in modern web development.
- It adds interactivity and dynamic behavior to static web pages.
- JavaScript works closely with HTML and CSS to build rich, responsive user interfaces.
- It supports both client-side and server-side development.
- JavaScript is object-based and supports object-oriented programming.
- It was initially called LiveScript and later renamed JavaScript by Netscape.
- Features of JavaScript include dynamic typing, prototype-based inheritance, and being an interpreted language.
- JavaScript code can be embedded inline, internally within <script> tags, or externally using a .js file.
- Inline JavaScript is added directly to HTML elements using event attributes like onclick.

- Internal JavaScript is written within <script> tags in the head or body of an HTML file.
- External JavaScript is stored in a separate .js file and linked using the src attribute of the <script> tag.
- JavaScript can respond to user actions such as clicks and form input in real time.
- JavaScript manipulates the DOM to dynamically update webpage content without reloading the page.
- JavaScript engines in browsers interpret and execute code line by line.
- The client-server model involves communication between a client (browser) and a server over a network.
- A client collects input from users and sends it to the server for processing.
- A server processes requests and sends back responses like HTML, CSS, JavaScript, and media files.
- DNS translates domain names into IP addresses for server identification.
- JavaScript supports asynchronous communication with servers using AJAX or Fetch API.
- Nodes in a network can include devices like computers, smartphones, and tablets.
- JavaScript plays a vital role in enabling real-time feedback and dynamic page behavior.
- Common types of servers include web servers, database servers, and mail servers.
- Browsers process server responses using a DOM interpreter, CSS engine, and JavaScript engine.

Objective Type Questions

1. What type of language is JavaScript?
2. What tag is used to include JavaScript in an HTML document?
3. What was the original name of JavaScript?
4. What model does JavaScript use for client-server communication?
5. What is the file extension for external JavaScript files?

6. What represents the structure of a web page in JavaScript?
7. Which operator is used to create object instances in JavaScript?
8. What type of typing does JavaScript use?
9. Which method is used to access an HTML element by its ID?
10. What protocol does a browser use to request web resources from a server?

Answers to Objective Type Questions

1. Scripting
2. script
3. LiveScript
4. Client-Server
5. .js
6. DOM
7. new
8. Dynamic
9. getElementById
10. HTTP

Assignments

1. Explain the client-server model in JavaScript with a real-life example.
2. Differentiate between inline, internal, and external JavaScript with examples.
3. List and explain any five key features of JavaScript that make it suitable for web development.
4. Describe the process of how a browser interacts with a server when loading a web page.

5. Write an HTML document that uses internal JavaScript to change the text of a paragraph when a button is clicked

Reference

1. Haverbeke, M. (2018). *Eloquent JavaScript: A modern introduction to programming* (3rd ed.). No Starch Press.
2. Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.
3. Robbins, J. N. (2018). *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics* (5th ed.). O'Reilly Media.
4. Simpson, K. (2015). *You don't know JS: Up & going*. O'Reilly Media.
5. Meloni, J. C. (2018). *HTML, CSS, and JavaScript all in one* (3rd ed.). Sams Publishing.

Suggested Reading

1. Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.
2. Haverbeke, M. (2018). *Eloquent JavaScript: A modern introduction to programming* (3rd ed.). No Starch Press.
3. Robbins, J. N. (2018). *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics* (5th ed.). O'Reilly Media.
4. Simpson, K. (2015). *You don't know JS: Up & going*. O'Reilly Media.
5. Meloni, J. C. (2018). *HTML, CSS, and JavaScript all in one* (3rd ed.). Sams Publishing.

Datatypes, Objects and Expressions

Learning Outcomes

After completing this unit, learners will be able to:

- recall the purpose of statements in JavaScript
- identify the different types of comments used in JavaScript
- recognize valid and invalid JavaScript identifiers
- list the primitive data types available in JavaScript

Prerequisites

Imagine you want to create a simple webpage that not only displays information but also responds to user actions like calculating sums, checking if entered values are valid, or showing different messages based on input. Before diving into complex programs, it is important to understand the basic building blocks of JavaScript: how instructions (statements) are written, how you store and manage data (variables and data types), and how you control the flow of your program using operators.

To build this foundation, we will start by learning what statements are in JavaScript, how to write comments to make code clear, and how to use identifiers and variables correctly. We will then explore the different types of data JavaScript can handle and understand the role of objects and operators in writing functional scripts. Mastering these basics will prepare you to create interactive and dynamic web pages.

Keywords

Comments, Literals, Identifiers, Primitive, Object, Function

Discussion

2.2.1 Statements in Javascript

Statements are commands or instructions given to the JavaScript interpreter to perform specific actions. This interpreter is built into almost all web browsers. A group of statements designed to complete a task is known as a script or program. JavaScript statements can be written as follows:

```
a = 100;                                    // stores value 100 in variable a
b = 200;                                    // stores value 200 in variable b
c = a + b;                                  // stores the sum of a and b in variable c
document.write ("Sum of A and B : ");       // displays the string
document.write(c);                          // displays the value of c
```

In JavaScript, a semicolon (;) is typically used to terminate a statement. However, when two statements are written on separate lines, the semicolon may be omitted.

Examples of valid statements:

```
(i) p = 10
    q = 20
(ii) x=12; y=25    // semicolon(;) separating two statements.
```

Example of an invalid statement:

```
x = 12 y = 25     // two statements on the same line without a semicolon
```

2.2.1.1 Comments

Comments are lines in the code that the interpreter completely ignores. They are used to add notes or explanations, making the code clearer and easier for other developers to understand. JavaScript supports two types of comments:

- Single-line comments using double slashes //
- Multi-line comments enclosed within /* ... */

2.2.1.2 Literals

Literals are constant values that are written directly in the JavaScript code.

```
a = 10;
b = 5.7;
document.write("Welcome");
```

In the above statements, 10, 5.7, and "Welcome" are all literals.

2.2.1.3 Identifiers

Identifiers are the names assigned by a programmer to variables, functions, arrays, and other elements in JavaScript. They can consist of uppercase and lowercase letters, numbers, underscores (_), and dollar signs ($). However, an identifier cannot start with a number and must not use any reserved JavaScript keywords as its name.

Some valid identifiers are : RollNo, bus_fee, _vp, $amt

Some invalid identifiers are : to day // Space is not allowed

17nov // must not begin with a number

%age // no special character is allowed

2.2.1.4 Reserved Words or Keywords

Reserved words in JavaScript are predefined terms that carry specific meanings and instructions for the interpreter. Because of this, they cannot be used as identifiers such as variable names, array names, object names, or function names. These reserved words are also referred to as Keywords. For example, the function keyword is used to create functions, while let, var, and const are used to declare variables.

2.2.1.5 Variables

A variable is an identifier used to hold values, which can be modified while the script is running. After a value is assigned to a variable, it can be accessed by its name. Although declaring a variable is not mandatory in JavaScript, it is considered good practice. The var keyword is used to declare variables.

Syntax : var var-name [= value] [..., var-name [= value]]

Example var name = “Sachin”; // Here ‘name’ is variable

document.write(name); // Prints Sachin

A JavaScript variable can hold a value of any data type.

For example : i = 7;

document.write(i); // prints 7

i = “seven”; // JavaScript allows to assign string values

document.write(i); // prints seven

Some valid examples of variable declaration: var cost; var num, cust_no = 0; var amount = 2000;

2.2.2 Data Types

JavaScript data types define the kind of data a variable can hold, influencing how values are handled and used in the program. Each data type has its own characteristics and behavior that determine how the data is stored, accessed, and manipulated.

JavaScript data types are categorized into Primitive and Non-Primitive types as in Fig 2.2.1

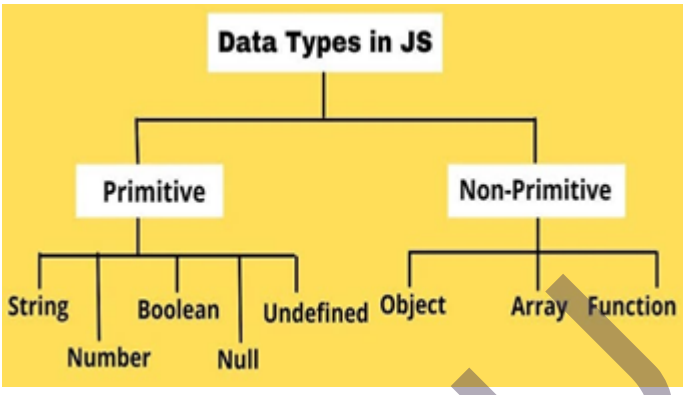


Fig. 2.2.1 Data Types

2.2.2.1 Primitive Data Types

In JavaScript, primitive data types represent basic, immutable values that are stored directly in memory, helping maintain efficiency in both memory usage and performance.

1. String

In JavaScript, a string is a sequence of characters enclosed within quotes.

```
let s1 = "Hello There";
console.log(s1);
let s2 = 'Single quotes work fine';
console.log(s2);
let s3 = `can embed ${s1}`;
console.log(s3);
```

Output

```
Hello There
Single quotes work fine
can embed Hello There
```

In JavaScript, there's no distinction between using single (' ') or double (" ") quotes for strings. However, backticks (` `) offer additional features, such as allowing variables and expressions to be embedded directly within the string.

2. Number

The Number data type in JavaScript includes both integers and floating-point numbers. Special values like Infinity, -Infinity, and NaN represent infinite values and computational errors, respectively.

```
let n1 = 2;
console.log(n1)
let n2 = 1.3;
console.log(n2)
```

Output

2

1.3

3. Boolean

A Boolean variable can hold only two possible values: true or false. Internally,the value true is typically represented as 1, and false as 0. Boolean variables are commonly used in programming and logical expressions to evaluate conditions and determine the truth value of a statement.

```
let b1 = true;
console.log(b1);
let b2 = false;
console.log(b2);
```

Output

true

false

4. Null

JavaScript includes a special data type called null, which represents "no value" or an empty value. It is important to note that null is not the same as 0.

```
let age = null;
console.log(age)
```

Output

null

5. Undefined

A variable that is declared but not given an initial value is automatically assigned the value undefined. This indicates that the variable exists but has not yet been assigned any value.

```
let a;
console.log(a);
```

Output

Undefined

2.2.2.2 Non-Primitive Data Types

Non-primitive data types are those that are built upon or derived from primitive data types. They are also referred to as derived data types or reference data types.

1. Object

In JavaScript, objects are collections of key-value pairs used for storing data. They can be created using curly braces {} or the new keyword. Objects are essential in JavaScript, as most elements in the language are based on them.

```
let gfg = {
    type: "Company",
    location: "Noida"
}
console.log(gfg.type)
```

Output

Company

2. Arrays

An array is a specific type of object designed to hold an ordered list of values, which can belong to any data type.

```
let a1 = [1, 2, 3, 4, 5];
console.log(a1);
let a2 = [1, "two", { name: "Object" }, [3, 4, 5]];
console.log(a2);
```

Output

```
[ 1, 2, 3, 4, 5 ]
[ 1, 'two', { name: 'Object' }, [ 3, 4, 5 ] ]
```

3. Function

In JavaScript, a function is a reusable block of code created to carry out a particular task whenever it is invoked.

```
// Defining a function to greet a user
function greet(name) { return "Hello, " + name + "!"; }
// Calling the function
console.log(greet("Ajay"));
```

Output

```
Hello, Ajay!
```

2.2.3 Objects

JavaScript is a scripting language that follows an object-based approach. It enables the creation of custom objects and user-defined variable types, while also providing a collection of built-in objects. Elements on a web page such as tables, forms, buttons, images, and links are treated as objects. Each object has associated values known as properties, and the operations it can perform are referred to as methods or behaviors. You can access an object's property using the syntax: ObjectName.PropertyName.

2.2.3.1 Document Object

The Document object is a component of the Window object and can be accessed using window.document. It represents the HTML document loaded in the browser and

provides access to all the elements within that document. For instance, the title of the current page can be retrieved using the document.title property.

Some of the common properties of document object are given in Table 2.2.1.

Table 2.2.1 Properties of Document Object

Properties	Purposes
Title	returns/ sets title of the current document.
bgColor	returns/ sets the background color of the current document.
linkColor	returns/ sets the color of hyperlinks in the document.
Forms	returns a list of the FORM elements within the current document.
URL	returns a string containing the URL of the current document.

2.2.3.2 Date Object

The JavaScript Date object is used for handling dates and times, enabling the creation, modification, and formatting of date-related data.

```
// Creating a new Date object for the current date and time
let currentDate = new Date();
// Displaying the current date and time
console.log(currentDate);

Output
2025-03-08T06:23:33.202Z
```

2.2.3.3 Math Object

The Math object provides methods and constants for performing advanced mathematical calculations as given in Table 2.2.2. Unlike other objects, it cannot be instantiated. All of its properties and methods are static, meaning they are accessed directly from the Math object. For example, the constant π is referenced as Math.PI, and the sine of a value can be calculated using Math.sin(x), where x is the input value.

Table 2.2.2 Math Object Properties

Properties	Description
Math.PI	Returns the value of p
Math.E	Euler's constant and the base of natural logarithms.
Math.LN2	Natural logarithm of 2
SQRT1_2	Square root of ½.

2.2.4 Expressions and Operators

An expression is a sequence of operators and operands that can be evaluated to produce a result. It can also include function calls that return values.

Example : “Hello India!” // a string literal

x = 7.5 // a numeric literal

Operators are symbols or reserved words used to carry out specific operations on one or more values, called operands. JavaScript provides a wide range of operators, such as arithmetic, comparison, logical, assignment, and others.

2.2.4.1 Arithmetic Operators

Arithmetic operators are used to carry out mathematical operations such as subtraction, division, multiplication, and more as given in Table 2.2.3. They operate on one or more numeric values (which can be either literals or variables) and return a single numeric result.

Assume variable x holds 10 and variable y holds 20, then the operations of arithmetic operators are depicted in the Table 2.2.3.

Table 2.2.3 Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds two operands.	x + y will give 30
- (Subtraction)	Subtracts the second operand from the first.	x - y will give -10.
* (Multiplication)	Multiplies both operands.	x * y will give 200
/ (Division)	Divides the numerator by the denominator.	y / x will give 2
% (Modulus)	Outputs the remainder of an integer division.	y % x will give 0
++ (Increment)	Increases an integer value by one.	x++ will give 11
-- (Decrement)	Decreases an integer value by one.	x-- will give 9

2.2.4.2 Comparison Operators

JavaScript comparison operators are used to compare two values and return a Boolean result either true or false as in Table 2.2.4. The language provides several types of comparison operators.

For example, if variable x is assigned the value 10 and variable y is assigned 20, then the operations of comparison operators are depicted in the Table 2.2.4.

Table 2.2.4 Comparison Operators

Operator	Description	Example
== (Equal)	Checks if the value of two operands is equal or not. If yes, then the condition becomes true.	(x == y) is not true.
!= (Not Equal)	Determines whether the values of two operands are equal. If they are not equal, the condition evaluates to true.	(x != y) is true.
=== (Strict equality)	It verifies whether both the value and data type of the operands are the same. If they match, the condition evaluates to true.	(x === y) is not true.
!== (Strict inequality)	It verifies whether the value and data type of the variable are not equal. If they differ, the condition evaluates to true.	(x !== y) is true
> (Greater than)	Evaluates whether the left operand's value is greater than the right operand's value. If it is, the condition returns true.	(x > y) is not true.
< (Less than)	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.	(x < y) is true.
>= (Greater than or Equal to)	Determines whether the left operand's value is greater than or equal to that of the right operand. If so, the condition evaluates to true.	(x >= y) is not true.
<= (Less than or Equal to)	Evaluates whether the value on the left is less than or equal to the value on the right. If this condition is met, it returns true.	(x <= y) is true

2.2.4.3 Logical Operators

Logical operators are used to combine multiple conditions. JavaScript provides the following three logical operators as given in table 2.2.5

Table 2.2.5 Logical Operators

Operator	Description with Example
&& (AND)	returns true if both operands are true else it returns false.
|| (OR)	returns false if both operands are false else it returns true.
! (NOT)	returns true if the operand is false and false if operand is true.

2.2.4.4 Concatenation Operators

The + operator concatenates two string operands. When used with both string and numeric operands, the + operator gives priority to string operands.It is evaluated from left to right,and the results depend on the order in which operations are performed.

For example :

Statement	Output
“Good” + “Morning”	“GoodMorning”
“Mumbai” + 0 +0+ 7	“Mumbai007”

2.2.4.5 Special Operators

1. Conditional Operator

The conditional operator in JavaScript is a unique operator that requires three operands, which is why it is also known as the Ternary Operator. It assigns a value to a variable depending on whether a specified condition is true or false.

Syntax: var_name = (condition) ? v_1 : v_2

If (condition) is true, the value v_1 is assigned to the variable, otherwise, it assigns the value v_2 to the variable.

Example : status = (age >= 18) ? “adult” : “minor”

This statement sets the variable status to "adult" if the age is 18 or older; if not, it assigns "minor" to status.

2. New

The new operator is used to create an instance and allocate memory for either predefined or user-defined object types.

Syntax: ObjectName = new objectType (param1 [,param2] ...[,paramN])

Example :d = new Date(); // date assigns to object d

r = new rectangle(4, 5, 7, 8);

3. Delete

The delete operator frees up the memory that was allocated with the new operator by removing an object, a property of an object, or an element from an array.

Syntax: delete object_name

delete object_name.property

delete array_name[index]

The delete operator can remove variables that are declared implicitly, but it cannot delete variables declared using the var keyword. It returns true if the deletion is successful and false if the operation cannot be performed.

4. This

JavaScript includes this operator, which refers to the current object. It acts like a reference or pointer to the object in context.

Syntax: this[.propertyName]

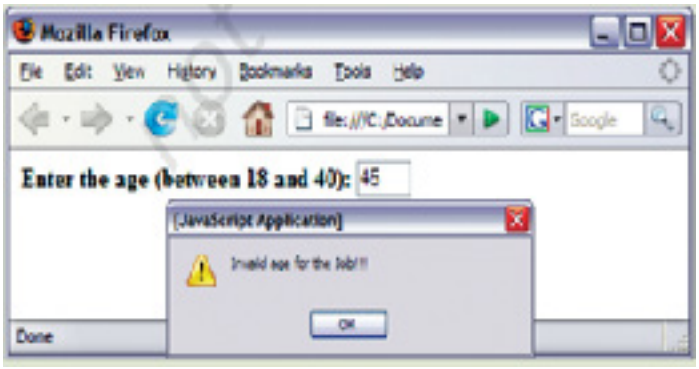
Example :

Use of this operator to validate the age. Here input is provided through the text box.

```
<html>
<head>
<script language =“JavaScript” type=”text/javascript”>
 Function validate(obj,min_age,max_age)
{
if((obj.value<min_age)||(obj.value>max_age))
alert(“Invalid age for the job!!!”);
</script>
</head>
<body>
<B>Enter the age(between 18 and 40)</B>
<Input type=”text” Name=”age” size=2
onChange=”validate(this,18,40)”>
</body>
</html>
```

In this example, the validate() function is triggered using the onChange event handler. The this operator is used to pass the current object (i.e., the text box) to the function.

Output



Recap

- JavaScript statements are instructions executed by the browser's interpreter, usually ending with a semicolon.
- Comments can be single-line (//) or multi-line (/* ... */) and are ignored by the interpreter.
- Literals represent constant values directly written in code, such as numbers and strings.
- Identifiers are names for variables and functions, following naming rules (no starting with numbers or using reserved keywords).
- Reserved words (keywords) have predefined meanings and cannot be used as identifiers.
- Variables hold data that can be changed during program execution and are declared using the var keyword.
- JavaScript has primitive data types: String, Number, Boolean, Null, and Undefined.
- Non-primitive data types include Objects, Arrays, and Functions, which are collections or reusable blocks of code.
- Objects represent entities with properties and methods; examples include Document, Date, and Math objects.
- Expressions combine operators and operands to produce values.
- Arithmetic operators perform math operations like addition, subtraction, multiplication, division, modulus, increment, and decrement.
- Comparison operators compare values and return Boolean results, including equal, not equal, strict equal, and inequality checks.

- Logical operators (AND, OR, NOT) combine or invert Boolean conditions.
- The + operator concatenates strings, prioritizing string operations from left to right.
- Special operators include the conditional (ternary) operator, new (to create object instances), delete (to remove objects or properties), and this (refers to the current object).
- The this operator is useful for referencing the current element or object in functions and event handlers.

Objective Type Questions

1. What symbol terminates a JavaScript statement?
2. What syntax starts a single-line comment?
3. What type of value is "Hello"?
4. What character cannot start an identifier?
5. What keyword declares a variable?
6. What data type holds true or false?
7. What object represents the current webpage?
8. What operator adds two numbers?
9. What operator checks equality including type?
10. What keyword refers to the current object?

Answers to Objective Type Questions

1. Semicolon
2. // (double slashes)
3. String
4. Number
5. var
6. Boolean

7. Document
8. +(plus)
9. === (strict equality)
10. this

Assignments

1. Write a JavaScript program that declares variables, performs arithmetic operations, and displays the result using document.write().
2. Explain the difference between primitive and non-primitive data types in JavaScript with examples.
3. Write a function in JavaScript that uses the this keyword to validate user input for an age range between 18 and 40.
4. Describe the purpose of the new and delete operators in JavaScript with suitable examples.
5. Create a JavaScript object representing a book with properties such as title, author, and year. Write a script to display the book's title using the object's property.

Reference

1. Haverbeke, M. (2018). *Eloquent JavaScript: A modern introduction to programming* (3rd ed.). No Starch Press.
2. Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.
3. Robbins, J. N. (2018). *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics* (5th ed.). O'Reilly Media.
4. Simpson, K. (2015). *You don't know JS: Up & going*. O'Reilly Media.
5. Meloni, J. C. (2018). *HTML, CSS, and JavaScript all in one* (3rd ed.). Sams Publishing.

Suggested Reading

1. Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.

2. Haverbeke, M. (2018). *Eloquent JavaScript: A modern introduction to programming* (3rd ed.). No Starch Press.

3. Robbins, J. N. (2018). *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics* (5th ed.). O'Reilly Media.

4. Simpson, K. (2015). *You don't know JS: Up & going*. O'Reilly Media.

5. Meloni, J. C. (2018). *HTML, CSS, and JavaScript all in one* (3rd ed.). Sams Publishing.

Pop up Boxes

Learning Outcomes

After completing this unit, learners will be able to:

- recall what a popup box is and its purpose in web pages
- identify the three types of popup boxes: Alert, Confirm, and Prompt
- explain how an Alert Box, Confirm Box, and Prompt Box work
- familiarize how to use \n to display messages in multiple lines in popup boxes

Prerequisites

Popup boxes are not just for showing messages. They play an important role in making websites interactive and safe. Imagine filling out an online form and accidentally pressing the wrong button. A confirm box can appear asking, "Are you sure you want to submit?" This gives you a chance to check and avoid mistakes. Similarly, a prompt box can quickly collect your input, like your name, so the website can greet you personally. And an alert box can warn you about important things, like low battery notifications or page restrictions.

In this unit, you will explore three types of popup boxes: Alert, Confirm, and Prompt. Each serves a different purpose, just like conversations in real life. Sometimes you simply need to be informed, sometimes you are asked to agree or disagree, and sometimes you are asked a question and expected to respond. Through simple examples, activities, and exercises, you will learn not only how to create these popup boxes but also when and why to use them effectively.

By the end of this unit, you will be able to create interactive web pages that can warn users, ask for confirmation, and collect input in a way that is easy to understand and fun to use. So let's dive in and discover the world of popup boxes in JavaScript!

Keywords

Alert box, confirm box, prompt box, line breaks

Discussion

2.3.1 What is a Pop up box?

When you visit a website, have you noticed small boxes suddenly appearing on the screen with a message like “Are you sure you want to leave this page?” or “Please enter your name to continue”? These are called popup boxes.

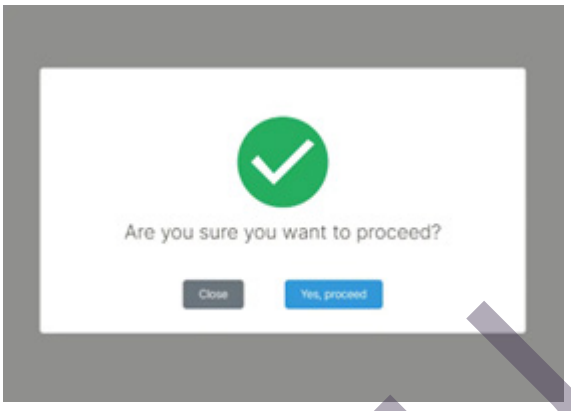


Fig. 2.3.1 A Pop up box

In JavaScript, popup boxes are a simple but powerful way for a web page to communicate with the user. They can be used to give information, ask for confirmation, or collect input. JavaScript provides three types of popup boxes, each one serves a different purpose, just like different kinds of conversations in real life. Sometimes we just inform, sometimes we ask for agreement, and sometimes we ask a question and wait for an answer. In this unit, we will explore these popup boxes step by step with simple examples, activities, and practice exercises.

2.3.2 Types of Pop up Boxes

Popup boxes are small windows that appear on the screen, and the user has to respond to them before continuing with the page. They are useful for showing important messages or warnings, asking the user to confirm an action, and collecting input directly from the user.

JavaScript provides us with three simple popup boxes for this:

1. Alert Box
2. Confirm Box
3. Prompt Box

2.3.2.1 Alert Box

An Alert Box is just like your teacher entering the class and saying:

"Listen everyone! Tomorrow is a holiday."

You cannot ignore it. Everyone must hear it and nod.

Similarly in Javascript an Alert Box is used when we want to display information to the user. The user must click OK to continue.

Syntax:

```
alert("Your message here");
```

Example 1: Simple Alert

```
alert("An alert box!");
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Alert</h2>
<button onclick="myFunction( )">Click here</button>
<script>
function myFunction( ) {
  alert("An alert box!");
}
</script>
</body>
</html>
```

When this code runs, a small window will appear with the text "I am an alert box!". The user must click OK to proceed.

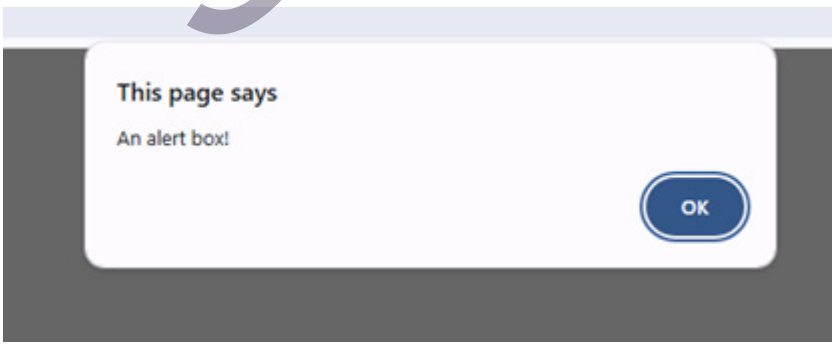


Fig. 2.3.2 Simple alert

Example 2: Alert with Warning

```
alert("Warning! Do not refresh this page!");
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Alert Example</title>
 <script>
// This function shows an alert box with a warning message
  function showWarning( ) {
   alert("Warning! Do not refresh this page!");
  }
 </script>
</head>
<!-- The onload attribute calls the showWarning() function
    automatically when the page loads - - >
 <body onload="showWarning( )">
  <h2>JavaScript Alert Example</h2>
 <p>This page will show an alert message when it loads.</p>
 </body>
</html>
```

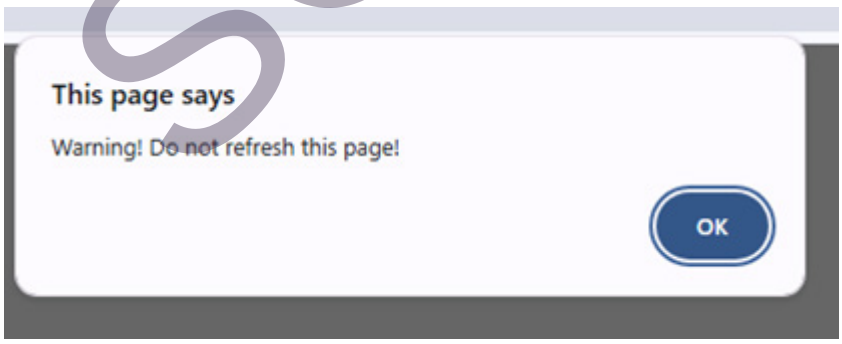


Fig. 2.3.3 Alert with warning

- The Alert Box in JavaScript always has only one button — OK.

When to Use an Alert Box?

- To display warnings (e.g., "Low battery!").
- To display notifications (e.g., "Form submitted successfully!").
- To ensure the user reads the message before continuing.

2.3.2.2 Confirm Box

Imagine you are about to delete a file on your computer. The system won't just delete it immediately, right?

It first asks you:

"Are you sure you want to delete this file?"

Here, you have two choices:

If you click OK , it means Yes, go ahead.

If you click Cancel , it means No, stop it.

That's exactly what a Confirm Box does in JavaScript! A Confirm Box is used when you want the user to verify or agree to an action. When it appears, the user must choose either "OK" or "Cancel" to continue. If the user clicks OK, the box returns true, and if the user clicks Cancel, it returns false.

Syntax:

```
confirm("Do you want to continue?");
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Confirm Box Example</title>
  <script>
// This function will be called when the user clicks the "Delete File" button
    function deleteFile( ) {
  // A confirm box appears asking the user for confirmation
// It returns true if the user clicks "OK"
// It returns false if the user clicks "Cancel"
      if (confirm("Do you really want to delete this file?")) {
    // If the user clicked "OK", show this alert message
        alert("File deleted!");
      } else {
```

```
    // If the user clicked "Cancel", show this alert message
                    alert("File not deleted.");
                }
            }
        </script>
    </head>
    <body>
        <h2>Confirm Box Example</h2>
<!-- A button that runs the deleteFile() function when clicked - - >
        <button onclick="deleteFile( )">Delete File</button>
    </body>
    </html>
```

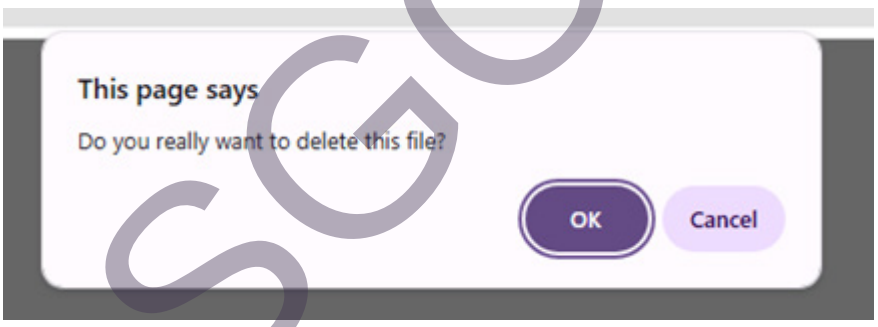


Fig. 2.3.4 Confirm box

A Confirm Box is useful whenever you need the user to make a decision before continuing with an action. It is commonly used in situations where the action may be important, risky, or irreversible.

2.3.2.3 Prompt box

A Prompt Box feels like a little question window that appears on your screen. It doesn't just talk to you like an alert or ask you to choose like a confirm box. It actually lets you answer! Inside the prompt box, there is a small text field where you can type something, and two buttons below it: OK and Cancel. If you type your answer and press OK, the box will send back whatever you wrote. If you decide not to answer and click Cancel, it will return null. This is a quick way for a web page to collect small pieces of information, like asking your name before greeting you personally.

Syntax

```
prompt("Message", "default text");
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Prompt Box Example</title>
</head>
<body>
<h2>JavaScript Prompt Box Example</h2>
<p>Click the button below to enter your name using a prompt box.</p>
<!-- A button that will call the JavaScript function askName( ) when clicked
- - >
<button onclick="askName( )">Try it</button>
<script>
// This function will be executed when the button is clicked
function askName() {
 // Show a prompt box asking the user to enter their name
// The second argument ("Harry Potter") is the default text shown in the box
let name = prompt("Please enter your name:", "Harry Potter");
// Check if the user clicked Cancel or left the field empty
if (name == null || name == "") {
  // If Cancel or empty input, show this alert message
    alert("User cancelled the prompt.");
  } else {
 // If the user entered a name, display a personalized greeting
    alert("Hello " + name + "!");
  }
}
</script>
</body>
</html>
```

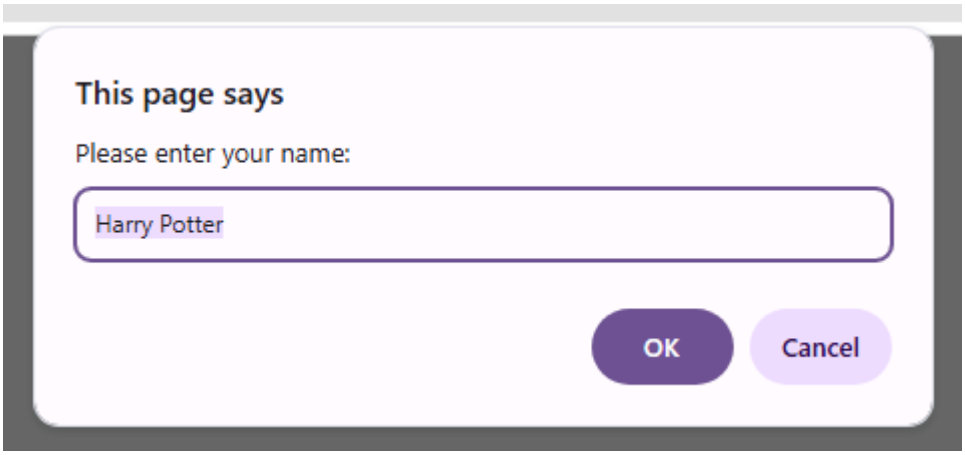


Fig. 2.3.5 Prompt box

2.3.2.4 Line Breaks in Popup Boxes

Sometimes, a message in a popup box can be too long for a single line. In such cases, we may want to split the message into multiple lines to make it easier to read. In JavaScript, we can do this by using the \n character, which tells the popup box to move the text to the next line.

The \n is called a newline character. It works in alert boxes, confirm boxes, and prompt boxes wherever text is displayed.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Line Breaks in Alert Box</title>
  <script>
// This function shows an alert box with multiple lines using \n
    function showMultiLineAlert() {
      alert("Hello!\nWelcome to our website.\nHave a great day!");
    }
  </script>
</head>
<body>
  <h2>Line Breaks in Popup Boxes Example</h2>
<p>Click the button below to see an alert box with multiple lines.</p>
```

```
<!-- When this button is clicked, the function showMultiLineAlert() is called - - >
        <button onclick="showMultiLineAlert()">Show Alert</button>
    </body>
    </html>
```

Fig. 2.3.6 On clicking the 'Show Alert' button as shown

Fig. 2.3.7 An alert in multiple lines will be displayed

Recap

Popup Boxes (JavaScript)

- Small windows appearing on the screen
- Require user interaction before continuing
- Types: Alert, Confirm, Prompt
- Alert: display information, only OK button
- Confirm: ask permission, OK or Cancel, returns true/false
- Prompt: collect input, text field, OK or Cancel, returns value/null
- Line breaks using \n for multi-line messages

- Improve user interaction and avoid mistakes
- Useful for warnings, confirmations, and input collection
- Easy to implement with simple JavaScript code

Objective Type Questions

1. Which programming language provides popup boxes?
2. How many types of popup boxes are available in JavaScript?
3. Which popup box is used only for displaying information?
4. Which popup box is used to ask for confirmation?
5. Which popup box allows the user to enter input?
6. What is the only button available in an alert box?
7. What are the two buttons available in a confirm box?
8. What are the two buttons available in a prompt box?
9. What value does a confirm box return when the user clicks OK?
10. What value does a confirm box return when the user clicks Cancel?
11. What is returned by a prompt box if the user clicks Cancel?
12. What is the default text argument in a prompt box called?
13. Which popup box is useful for displaying warnings?
14. Which popup box is useful for asking decisions?
15. Which popup box is useful for collecting input?
16. Which character is used in JavaScript for line breaks?
17. What is the newline character in JavaScript?
18. Which popup box is best suited to display success messages?
19. Which popup box is best suited to display error messages?
20. Which popup box must the user respond to before continuing?

21. What type of window is a popup box—small or large?

22. What keyword is used in JavaScript to show an alert?

23. What keyword is used in JavaScript to show a confirm box?

24. What keyword is used in JavaScript to show a prompt box?

25. In which popup box does the user type input?

26. In which popup box does the user only click OK or Cancel?

27. In which popup box does the user only click OK?

28. What is shown to the user in a confirm box if they must verify an action—Message or Image?

29. In a prompt box, what type of input field is displayed—Text or Number?

30. What type of communication are popup boxes—One-way or Two-way?

Answers to Objective Type Questions

1. JavaScript
2. Three
3. Alert
4. Confirm
5. Prompt
6. OK
7. OK and Cancel
8. OK and Cancel
9. True
10. False
11. Null
12. Default

13. Alert

14. Confirm

15. Prompt

16. \n

17. \n

18. Alert

19. Alert

20. Popup

21. Small

22. alert

23. confirm

24. prompt

25. Prompt

26. Confirm

27. Alert

28. Message

29. Text

30. Two-way

Assignments

1. Explain the three types of JavaScript popup boxes with suitable examples.

2. Describe the importance of the \n newline character in popup boxes. Give an example showing its usage

3. Suppose you are designing an online examination system. Which popup box would you use to:

- Warn students not to close the browser,
- Confirm submission of the exam, and
- Collect the student's name before starting the test? Justify your choices.

4. A developer uses an alert box to ask the user for confirmation before deleting a record. Analyze whether this is the correct choice. If not, suggest the proper popup box with reasons.

Reference

1. Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.
2. McFarland, D. S. (2015). *JavaScript & jQuery: The missing manual* (3rd ed.). O'Reilly Media.
3. Goodman, D. (2007). *JavaScript Bible* (6th ed.). Wiley Publishing.
4. Zakas, N. C. (2012). *Professional JavaScript for web developers* (3rd ed.). Wrox.
5. Resig, J., & Bibeault, B. (2013). *Secrets of the JavaScript ninja* (2nd ed.). Manning Publications.

Suggested Reading

1. W3Schools. (n.d.). *JavaScript popup boxes.* Retrieved September 29, 2025, from https://www.w3schools.com/js/js_popup.asp
2. Mozilla Developer Network (MDN). (n.d.). *Window.alert().* Retrieved September 29, 2025, from https://developer.mozilla.org/en-US/docs/Web/API/Window/alert

Branching and Looping

Learning Outcomes

After completing this unit, learners will be able to:

- identify different branching statements
- recall looping statements
- recognize how to create and access objects and their properties
- remember the syntax and basic use of functions, parameters, and return statements

Prerequisites

In today's digital world, websites and web applications are no longer just static pages; they are dynamic, interactive platforms that respond to user actions, process information, and provide meaningful feedback. JavaScript is the engine that powers this interactivity, making web pages lively and responsive. By learning JavaScript, you gain the ability to control how a program reacts to decisions, how it repeats tasks efficiently, and how it organizes information using objects and functions. These are fundamental skills not only for web development but also for understanding the logic that drives modern software applications.

Imagine building a web page where users can submit a form, receive instant feedback, and see results calculated in real time. Or think of creating a game, a data visualization, or a personalized dashboard. All these require understanding how programs make decisions, repeat actions, and manage data effectively. Branching statements help your program "think" and choose actions based on conditions. Looping statements allow it to perform repetitive tasks without rewriting code. Object handling and functions make your code structured, reusable, and smarter, just like organizing real-world entities into manageable units.

This topic invites you to step into the world of programming logic and problem-solving. By exploring branching, loops, objects, and functions, you will not only learn how to write code but also how to think like a programmer. Every concept you master here is a building block for creating real-world applications and transforming your creative ideas into reality. Get ready to dive into JavaScript and experience the power of making your web pages interactive, intelligent, and engaging.

Keywords

if, if…..else, else if ladder, switch, loops, object handling statements

Discussion

2.4.1 Branching statements

Branching statements allow your program to make decisions based on certain conditions. Depending on whether a condition is true or false, your program can take different paths.

2.4.1.1 if Statement

The if statement executes a block of code only if a condition is true.

Syntax:

```
if (condition) {
  // code to execute if condition is true
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>if Statement Example</title>
</head>
<body>
<script>
// Declare a variable
let age = 18;
// Check if the person is eligible to vote
if (age >= 18) {
   // This block runs only if the condition is true
   alert("You are eligible to vote.");
}
</script>
</body>
</html>
```

2.4.1.2 if...else Statement

The if...else statement allows the program to choose between two options.

Syntax:

```
if (condition) {
  // code if condition is true
} else {
  // code if condition is false
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>if...else Example</title>
</head>
<body>
<script>
let marks = 45;
// Check pass or fail
if (marks >= 50) {
   alert("You passed the exam!");
} else {
   alert("You failed the exam.");
}
</script>
</body>
</html>
```


2.4.1.3 else if Ladder

The else if ladder in JavaScript is used when a program needs to check multiple conditions one after another. Instead of using multiple separate if statements, the else if ladder allows the program to evaluate a series of conditions in a sequence. The first condition that evaluates to true executes its corresponding block of code, and the rest are skipped. If none of the conditions are true, the optional else block runs. This structure helps in making decisions more organized and efficient, especially when there are several possible outcomes to consider.

Syntax:

```
if (condition1) {
   // code to execute if condition1 is true
}
else if (condition2) {
   // code to execute if condition2 is true
}
else if (condition3) {
   // code to execute if condition3 is true
}
// you can add more else if blocks as needed
else {
   // code to execute if none of the above conditions are true
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>else if Ladder Example</title>
</head>
<body>
<script>
let score = 85;
// Determine grade
if (score >= 90) {
   alert("Grade A");
} else if (score >= 75) {
   alert("Grade B");
} else if (score >= 50) {
   alert("Grade C");
} else {
   alert("Fail");
}
</script>
</body>
</html>
```

2.4.1.4 switch Statement

The switch statement in JavaScript is a control structure that is particularly useful when a variable can take multiple possible values, and you want to execute different blocks of code based on each value. Instead of writing many if...else if statements, a switch provides a cleaner and more organized way to handle multiple cases. Each possible value is defined in a case block, and the code under the matching case is executed. An optional default block can be included to handle any values that don't match the specified cases. This makes the code easier to read, maintain, and debug when dealing with multiple conditional options.

Syntax:

```
switch(expression) {
  case value1:
    // code
    break;
  case value2:
    // code
    break;
  default:
    // code if no case matches
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>Switch Statement Example</title>
</head>
<body>
<script>
let day = 3;
// Determine day of the week
switch(day) {
    case 1:
        alert("Monday");
        break;
```

```
        case 2:
            alert("Tuesday");
            break;
        case 3:
            alert("Wednesday");
            break;
        default:
            alert("Other day");
    }
    </script>
    </body>
    </html>
```

2.4.2 Looping Statements

Loops in JavaScript are used to repeat a block of code several times without writing the same code again and again. They help automate tasks that need to be done repeatedly, such as printing numbers, processing items in a list, or performing calculations multiple times. By using loops, a program can save time, reduce errors, and make the code shorter and easier to manage.

2.4.2.1 for Loop

In JavaScript, the for loop is used when we want to repeat a block of code a fixed number of times. It works by initializing a variable, checking a condition, and updating the variable after each iteration. As long as the condition is true, the loop continues to execute the code inside its block. Once the condition becomes false, the loop stops running. This makes the for loop especially useful when we know in advance how many times we want the code to repeat, such as printing numbers from 1 to 10 or processing items in a list.

Syntax:

```
for(initialization; condition; increment/decrement) {
  // code to execute
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>For Loop Example</title>
</head>
<body>
  <h2>JavaScript For Loop Example</h2>
  <p>Open the console to see the output.</p>
  <script>
// JavaScript For Loop Example
// The loop starts with i = 1
// It runs as long as i <= 5
// After each iteration, i increases by 1
    for (let i = 1; i <= 5; i++) {
      // Print the current value of i in the console
      console.log("Number is: " + i);
    }
  </script>
</body>
</html>
```


2.4.2.2 while Loop

In JavaScript, the while loop is used to repeatedly execute a block of code as long as a specified condition remains true. Before each iteration, the condition is evaluated, and if it returns true, the code inside the loop is executed. Once the condition evaluates to false, the loop terminates, and the program continues with the next statement following the loop. The while loop is particularly useful when the number of iterations is not predetermined and depends on a condition, such as waiting for a value to reach a specific limit or processing input until the user decides to stop.

Syntax:

```
while (condition) {
  // code
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>While Loop Example</title>
</head>
<body>
  <h2>JavaScript While Loop Example</h2>
  <script>
// Initialize variable
    let i = 1;
// The while loop will run as long as i <= 5
    while (i <= 5) {
// Print the current value of i on the web page
      document.write("Number is: " + i + "<br>");
// Increase i by 1 after each iteration
      i++;
    }
  </script>
</body>
</html>
```


2.4.2.3 break and continue Statements

In JavaScript, the break and continue statements are used to control the flow of loops. The break statement immediately exits a loop when a certain condition is met, even if the loop has not finished all its iterations. On the other hand, the continue statement skips the current iteration of the loop and moves on to the next one without stopping the entire loop. These statements make loops more flexible by allowing programmers to stop or skip certain parts of the execution based on conditions.

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>Break and Continue Example</title>
</head>
<body>
   <h2>JavaScript Break and Continue Example</h2>
   <script>
      document.write("<h3>Using break:</h3>");
      for (let i = 1; i <= 10; i++) {
         if (i === 6) {
            // Exit the loop completely when i = 6
            break;
         }
         document.write("Number is: " + i + "<br>");
      }
      document.write("<h3>Using continue:</h3>");
      for (let i = 1; i <= 10; i++) {
         if (i === 6) {
            // Skip this iteration when i = 6
            continue;
         }
         document.write("Number is: " + i + "<br>");
      }
   </script>
</body>
</html>
```

2.4.3 Object Handling Statements

In JavaScript, an object is like a real-world thing that has properties and actions. For example, think of a car. A car has properties like color, brand, and speed, and it can do actions like start( ) or stop( ). JavaScript objects work the same way. They let you group related data and functions together in one place.

Object handling statements are the tools JavaScript gives us to create objects, access their properties, change their values, or call their methods. You can use dot notation (car.color) or bracket notation (car["color"]) to get or set values. You can also use loops like for...in to go through all properties of an object, or built-in methods like Object. keys() and Object.values() to work with the data easily.

By learning object handling, you can organize your code better, avoid repeating data, and make your programs smarter and more flexible—just like turning raw information into a real-life "thing" you can work with!

2.4.3.1 Creating Objects

Method 1: Using Object Literal

The object literal is the simplest and most common way to create an object in JavaScript. You define an object by placing its properties inside curly braces {}. Each property is written as a key–value pair, separated by a colon, and multiple properties are separated by commas.

Example:

```
let student = {
name: "John",
age: 20,
course: "MCA"
};
```

Here, student is an object with three properties: name, age, and course.

You can access these properties using dot notation (student.name) or bracket notation (student["course"]).

This method is concise, easy to read, and ideal for most cases.

Method 2: Using new Object()

You can also create an object using the new Object() constructor. First, you create an empty object, and then you assign properties to it one by one.

Example:

```
let student = new Object();
student.name = "John";
student.age = 20;
student.course = "MCA";
```

2.4.3.2 Accessing Object Properties

As already mentioned, objects are collections of key–value pairs, where each key represents a property name and the value holds specific information. To work with these objects, we often need to access the values stored inside them. For example, if we create an object called student that contains details such as name, age, and course, we can retrieve these details in two main ways:

- dot notation and
- bracket notation

Both notations allow us to interact with object properties, but they are used in slightly different situations.

Dot notation: This is the most common and straightforward method. You use a dot (.) followed by the property name.

Example:

```
let student = { name: "John", age: 20 };
console.log(student.name);              // Output: John
```

Here, student.name directly returns "John".

Bracket notation: This method uses square brackets ([ ]) with the property name provided as a string.

Example:

```
console.log(student["age"]); // Output: 20
```

Bracket notation is especially useful when the property name is dynamic (stored in a variable) or contains spaces or special characters.

2.4.3.3 Modifying Object Properties

Once an object is created, we can change the values of its properties. This process is called modifying object properties. We can do it either with dot notation or bracket notation.

```
student.age = 21;       // Updating using dot notation
student["course"] = "BCA"; // Updating using bracket notation
```

Here, the age property is updated from 20 to 21, and the course property is updated from "MCA" to "BCA".

2.4.3.4 Deleting Object Properties

Sometimes, we may need to remove a property from an object. This can be done using the delete keyword.

```
delete student.age;  // Removes the 'age' property
```

After this operation, the object will no longer have the property age.

2.4.3.5 Iterating Over Objects

To go through all properties of an object, we can use the for...in loop. This loop iterates over each property name (key) of the object.

```
for (let key in student) {
  console.log(key + ": " + student[key]);
}
```

Here, key will take each property name (like "name", "course", etc.), and student[key] will give the corresponding value.

2.4.3.6 Object Methods

Objects can also have functions as their properties. These functions are called methods. Methods allow objects to perform actions.

Example:

```
let student = {
  name: "John",
  greet: function( ) {
    console.log("Hello " + this.name);
  }
};

student.greet( ); // Output: Hello John
```

In this example, greet is a method of the object student. The keyword this refers to the current object, so this.name points to "John".

Here is a complete JavaScript program that demonstrates all the object handling features you've covered:

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Object Handling Example</title>
</head>
```

```
<body>
  <h2>JavaScript Object Handling Demo</h2>
  <p>Open the console (F12 → Console tab) to see the output.</p>
  <script>
    // Object Handling in JavaScript
    // ----------------------
    // 1. Creating Objects
    // ----------------------
    // Method 1: Using Object Literal
    let student1 = {
      name: "John",
      age: 20,
      course: "MCA",
      // Adding a method inside the object
      greet: function() {
        console.log("Hello " + this.name);
      }
    };
    // Method 2: Using new Object()
    let student2 = new Object();
    student2.name = "Alice";
    student2.age = 22;
    student2.course = "BCA";
    // ----------------------
    // 2. Accessing Object Properties
    // ----------------------
    console.log("Accessing Properties:");
    console.log(student1.name);        // Dot notation → John
    console.log(student1["course"]);   // Bracket notation → MCA
```

```
console.log(student2.name);       // Dot notation → Alice
console.log(student2["age"]);     // Bracket notation → 22
console.log("--------------------------------");
// ----------------------
// 3. Modifying Object Properties
// ----------------------
console.log("Modifying Properties:");
student1.age = 21;                // Update using dot notation
student1["course"] = "BCA";       // Update using bracket notation
console.log(student1);            // Updated student1 object
console.log("--------------------------------");
// ----------------------
// 4. Deleting Object Properties
// ----------------------
console.log("Deleting Properties:");
delete student2.age;              // Remove age property from student2
console.log(student2);            // student2 now has no 'age'
console.log("--------------------------------");
// ----------------------
// 5. Iterating Over Objects
// ----------------------
console.log("Iterating Over student1:");
for (let key in student1) {
  console.log(key + ": " + student1[key]);
}
console.log("--------------------------------");
// ----------------------
// 6. Object Methods
// ----------------------
console.log("Calling Object Method:");
```

```
    student1.greet(); // Method inside object → Hello John
    console.log("--------------------------------");
    // ----------------------
    // 7. Extra: Using Object.keys() and Object.values()
    // ----------------------
    console.log("Using Object.keys() and Object.values():");
    console.log(Object.keys(student1));   // List of property names
    console.log(Object.values(student1)); // List of property values
  </script>
</body>
</html>
```

2.4.4 JavaScript Functions

Functions are among the most important building blocks in JavaScript. They allow developers to group a set of instructions into a reusable block of code. Instead of repeating the same statements multiple times, you can define them once inside a function and call it whenever needed. Functions enhance readability, reduce redundancy, and make programs easier to maintain.

2.4.4.1 Function Declaration

A function declaration defines a named function using the function keyword. Once declared, the function can be called by its name anywhere in the code.

Example:

```
function greet() {
  console.log("Hello World!");
}
greet( ); // Output: Hello World!
```

Here, greet( ) is a function that prints "Hello World!" when called.

2.4.4.2 Function with Parameters

Functions can accept parameters, which act like variables that receive values when the function is called.

Example:

```
function add(a, b) {
  return a + b;
}
console.log(add(5, 3)); // Output: 8
```

The values 5 and 3 are passed to the parameters a and b, and the function returns their sum.

2.4.4.3 Arrow Functions

Arrow functions simplify the syntax of JavaScript functions, providing a shorter and cleaner way to define small or anonymous functions.

Example:

```
let square = (x) => x * x;

console.log(square(6)); // Output: 36
```

This arrow function takes one argument x and returns its square.

2.4.4.4 Function with Default Parameters

Functions can have default parameter values. If a value is not passed for that parameter, the default value is used.

Example:

```
function greet(name = "Guest") {
  console.log("Hello " + name);
}
greet();        // Output: Hello Guest
greet("John");  // Output: Hello John
```

Here, if no argument is given, the parameter name takes the default value "Guest".

2.4.4.5 Return Statement

Functions can return values using the return keyword.

Example:

```
function add(a, b) {
  return a + b;
}
let sum = add(10, 15);
console.log(sum);       // 25
```

Recap

- Branching statements allow programs to make decisions based on conditions.
- if statement executes code only if the condition is true.
- if...else statement chooses between two paths based on condition.
- else if ladder checks multiple conditions sequentially; executes first true condition.
- switch statement executes code based on matching case value; cleaner than multiple if...else if.
- Loops repeat a block of code multiple times.
- for loop is used when the number of iterations is known.
- while loop runs as long as a condition is true; number of iterations may vary.
- break statement exits the loop immediately.
- continue statement skips current iteration, continues next iteration.
- Objects group related data (properties) and functions (methods).
- Objects can be created using object literal {key: value} or new Object() constructor.
- Properties can be accessed using dot notation obj.key or bracket notation obj["key"].
- Object properties can be modified using dot or bracket notation.
- Properties can be deleted using delete obj.key.
- for...in loop can iterate over all object properties.
- Functions inside objects are called methods; this refers to the current object.
- Object.keys(obj) returns property names; Object.values(obj) returns property values.
- Functions group code into reusable blocks to reduce redundancy.
- Function declaration: function name() { }
- Functions can accept parameters to receive input values.
- Arrow functions provide shorter syntax for small functions.
- Functions can have default parameter values if no argument is passed.
- return statement allows a function to return a value.

Objective Type Questions

1. Which statement executes code only if a condition is true?
2. Which statement chooses between two options based on a condition?
3. What is used to check multiple conditions sequentially?
4. Which statement executes code based on matching a case value?
5. Which loop runs a fixed number of times?
6. Which loop runs as long as a condition is true?
7. Which statement exits a loop immediately?
8. Which statement skips the current iteration of a loop?
9. What groups related data and functions in JavaScript?
10. Which notation uses a dot to access object properties?
11. Which notation uses square brackets to access object properties?
12. Which keyword is used to delete an object property?
13. What do we call functions inside objects?
14. Which method lists all property names of an object?
15. Which statement in a function is used to return a value?

Answers to Objective Type Questions

1. if
2. if else
3. else if
4. switch
5. for
6. while
7. break

8. continue
9. object
10. dot
11. bracket
12. delete
13. method
14. Object.keys
15. return

Assignments

1. Write a JavaScript program to check if a person is eligible to vote using an if statement.
2. Create a program that assigns grades (A, B, C, Fail) to students based on their marks using an else if ladder.
3. Write a JavaScript program that prints numbers from 1 to 10 using a for loop and skips the number 5 using continue.
4. Create an object representing a student with properties name, age, and course, and add a method to display a greeting message.
5. Write a function that takes two numbers as parameters and returns their sum; call the function with different values.

Reference

1. Crockford, D. (2008). *JavaScript: The good parts*. O'Reilly Media.
2. Keith, J. (2010). *DOM scripting: Web design with JavaScript and the Document Object Model* (2nd ed.). Friends of ED.
3. Stoyan, S., & Osherove, R. (2016). *JavaScript patterns*. Manning Publications.
4. McFarland, D. S. (2019). *JavaScript: The complete guide to modern JavaScript*. Independently published.

5. Harwani, N. (2020). *Learning JavaScript: A hands-on guide to the fundamentals of modern JavaScript*. Packt Publishing.

Suggested Reading

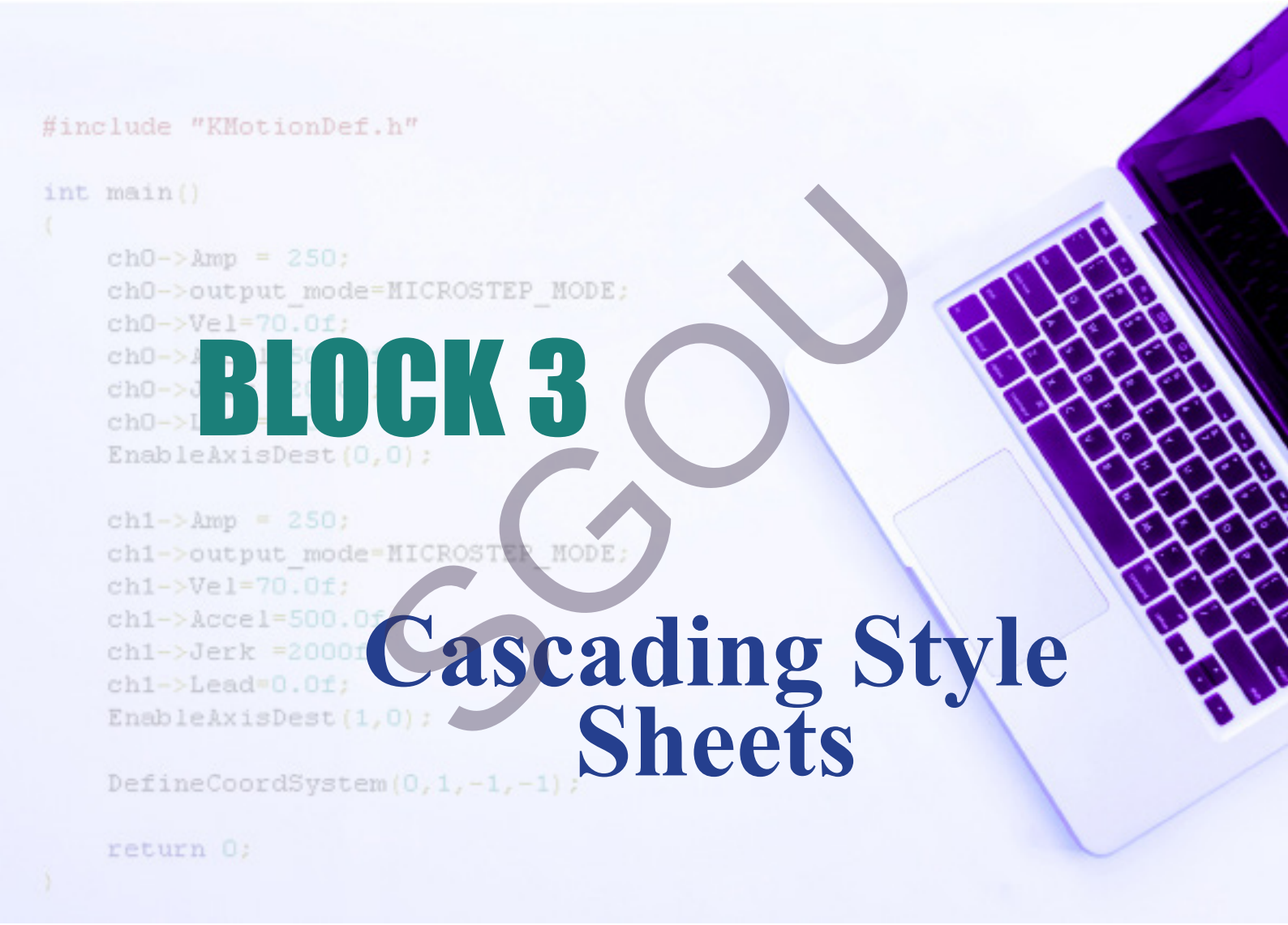
1. Duckett, J. (2014). *JavaScript and jQuery: Interactive front-end web development*. Wiley.

2. Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.

3. Freeman, E., & Robson, E. (2014). *Head First JavaScript programming*. O'Reilly Media.

4. Nixon, R. (2018). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (5th ed.). O'Reilly Media.

5. Resig, J., & Bibeault, B. (2013). *Pro JavaScript techniques*. Apress.



BLOCK 3

Cascading Style Sheets

Introduction to CSS

Learning Outcomes

After completing this unit, learners will be able to:

- understand how CSS helps make web pages look attractive and consistent
- know the three types of CSS : Inline, Internal, External and when to use each one
- use CSS to style different parts of a webpage
- write simple CSS codes by following rules

Prerequisites

It is important to understand why CSS is needed in web development. HTML is used to structure the content of a webpage, such as headings, paragraphs, images, and links, but it does not control how the content looks. Without CSS, web pages would appear plain, unorganized, and unattractive. CSS is used to style and format the content, making websites visually appealing, readable, and user-friendly. For example, designing a newspaper with just plain text on paper would make it hard to read and uninteresting. Editors use different fonts, colors, headings, spacing, and images to make the newspaper attractive and organized. Similarly, CSS acts as the "editor" for a webpage, applying colors, fonts, spacing, alignment, borders, and other design features to make the website professional and engaging. This shows that learning CSS is essential for creating modern and user-friendly websites.

Keywords

Inlizeine CSS, Embedded CSS, External CSS, Selector, Property, Font-size, Text-align

Discussion

Types of CSS (Cascading Style Sheet)

Cascading Style Sheets (CSS) play a crucial role in web development by defining the visual presentation and layout of web pages. While HTML is responsible for structuring the content, CSS enhances the appearance of that content by applying colors, fonts, spacing, alignment, borders, animations, and various other design features. This helps developers create attractive, user-friendly, and professional-looking websites. By separating design from structure, CSS ensures that the website maintains a consistent look and feel across all its pages, making updates and maintenance much easier. Additionally, CSS allows developers to make responsive designs, meaning web pages can automatically adjust their layout according to the screen size of desktops, tablets, and smartphones.

To apply CSS to a webpage, developers can use three different methods:

1. Inline CSS – Styling is applied directly to individual HTML elements using the style attribute. This method is useful for quick styling changes but not suitable for large projects.

2. Internal or Embedded CSS – CSS rules are written inside a <style> tag within the <head> section of an HTML file. This is effective for styling a single page without using external files.

3. External CSS – A separate .css file contains the styling rules, which is linked to an HTML document using the <link> tag. This is the most efficient and recommended method, especially for larger websites, as it ensures consistency across multiple pages and simplifies updates.

By using these three approaches appropriately, developers can manage website styling efficiently and create well-designed, modern web interfaces.

3.1.1 Inline CSS

Inline CSS refers to the method of adding style rules directly within an HTML element by using the style attribute. Instead of writing the styling code in a separate CSS file or inside the <style> section of the webpage, inline CSS attaches the formatting instruction right next to the element it affects. This makes it easy to apply quick and unique styles to specific tags, especially when only a single element needs modification. Inline CSS also has the highest priority in the CSS cascade, meaning it can override both internal and external style rules when conflicts occur. However, because it mixes styling with content and becomes difficult to manage in large projects, inline CSS is typically used for small adjustments or testing purposes rather than full-scale webpage design.

Inline CSS Example:

```
<p style="color:#000900;
        font-size:70px;
```

```
font-style:bold;
        text-align:left;">
  CSS can be Inline, like this
</p>
```

Output

CSS can be Inline, like this

3.1.2 Internal or Embedded CSS

Internal or embedded CSS is a method in which styling rules are written directly within the same HTML file, rather than being placed in a separate stylesheet. These styles are defined inside the <style> tag, which is located in the <head> section of the webpage. By doing so, you can format and design specific elements on that particular page without affecting other pages of the website.

Internal CSS is especially useful when you want to apply consistent styling to a single webpage or when working on small and simple projects. It helps maintain design control within the same file and allows for quick updates without switching between multiple files. However, it may not be the best choice for larger websites with multiple pages, as updating styles across several pages can become time-consuming and inefficient.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal CSS Example</title>
  <!-- Internal CSS starts here -->
  <style>
    body {
      background-color: #f2f2f2; /* Setting page background color */
      font-family: Arial, sans-serif; /* Default font style */
    }
    h1 {
      color: #0066cc; /* Heading text color */
      text-align: center; /* Center align the heading */
```

```
    }
        p {
            font-size: 18px; /* Font size for paragraph */
            color: #333333; /* Text color */
            text-align: justify; /* Justify content */
        }
        .highlight {
            background-color: yellow; /* Highlight text background */
            font-weight: bold; /* Bold text */
        }
    </style>
    <!-- Internal CSS ends here -->
</head>
<body>
    <h1>Internal CSS Demonstration</h1>
```

This example shows how to use Internal CSS to style a webpage. Internal CSS helps apply style rules within the same HTML file using the <style> tag placed in the <head> section.

</body>

This HTML example demonstrates the use of internal CSS, where styles are written inside the <style> tag within the <head> section of the document. The internal CSS controls the appearance of elements on the webpage without needing a separate external stylesheet. In this code, the <body> is styled with a light gray background and a default font to make the page visually clean and readable. The <h1> heading is styled with a blue color and centered alignment to make the title stand out. The paragraph ( ) text is given a medium font size, dark color, and justified alignment to improve readability. A custom CSS class named .highlight is included to add emphasis to specific text inside the paragraph by giving it a yellow background and bold style. This class-based styling allows selective formatting of certain text portions without affecting the entire paragraph. Internal CSS is especially helpful for small webpages or for testing styles quickly before moving to external CSS. Unlike inline CSS, which styles only individual elements, internal CSS allows styling multiple elements efficiently in one place. This method keeps the HTML clean while maintaining styling control inside the same document.

3.1.3 External CSS

External CSS stores all styling rules in a separate file that ends with the .css extension. Instead of writing design properties inside the HTML file, the styles are placed in this external stylesheet, and then connected to the webpage using the <link> tag inside the <head> section. With this method, elements can be styled once using selectors such as class, id, or tag names, and those styles can be reused across multiple web pages. This allows developers to maintain a consistent look throughout an entire website. It also makes the code cleaner, easier to manage, and faster to update, since changes made in the external CSS file automatically reflect on every linked webpage. External CSS is the preferred method for styling large websites due to its flexibility, reusability, and efficient maintenance.

Example:

Index.html file

```
<!DOCTYPE html>
<html>
<head>
   <title>External CSS Example</title>
   <!-- Linking External CSS File -->
   <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
   <h1>External CSS Demonstration</h1>
    <p>This example shows how to use <span class="highlight">External CSS</span> to style a webpage.
   External CSS helps apply styles from a separate file, keeping HTML clean and organized.</p>
</body>
</html>
```

Style.css file for external CSS

```
/* Page background and font */
body {
   background-color: #f2f2f2; /* Setting page background color */
```

```
font-family: Arial, sans-serif; /* Default font style */
}
/* Heading styling */
h1 {
    color: #0066cc; /* Heading text color */
    text-align: center; /* Center align the heading */
}
/* Paragraph styling */
p {
    font-size: 18px; /* Font size for paragraph */
    color: #333333; /* Text color */
    text-align: justify; /* Justify content */
}
/* Highlighted text style */
.highlight {
    background-color: yellow; /* Highlight text background */
    font-weight: bold; /* Bold text */
}
```


This code demonstrates how to apply external CSS to style a webpage by separating design from content. The HTML file starts with the <!DOCTYPE html> declaration and includes a <head> section where the external CSS file style.css is linked using the <link> tag. This link instructs the browser to load styles from a separate file instead of writing them inside the HTML. The <body> contains a heading and paragraph to display text on the page. In the CSS file, the body selector styles the entire webpage by setting a light background color and defining a clean font. The h1 selector changes the heading color to blue and centers it for better appearance. The p selector adjusts the paragraph text size, color, and aligns the content for a neat look. Additionally, the .highlight class adds a yellow background and makes text bold to emphasize certain words. This structure shows how external CSS promotes cleaner HTML and easier maintenance, especially for large websites.

3.1.4 Usage of different types of CSS

Inline CSS is most useful when you want to make quick and simple style changes to a single element without modifying other parts of the webpage. It is helpful when

you need to override existing styles temporarily or apply a one-time visual adjustment. Inline CSS is also commonly used in email templates or certain HTML-based systems where linking external stylesheets is not supported. However, it is not ideal for styling multiple elements because it can make the code messy and harder to maintain.

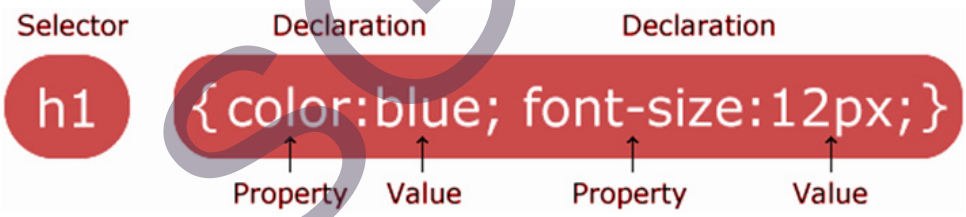
Internal CSS is suitable when working on a single-page website or a small project where using external files is unnecessary. This method allows centralized control over styles within the same HTML file, making it easier to apply consistent design rules without repeating inline styles. It offers greater flexibility and better organization than inline CSS, though it may still be inefficient for large projects involving multiple pages

External CSS is the best choice for medium to large web projects, especially when multiple pages require the same styling. By linking an external .css file, you can maintain a clean separation between content and design, making the code easier to update and manage. This method also improves website performance because the CSS file can be cached by browsers, reducing load time. External CSS ensures scalability, consistency, and professional structure in modern web development.

External CSS is the most efficient option for styling websites with multiple pages. It provides uniform styling across all pages, simplifies updates by modifying a single file, and enhances loading speed through caching. With better structure and easier maintenance, external CSS is considered the standard practice for modern web development, especially for medium and large-scale projects that prioritize efficiency, scalability, and clean code organization.

Syntax:

A CSS rule consists of a selector and a declaration block:



A CSS rule is made up of two main parts: a selector and a declaration block. The selector identifies the HTML element or group of elements that you want to style. The declaration block contains one or more style declarations, each defining how the selected element should look.

Each individual declaration consists of a property (such as color, font-size, margin, or background) and a corresponding value (such as red, 20px, 10px, or blue), separated by a colon. Multiple declarations are enclosed within curly braces { } and separated by semicolons ;. This structure enables developers to apply multiple styles to a single element.

CSS rules can control a wide range of visual properties including text formatting, colors, spacing, borders, backgrounds, positioning, and layout. For example, you can adjust font size, center content, add padding, or apply background colors to improve the look and feel of a webpage.

Example CSS Rule

The rule below styles all <p> elements so that the text appears in red and is centered in the page:

```
p {
  color: red;
  text-align: center;
  font-size: 18px;
  margin-top: 10px;
}
```

In this example:

- p is the selector
- color, text-align, font-size, and margin-top are CSS properties
- red, center, 18px, and 10px are the values

This shows how multiple styling instructions can be applied to the same HTML element using a single CSS rule.

Recap

- CSS (Cascading Style Sheets) defines the visual presentation and layout of web pages.
- CSS enhances HTML by styling colors, fonts, spacing, alignment, borders, and animations.
- CSS separates design from structure, ensuring consistent look and easier maintenance.
- CSS allows web pages to adjust layouts for desktops, tablets, and smartphones.
- There are three types of CSS: Inline, Internal (Embedded), and External.
- Inline CSS applies styles directly to an HTML element using the style attribute and has the highest priority in the cascade.
- Inline CSS is best for quick, one-time changes or testing small adjustments.
- Internal CSS is written inside a <style> tag in the HTML <head> section and affects only that page.
- Internal CSS is suitable for single-page websites or small projects and provides easier control than inline CSS.

- External CSS stores styles in a separate .css file and links it via <link>, allowing consistent styling across multiple pages.
- External CSS keeps HTML clean, allows reusable styles, simplifies maintenance, and improves performance through caching.
- A CSS rule consists of a selector that targets HTML elements and a declaration block {property: value;}.
- Multiple declarations in a CSS rule can control color, size, spacing, layout, and other properties for an element.
- Selectors can be tag-based, class-based, or ID-based to target specific elements or groups.
- Inline CSS is used sparingly, internal CSS for small projects, and external CSS is preferred for medium to large websites for efficiency and scalability.

Objective Type Questions

1. Which type of CSS is applied directly to an HTML element using the style attribute?
2. Which CSS type is written inside the <style> tag in the <head> section of an HTML page?
3. Which CSS type is stored in a separate .css file and linked to an HTML document?
4. Which CSS type has the highest priority in the cascade?
5. Which CSS method is most suitable for styling multiple pages of a website consistently?
6. Inline CSS is mainly used for _____ styling changes.
7. Internal CSS is ideal for _____-page websites.
8. External CSS improves website _____ by caching the CSS file.
9. What symbol is used to enclose a CSS declaration block?
10. In a CSS declaration, what separates a property from its value?
11. In a CSS rule, the part that identifies the HTML element is called the _____.

12. In the rule p { color: red; text-align: center; }, “red” is a CSS _____.

13. Which CSS type is generally not recommended for large projects?

14. The <link> tag is used to include _____ CSS.

15. A CSS rule is composed of a selector and a _____ block.

Answers to Objective Type Questions

1. Inline
2. Internal
3. External
4. Inline
5. External
6. Quick
7. Single
8. Performance
9. Braces
10. Colon
11. Selector
12. Value
13. Inline
14. External
15. Declaration

Assignments

1. Explain the role of CSS in web development.
2. Compare and contrast Inline, Internal, and External CSS.
3. Describe Inline CSS with an example.
4. Explain Internal CSS and its usage.
5. Explain External CSS and its benefits.
6. Write a CSS rule and explain its components.
7. Discuss the practical usage of different types of CSS in web development.
8. Demonstrate how multiple style properties can be applied to a single HTML element.

Reference

1. Negrino, T., & Smith, D. (2011). *JavaScript and Ajax for the web: Visual QuickStart guide* (8th ed.). Peachpit Press.
2. Felke-Morris, T. (2020). *Web development and design foundations with HTML5* (9th ed.). Pearson.
3. McFarland, D. S. (2015). *CSS: The missing manual* (4th ed.). O'Reilly Media.

Suggested Reading

1. Duckett, J. (2011). *HTML and CSS: Design and build websites*. Wiley.
2. Nixon, R. (2021). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (6th ed.). O'Reilly Media.
3. Castro, E., & Hyslop, B. (2013). *HTML5 and CSS: Visual QuickStart guide* (8th ed.). Peachpit Press.
4. Freeman, E. (2012). *Head First HTML and CSS* (2nd ed.). O'Reilly Media.

Types of Selectors

Learning Outcomes

After completing this unit, learners will be able to:

- explain the purpose and importance of CSS selectors in connecting HTML structure with webpage styling
- differentiate between various types of CSS selectors such as universal, element, class, ID, and attribute selectors
- illustrate the use of pseudo-classes and pseudo-elements in creating interactive and visually appealing web designs
- apply different CSS selectors to style specific HTML elements effectively in a webpage

Prerequisites

Before learning about the types of selectors in CSS, it is important to understand why selectors are needed. A webpage usually contains many HTML elements such as headings, paragraphs, links, buttons, and images. To make the webpage visually appealing, developers need to apply different styles to these elements. However, not all elements require the same style. CSS selectors help identify and target specific elements or groups of elements. They act as a bridge between the HTML content and CSS styles. Selectors tell the browser which elements should receive which styles. This allows developers to style multiple elements efficiently. Without selectors, each element would need to be styled individually, which is time-consuming. For example, consider decorating a classroom. All students may receive white shirts. Class monitors may receive blue shirts. Sports team members may receive green shirts. In this example, “all students” is like a universal selector. “Class monitors” are like a class selector. “Sports team members” are like an ID selector. Just as the teacher assigns different uniforms to different groups, CSS selectors assign different styles to HTML elements. Selectors make styling organized and manageable. They save time and ensure consistency across a webpage. Understanding selectors is essential before learning their types and applications in CSS.

Keywords

Universal Selectors, Element, Class, ID, Attribute, Pseudo-class, Pseudo-element

Discussion

CSS selectors are the fundamental building blocks that determine how styles are applied to specific HTML elements within a webpage. They serve as the connection between structure (HTML) and presentation (CSS), instructing the browser on exactly which elements should receive particular visual properties. Through the use of selectors, developers can efficiently modify the layout, colors, fonts, spacing, and overall appearance of a webpage. Without selectors, CSS would have no way of identifying which parts of a webpage to style, resulting in disorganized and inconsistent designs. In essence, selectors provide precision and control in web development, enabling designers to create visually appealing and well-structured websites.

Selectors in CSS are categorized based on the way they identify and target HTML elements. Simple selectors are the most basic type they select elements using their tag name (like <p>), class (.classname), or ID (#idname). Combinator selectors extend this by selecting elements based on their relationship with others in the document structure, such as parent-child or sibling relationships, enabling complex and context-specific styling. Pseudo-class selectors apply styles when an element is in a particular state, like when a user hovers over a link or focuses on an input field. Pseudo-element selectors, on the other hand, allow developers to style specific parts of an element, such as the first letter of a paragraph or the first line of text. Lastly, attribute selectors target elements based on their attributes or attribute values, for example, styling all input fields of a certain type.

Together, these categories of selectors make CSS a powerful and flexible language. They enable developers to control not just how content looks, but also how it behaves under various conditions, ensuring both functionality and aesthetic appeal in modern web design.

3.2.1 Universal Selector

The CSS Universal Selector is a powerful tool used to apply a common style to all elements within a webpage. It is represented by an asterisk symbol *, which acts as a wildcard that targets every HTML element in the document—regardless of its tag name, class, or ID.

For instance, the following CSS rule applies the same styling to all elements on the page:

```
* {
  text-align: center;
  color: blue;
}
```

In this example, every element such as headings, paragraphs, links, and divs—will have its text centered and colored blue.

The universal selector is particularly useful when you want to set consistent base styles across a webpage, such as resetting margins, padding, or font properties. However, it should be used carefully in large projects since applying styles to every element may impact performance or override specific styles unintentionally.

3.2.2 Element Selector

The element selector in CSS is one of the most basic and commonly used selectors. It allows developers to apply styles to all instances of a particular HTML element throughout a webpage by simply referring to the element's tag name. This means that if you want to style every paragraph, heading, or table in the same way, you can do so easily using this selector without needing to assign a class or ID.

Another name for the element selector is the type selector.

Both terms refer to the same concept , selecting HTML elements based on their tag name (like p, h1, div, span, etc.).

For example:

```
p {
  color: blue;
}
```

Here, the selector p (or type selector) targets all paragraph elements on the web page.

The element selector ensures consistency by applying uniform formatting to all elements of that type. For instance, you might want all paragraphs to have the same text color, font size, or alignment. By defining the style once in CSS, it automatically applies to every <p> element in the document, reducing repetition and improving efficiency.

Here's an example that demonstrates how it works. The following CSS rule targets all <p> (paragraph) tags in an HTML document and formats the text to appear centered with a red color:

```
p {
  text-align: center;
  color: red;
}
```

In this example, every paragraph within the webpage will be center-aligned and displayed in red text, ensuring a consistent and visually appealing layout. The element

selector is particularly useful in maintaining a clean and uniform design, especially for styling basic HTML tags such as headings, paragraphs, and lists across multiple sections of a website.

3.2.3 The CSS Class Selector

The CSS class selector is used to style one or more HTML elements that share the same class attribute. It provides flexibility by allowing developers to apply the same set of styles to multiple elements on a webpage, making the design consistent and easier to manage. Unlike the ID selector, which is unique to a single element, a class can be reused any number of times within a document.

To define a class selector in CSS, a dot (.) is placed before the class name. This tells the browser that all elements containing that class should be styled according to the defined rules.

For example, the following CSS rule targets all elements that have the class name "center" and applies red-colored text that is aligned to the center of the page:

```
.center {
  text-align: center;
  color: red;
}
```

This means that any HTML tag, such as a paragraph <p>, a heading <h1>, or a division , assigned with class="center" will automatically adopt the specified styles.

The class selector is especially useful for creating reusable design components across different parts of a website. It supports clean, modular styling and is widely used in modern front-end frameworks and responsive web design. Additionally, class selectors can be combined with other selectors, such as element selectors or pseudo-classes, to create more specific and powerful style rules.

In the below example, the styling is applied only to paragraph (<p>) elements that have the class name "center". These elements will display their text in red color and be aligned to the center of the page.

```
p.center {
  text-align: center;
  color: red;
}
```

This means that even if other elements share the same class name "center", the defined style will not affect them. It will apply only to tags that carry this class. This method allows developers to combine element selectors and class selectors for more precise control over which elements receive specific styling rules.

HTML elements can be assigned multiple classes at the same time to apply different styles together.

For example, in the following code, the paragraph ( ) element uses both the "center" and "large" classes. As a result, it inherits the styling rules defined for both classes:

This paragraph applies two different class styles.

By combining multiple classes, developers can reuse existing CSS rules efficiently and create flexible, modular styling for elements without duplicating code.

3.2.4 ID Selector

The ID selector in CSS is used to style a specific HTML element that possesses a unique id attribute. Unlike class selectors, which can be reused across multiple elements, an ID should be assigned to only one element on a webpage. This makes the ID selector particularly useful when you need to apply a distinctive style to a single component, such as a header, footer, or special message box without influencing other elements.

To define an ID selector in CSS, a hash symbol (#) is placed before the ID name. The style rules written under this selector will apply only to the element that matches the given ID. For instance, the following example styles the element with id="para1" so that its text is centered and displayed in red color:

```
#para1 {
  text-align: center;
  color: red;
}
```

Using ID selectors improves code precision, allowing developers to make specific adjustments to unique elements without altering the overall design. However, overusing IDs for styling is not recommended, as it can limit flexibility and make code harder to maintain especially in large projects.

Apart from ID selectors, CSS provides other types of selectors such as class selectors (for styling multiple elements), universal selectors (which apply to all elements), and group selectors (for combining multiple selectors in one rule). In modern web design, these are often combined with advanced selectors and responsive design techniques to create flexible, adaptive layouts.

A strong understanding of selectors, including ID selectors, is essential for front-end developers to write clean, scalable, and efficient CSS that ensures consistent styling and easy maintenance across web pages.

3.2.5 Group Selector

The CSS Group Selector, also known as a Grouping Selector, is used when multiple HTML elements share the same styling properties. Instead of writing separate CSS rules for each element, you can combine them into a single rule to make your code cleaner, shorter, and easier to maintain.

For example, consider this CSS code where the <h1>, <h2>, and <p> elements all have identical styling:

```
h1 {
  text-align: center;
  color: red;
}
h2 {
  text-align: center;
  color: red;
}
p {
  text-align: center;
  color: red;
}
```


Writing repetitive code like this can make your stylesheet unnecessarily long. To simplify it, you can group the selectors by separating them with commas:

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

This grouped rule applies the same style to all three elements at once. Group selectors are particularly useful for maintaining consistency in web design and improving the readability of your CSS. They also help reduce file size and make future updates easier since changes can be made in one place rather than multiple lines of code.

3.2.6 Attribute Selectors

CSS attribute selectors allow you to select and style HTML elements based on the presence or value of their attributes. They are especially useful when you want to target elements dynamically without assigning additional classes or IDs.

Below are the various types of CSS attribute selectors and their uses.

1. [attribute] Selector

This selector targets all elements that contain a specific attribute, regardless of its value.

Example:

```
a[target] {
  background-color: yellow;
}
```

This styles all <a> elements that have a target attribute.

2. [attribute="value"] Selector

This selector targets elements with an attribute that exactly matches a specified value.

Example:

```
a[target="_blank"] {
  background-color: yellow;
}
```

This applies styling to <a> elements where the target attribute is exactly _blank.

3. [attribute~="value"] Selector

This selector matches elements whose attribute value contains a specific word within a space-separated list.

Example:

```
[title~="flower"] {
  border: 5px solid yellow;
}
```

This matches elements with titles like "flower", "summer flower", or "flower new", but not "my-flower" or "flowers".

4. [attribute|="value"] Selector

This selector targets elements where the attribute's value is either an exact match or begins with the given value followed by a hyphen (-).

Note: The value must be a complete word (e.g., class="top" or class="top-text").

Example:

```
[class|="top"] {
  background: yellow;
}
```

This will select elements with class="top" or class="top-section".

5. [attribute^="value"] Selector

This selector matches elements whose attribute value begins with a specific string.

Example:

```
[class^="top"] {
  background: yellow;
}
```

This applies styles to all elements with a class name starting with "top" such as "topbar" or "topmenu".

6. [attribute$="value"] Selector

This selector targets elements whose attribute value ends with a specific string.

Example:

```
[class$="test"] {
  background: yellow;
}
```

This will match elements with class values like "unittest" or "mytest".

7. [attribute*="value"] Selector

This selector matches elements whose attribute value contains a specific substring anywhere within it.

Example:

```
[class*="te"] {
  background: yellow;
}
```

This selects all elements whose class name includes "te", such as "test", "note", or "late".

Styling Form Elements Using Attribute Selectors

Attribute selectors are extremely useful for targeting form elements based on their type.

Example:

```
input[type="text"] {
  width: 150px;
  padding: 6px;
  margin-bottom: 10px;
  background-color: pink;
}
input[type="button"] {
  width: 100px;
  padding: 6px;
  background-color: lightgreen;
}
```

Here:

- Text input fields are styled with pink backgrounds and custom padding.
- Button elements receive a light green background with adjusted width.

CSS attribute selectors — such as [attribute], [attribute="value"], [attribute~="value"], [attribute|="value"], [attribute^="value"], [attribute$="value"], and [attribute*="value"] — provide flexible and powerful ways to style elements based on their attributes and attribute values without modifying HTML structure.

3.2.7 CSS Pseudo Classes

A CSS pseudo class is a keyword added to a selector to apply styles to an element in a specific state or condition. It allows developers to change the appearance of elements based on user actions or their position within the HTML structure.

Pseudo-classes are particularly useful for creating interactive and dynamic web designs without the need for JavaScript. They define styles that automatically change in response to specific user actions or element states.

Some common uses of pseudo classes include:

- Styling an element when a user hovers the mouse over it.
- Displaying different colors for visited and unvisited links.
- Highlighting an element when it receives keyboard focus.
- Customizing input fields based on their validation state (valid, invalid, required, or optional).
- Styling an element that appears as the first child of its parent container.

Syntax:

A pseudo class is written by adding a colon (:) after the selector, followed by the pseudo class name.

Example:

```
selector:pseudo-class-name {
  /* CSS properties */
}
```

Pseudo classes add flexibility and interactivity to web design, enhancing user experience without complex scripts.

CSS :hover Pseudo Class

The :hover pseudo class is used to apply a style to an element when the user places the mouse pointer over it. This feature helps create interactive and visually engaging effects, commonly used for buttons, links, and menus.

CSS Syntax

```
:hover {
  css declarations;
}
```

Example:

In the example below, when the user moves the mouse over a hyperlink, its background color changes to yellow and the text size increases to 18 pixels.

```
a:hover {
  background-color: yellow;
  font-size: 18px;
}
```

This makes the element respond visually to user interaction, improving the overall usability and design experience of the webpage.

CSS :focus Pseudo-class

The :focus pseudo-class is used to apply a specific style to an element when it becomes active or is selected by the user—for example, when a user clicks inside an input box or navigates to it using the keyboard. This helps indicate which element is currently ready for user interaction.

CSS Syntax

```
:focus {
  css declarations;
}
```

Example 1:

In the example below, the input field's background color changes to yellow when it receives focus.

```
input:focus {
  background-color: yellow;
}
```

Example 2:

Change background color and width when an input field gets focus:

```
input:focus {
  background-color: yellow;
  width: 250px;
}
```

This visual effect enhances the user experience by clearly showing which field is active, making web forms more intuitive and accessible, especially for users navigating with the keyboard.

Pseudo Classes for Links

Pseudo classes can be applied to hyperlinks in HTML to change their appearance based on how users interact with them. These allow developers to visually differentiate between links that have been visited, hovered over, or clicked. The most commonly used link-related pseudo classes include:

- :link – Styles links that have not yet been visited.
- :visited – Styles links that have already been visited.
- :hover – Styles links when the mouse pointer is placed over them.
- :active – Styles links at the moment they are clicked or activated.

Example:

The following example displays links in different colors depending on their state:

```
/* Unvisited link */
a:link {
  color: #FF0000;
}
/* Visited link */
a:visited {
  color: #00FF00;
}
/* Mouse over link */
a:hover {
  color: #FF00FF;
}
/* Active or selected link */
a:active {
  color: #0000FF;
}
```

Important Notes:

- The order of pseudo classes in the CSS code matters for proper styling.
  - a:hover should come after a:link and a:visited.
  - a:active should come after a:hover.
- Pseudo class names are not case sensitive.

Using these link pseudo classes enhances user experience by providing clear visual feedback when navigating through hyperlinks on a webpage.

3.2.8 CSS Pseudo-Elements

A CSS pseudo-element is a special keyword added to a selector that allows you to style a specific portion of an element rather than the entire element.

Common Uses of Pseudo-Elements

Pseudo-elements are often used to:

- Apply styles to the first letter or first line of text.
- Insert content before or after an element.
- Customize the markers in list items.
- Style the text selected by the user.
- Modify the area behind elements such as dialog boxes.

Syntax

A pseudo-element is written using a double colon (::) followed by its name:

```
selector::pseudo-element {
  /* CSS properties */
}
```

Example: ::first-line Pseudo-Element

The ::first-line pseudo-element is used to apply unique styles to the first line of text within a block-level element.

Note: The ::first-line pseudo-element only works on block-level elements such as <p>, , or <section>.

Example Code:

```
p::first-line {
  color: red;
  font-variant: small-caps;
  font-size: 19px;
}
```

This example changes the color, font variant, and font size of the first line of text in all <p> elements.

1. The ::first-letter Pseudo-element

The ::first-letter pseudo-element applies unique styling to the first letter of a block of text.

Note: It can only be used with block-level elements such as <p> or .

Example:

```
p::first-letter {
  color: red;
  font-size: xx-large;
}
```

This code changes the first letter of each paragraph to red and enlarges its size.

2. The ::before Pseudo-element

The ::before pseudo-element inserts content before the actual content of an element. You must use the content property to define what is inserted.

Example:

```
h3::before {
  content: url(smiley.gif);
}
```

This adds an image before the content of every <h3> heading.

3. The ::after Pseudo-element

The ::after pseudo-element inserts content after the element's content.

Like ::before, it also requires the content property.

Example:

```
h3::after {
  content: url(smiley.gif);
}
```

This places an image after the text of each <h3> element.

4. The ::marker Pseudo-element

The ::marker pseudo-element styles the bullet or numbering marker of list items in ordered or unordered lists.

Example:

```
::marker {
  color: red;
  font-size: 23px;
}
```

This changes the color and size of list item markers.

5. The ::selection Pseudo-element

The ::selection pseudo-element is used to style the portion of text selected by the user.

Example:

```
::selection {
  color: red;
  background: yellow;
}
```

This gives the selected text a red color and a yellow background.

6. The ::backdrop Pseudo-element

The ::backdrop pseudo-element styles the background area (viewbox) that appears behind dialog boxes or popovers.

Example:

```
dialog::backdrop {
  background-color: lightgreen;
}
```

This changes the background behind a dialog box to light green.

7. Combining Pseudo-elements with Classes

Pseudo-elements can be used along with HTML classes to target specific elements more precisely.

Example:

```
p.intro::first-letter {
  color: #ff0000;
  font-size: 200%;
}
```

Here, only the first letter of paragraphs with the class intro will appear in red and be twice the normal size.

8. Using Multiple Pseudo-elements Together

You can apply more than one pseudo-element to an element to create layered styles.

Example:

```
p::first-letter {
  color: red;
  font-size: xx-large;
}

p::first-line {
  color: blue;
  font-variant: small-caps;
}
```

In this example:

- The first letter is styled in red and an extra-large font.
- The rest of the first line appears in blue and small caps.
- The remaining text of the paragraph retains the default style.

CSS pseudo-elements such as ::first-letter, ::before, ::after, ::marker, ::selection, and ::backdrop enhance the presentation of web pages by targeting and styling specific portions of elements or by inserting additional content without altering the HTML structure.

Recap

- CSS selectors are used to apply styles to specific HTML elements.
- They connect HTML structure with CSS presentation.
- Simple selectors include element, class, and ID selectors.
- Combinator selectors define relationships like parent-child or sibling.
- Pseudo-classes apply styles based on an element's state or user action.
- Pseudo-elements style specific parts of an element such as the first line or letter.
- The universal selector (*) applies a common style to all elements on a webpage.
- The element (type) selector targets all instances of a particular HTML tag.
- The class selector (.) allows multiple elements to share the same styling.
- The ID selector (#) targets a unique element with a specific ID.
- The group selector combines multiple selectors with commas to reduce redundancy.
- Attribute selectors style elements based on attribute presence or values.
- Pseudo-classes like :hover, :focus, :link, and :visited add interactivity.
- Pseudo-elements such as ::before and ::after can insert extra content visually.
- Understanding and combining selectors improve CSS efficiency, consistency, and maintainability.

Objective Type Questions

1. Which symbol represents the universal selector in CSS?
2. What is another name for the element selector?
3. Which character is used before a class name in CSS?
4. Which character is used before an ID name in CSS?
5. Which selector is used to apply the same style to multiple elements at once?
6. Which selector targets elements based on their attributes?

7. What pseudo-class is used when a user hovers over an element?
8. What pseudo-class is used when an element gains focus?
9. Which pseudo-class styles unvisited links?
10. Which pseudo-class styles visited links?
11. Which pseudo-element styles the first letter of a block of text?
12. Which pseudo-element inserts content before an element's actual content?
13. Which pseudo-element inserts content after an element's actual content?
14. Which pseudo-element styles the bullet marker of list items?
15. Which pseudo-element styles the text selected by a user?

Answers to Objective Type Questions

1. *
2. Type Selector
3. .
4. #
5. Group
6. Attribute
7. :hover
8. :focus
9. :link
10. :visited
11. ::first-letter
12. ::before
13. ::after
14. ::marker
15. ::selection

Assignments

1. Explain the importance of CSS selectors in web development and how they link HTML structure with visual presentation.

2. Describe and differentiate between element selectors, class selectors, and ID selectors with suitable examples.

3. Explain the use of the universal selector in CSS. Provide an example where applying a universal style would be useful.

4. Discuss the purpose and advantages of group selectors in CSS. Illustrate with an example.

5. Explain CSS attribute selectors and their different types. Provide examples for at least three attribute selectors.

6. Describe CSS pseudo-classes and their practical use in enhancing user interaction on web pages. Include examples of :hover and :focus pseudo-classes.

7. Explain CSS pseudo-elements and their common applications. Illustrate your answer with examples like ::first-letter, ::before, and ::after.

8. Discuss how combining pseudo-elements with classes or using multiple pseudo-elements together can improve the styling and presentation of a webpage. Provide examples.

Reference

1. Negrino, T., & Smith, D. (2011). *JavaScript and Ajax for the web: Visual QuickStart guide* (8th ed.). Peachpit Press.

2. Felke-Morris, T. (2020). *Web development and design foundations with HTML5* (9th ed.). Pearson.

3. McFarland, D. S. (2015). *CSS: The missing manual* (4th ed.). O'Reilly Media.

Suggested Reading

1. Duckett, J. (2011). *HTML and CSS: Design and build websites*. Wiley.

2. Nixon, R. (2021). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (6th ed.). O'Reilly Media.

3. Castro, E., & Hyslop, B. (2013). *HTML5 and CSS: Visual QuickStart guide* (8th ed.). Peachpit Press.

4. Freeman, E. (2012). *Head First HTML and CSS* (2nd ed.). O'Reilly Media.

Colors and Background

Learning Outcomes

After completing this unit, learners will be able to:

- remember different CSS properties used for text formatting
- explain how to use text color and background color in web pages
- apply CSS properties to align text, change its style, and add decoration
- use CSS font properties to change the look and size of text
- understand the box model and styling elements

Prerequisites

Colors and backgrounds play a crucial role in web design. They help make websites visually appealing and engaging for users. Without proper use of colors, a website can appear dull and unattractive. Backgrounds provide context and structure to the layout of a webpage. They can separate different sections, making content easier to read. Text color must contrast with the background to ensure readability. CSS allows developers to apply consistent colors across all web pages. This consistency helps maintain a professional and unified look. Colors can also convey emotions and themes. For example, green is often associated with nature, while red can indicate urgency or importance. Background images or gradients can enhance the overall design and create a more immersive experience. In a real-life scenario, a restaurant website can use a beige background to create a warm atmosphere. Special offers can be highlighted with bright colors to attract attention. CSS makes it easy to change colors and backgrounds without altering the HTML structure. Using colors and backgrounds will effectively improve usability, readability, and user engagement.

Keywords

Background-color, Text-align, Text-decoration, Text-transform, Font-family, Font-size, Line-height

Discussion

3.3.1 Text Color and Formatting

CSS provides a wide range of properties that enable developers to style and format text, thereby improving readability and enhancing the visual appeal of web pages. Text formatting in CSS involves adjusting alignment, spacing, transformation, and color. For example, text can be aligned to the center, transformed to uppercase, or spaced out by modifying the letter-spacing property. Even hyperlinks can be customized for instance, by removing the default underline and changing their color to match the overall design theme.

Text Color

The color property in CSS is used to define the color of text on a web page. There are several ways to specify a color value:

- Color names – e.g., "red", "blue", or "green"
- HEX values – e.g., #ff0000 for red
- RGB values – e.g., rgb(255, 0, 0) for red

You can refer to the full list of color options in the CSS color values chart.

Generally, the default text color of an entire webpage is set using the body selector. You can then customize text colors for specific elements such as headings or paragraphs.

Example:

```
body {
  color: blue;    /* Sets default text color to blue */
}
h1 {
  color: green;   /* Changes heading 1 text to green */
}
h2 {
  color: red;     /* Changes heading 2 text to red */
}
```

In this example, the main body text will appear blue, while headings have individual colors. This simple yet powerful CSS property allows designers to create a visually consistent and attractive website layout.

3.3.1.1 Text and Background Colors in CSS

In CSS, you can style web content by setting both the text color and the background color of elements. The color property defines the color of the text, while the background-

color property determines the color behind it. Using these properties together helps create visually appealing and easy-to-read web designs.

Example:

```
body {
  background-color: lightgrey;  /* Sets the page background to light grey */
  color: blue;                  /* Sets all text to blue */
}
h1 {
  background-color: black;      /* Adds a black background for headings */
  color: white;                 /* Changes heading text color to white */
}
div {
  background-color: blue;       /* Gives div elements a blue background */
  color: white;                 /* Sets the text color to white */
}
```

In this example, the background and text colors are chosen to make each section stand out clearly. For example, the white text on a black or blue background creates a strong contrast, making the content easy to read.

Note: Maintaining high contrast between text and background is essential for accessibility. This ensures that users, including those with visual impairments, can comfortably read the content. Always choose color combinations that provide clear visibility and readability.

3.3.1.2 Text Alignment and Direction

This section explains how to control text alignment and text direction in CSS using several key properties.

Text Alignment

The text-align property determines how text is horizontally aligned within an element.

Possible values include:

- left – aligns text to the left.
- right – aligns text to the right.
- center – centers the text horizontally.
- justify – adjusts spacing so that each line has equal width, aligning both left and right margins (similar to text in newspapers or magazines).

Example:

```
h1 {
  text-align: center;
}
h2 {
  text-align: left;
}
h3 {
  text-align: right;
}
```

Note: For left-to-right text, the default alignment is left, and for right-to-left text, it is right.

To justify text so that it spans the full width of a container:

```
div {
  text-align: justify;
}
```

Text Align Last

The text-align-last property defines how the last line of a text block is aligned.

Available values:

- auto – default; usually justifies the text and aligns it left.
- left – aligns the last line to the left.
- right – aligns the last line to the right.
- center – centers the last line.
- justify – stretches the last line so that it is fully justified.
- start – aligns the last line to the beginning of the line (based on text direction).
- end – aligns the last line to the end of the line (based on text direction).

Example:

```
p.a {
  text-align-last: right;
}
p.b {
```

```
  text-align-last: center;
}
p.c {
  text-align-last: justify;
}
```

Each paragraph in the example demonstrates a different alignment style for the final line of text, showing how this property can fine-tune text layout and readability.

Vertical Alignment

The vertical-align property controls how an element is positioned vertically relative to its parent or surrounding elements.

Possible Values

- baseline – Default. Aligns the element with the baseline of its parent.
- length / % – Moves the element up or down by a specific length or percentage.
- sub – Aligns the element with the subscript position of the parent text.
- super – Aligns the element with the superscript position of the parent text.
- top – Aligns the element with the top of the tallest element in the line.
- text-top – Aligns the element with the top of the parent's font.
- middle – Positions the element in the middle of the parent element.
- bottom – Aligns the element with the lowest element in the line.
- text-bottom – Aligns the element with the bottom of the parent's font.

Example

The following example demonstrates different vertical alignments of an image within text:

```
img.a {
  vertical-align: baseline;
}
img.b {
  vertical-align: text-top;
}
img.c {
  vertical-align: text-bottom;
```

```
}
img.d {
  vertical-align: sub;
}
img.e {
  vertical-align: super;
}
```

3.3.1.3 Text Direction

The direction property defines the direction in which text is written within a block-level element.

It is often used alongside the unicode-bidi property to properly display content that includes multiple languages with different writing directions.

Example:

```
p {
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

This example sets the text direction to right-to-left (rtl) and uses bidi-override to ensure proper display of languages like Arabic or Hebrew within the same document.

Text Decoration

The text-decoration property in CSS is used to add, style, or remove decorative lines applied to text. It helps enhance the visual presentation of web content by underlining, overlining, or striking through text. This property acts as a shorthand for several individual properties:

- text-decoration-line
- text-decoration-color
- text-decoration-style
- text-decoration-thickness

Adding Decorative Lines to Text

The text-decoration-line property determines the type of decoration applied. You can use one or more of the following values:

- none – removes any decorative line (default).

- underline – places a line below the text.
- overline – adds a line above the text.
- line-through – strikes a line through the text.

You can even combine multiple styles (like underline and overline) for emphasis.

Example:

```
h1 {
  text-decoration-line: overline;
}
h2 {
  text-decoration-line: line-through;
}
h3 {
  text-decoration-line: underline;
}
p {
  text-decoration-line: overline underline;
}
```

Note: Avoid underlining non-link text, as it can confuse readers into thinking it's a clickable link.

Changing the Color of Decoration Lines

The text-decoration-color property allows you to set a specific color for the decoration line.

Example:

```
h1 {
  text-decoration-line: overline;
  text-decoration-color: red;
}
h2 {
  text-decoration-line: line-through;
  text-decoration-color: blue;
}
h3 {
  text-decoration-line: underline;
```

```
  text-decoration-color: green;
}
p {
  text-decoration-line: overline underline;
  text-decoration-color: purple;
}
```

Setting the Style of the Decoration Line

The text-decoration-style property defines how the line appears. It supports multiple styles:

- solid – a single straight line (default)
- double – a pair of parallel lines
- dotted – made up of small circular dots
- dashed – composed of dashes
- wavy – a wave-like decorative line

Example:

```
h1 {
  text-decoration-line: underline;
  text-decoration-style: solid;
}
h2 {
  text-decoration-line: underline;
  text-decoration-style: double;
}
h3 {
  text-decoration-line: underline;
  text-decoration-style: dotted;
}
p.ex1 {
  text-decoration-line: underline;
  text-decoration-style: dashed;
}
```

```
p.ex2 {
  text-decoration-line: underline;
  text-decoration-style: wavy;
}
p.ex3 {
  text-decoration-line: underline;
  text-decoration-color: red;
  text-decoration-style: wavy;
}
```

Adjusting the Thickness of the Decoration Line

The text-decoration-thickness property controls how thick or thin the decoration line appears.

Example:

```
h1 {
  text-decoration-line: underline;
  text-decoration-thickness: auto;
}
h2 {
  text-decoration-line: underline;
  text-decoration-thickness: 5px;
}
h3 {
  text-decoration-line: underline;
  text-decoration-thickness: 25%;
}
p {
  text-decoration-line: underline;
  text-decoration-color: red;
  text-decoration-style: double;
  text-decoration-thickness: 5px;
}
```

Shorthand Notation

You can combine all these properties into a single text-decoration declaration for simplicity.

Example:

```
h1 {
  text-decoration: underline;
}
h2 {
  text-decoration: underline red;
}
h3 {
  text-decoration: underline red double;
}
p {
  text-decoration: underline red double 5px;
}
```

Removing Underlines from Links

By default, all HTML links appear underlined. If you want to remove the underline, use the text-decoration: none; property.

Example:

```
a {
  text-decoration: none;
}
```

This approach is commonly used to create cleaner navigation menus or custom-styled hyperlinks while maintaining readability and design consistency.

3.3.1.4 Text Transformation

The text-transform property in CSS is used to modify the capitalization style of text within an HTML element. It allows designers to display text in uppercase, lowercase, or capitalized form without changing the actual content in the HTML source code. This makes it useful for maintaining consistent text formatting across a website.

The text-transform property supports the following values:

- none – Keeps the text in its original form without any transformation.
- capitalize – Converts the first letter of every word to uppercase.

- uppercase – Changes all letters in the text to uppercase.
- lowercase – Changes all letters in the text to lowercase.

This property is often used to enhance readability, highlight headings, or standardize the appearance of text in menus and buttons.

Example:

```
p.uppercase {
  text-transform: uppercase;
}
p.lowercase {
  text-transform: lowercase;
}
p.capitalize {
  text-transform: capitalize;
}
```

In this example:

- The paragraph with class “uppercase” will display all text in capital letters.
- The one with class “lowercase” will show all text in small letters.
- The paragraph with class “capitalize” will automatically capitalize the first letter of each word.

Using text-transform helps achieve a clean, professional look in web typography without manually rewriting text in different cases.

3.3.1.5 Text Spacing

CSS provides several properties that allow you to control the spacing within and between lines of text, characters, and words.

Key Text Spacing Properties

In this section, you will learn about the following properties:

- text-indent
- letter-spacing
- line-height

- word-spacing
- white-space

Text Indentation

The text-indent property defines how much the first line of a text block is indented.

Tip: You can use negative values to move the first line to the left instead of to the right.

Example:

```
p {
  text-indent: 50px;
}
```

This example indents the first line of every paragraph by 50 pixels.

Letter Spacing

The letter-spacing property adjusts the space between individual characters in a text.

Tip: Negative values can be used to make letters closer together.

Example:

```
h1 {
  letter-spacing: 5px;
}
h2 {
  letter-spacing: -2px;
}
```

The first heading increases the space between letters, while the second decreases it.

Line Height

The line-height property controls the vertical space between lines of text.

Note: Negative values are not allowed for this property.

Example:

```
p.small {
  line-height: 0.8;
}
p.big {
```

```
  line-height: 1.8;
}
```

In this example, the .small paragraph has tightly packed lines, while .big provides more space between lines for easier readability.

3.3.1.6 Text Shadow

The text-shadow property is used to apply shadow effects to text, giving it depth or a glowing appearance.

At its most basic level, you define the horizontal and vertical shadow offsets. You can also include an optional color and blur radius to create more visually appealing effects.

Basic Example

Apply a simple horizontal and vertical shadow to a heading:

```
h1 {
  text-shadow: 2px 2px;
}
```

Adding Color

You can specify a shadow color to enhance visibility or style:

```
h1 {
  text-shadow: 2px 2px red;
}
```

This example adds a red shadow to the text.

Adding Blur Effect

For a softer, more realistic shadow, include a blur radius value:

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```

Here, the 5px blur makes the shadow appear smooth and diffused.

Additional Text Shadow Examples

1. White Text with Black Shadow

```
h1 {
  color: white;
  text-shadow: 2px 2px 4px #000000;
}
```

This example creates contrast by giving white text a subtle black shadow, making it stand out on light or colored backgrounds.

2. Red Neon Glow Effect

```
h1 {
  text-shadow: 0 0 3px #ff0000;
}
```

This creates a glowing red neon effect by using zero offset and a small blur radius, giving the text a luminous appearance.

3.3.2 CSS Fonts

Fonts play a vital role in web design because they directly affect readability, user experience, and the overall visual appeal of a website. The right font not only enhances the design but also helps establish a unique identity for your brand. When selecting fonts, it's important to prioritize readability and pair them with suitable colors and sizes to ensure a pleasant viewing experience for users.

3.3.2.1 The CSS font-family Property

The font-family property in CSS is used to define the typeface of text within an element. It allows designers to specify one or more fonts in a prioritized list known as a *fallback system*. If the browser cannot display the first font, it automatically moves to the next available one in the list.

When listing fonts, each font name should be separated by a comma. If a font name includes multiple words, it must be enclosed in quotation marks—for example, "Times New Roman". It is also best practice to end the list with a *generic family name* (such as serif, sans-serif, or monospace), allowing the browser to choose a similar-looking font if none of the specified fonts are available.

Example:

```
.p1 {
  font-family: "Times New Roman", Times, serif;
```

```
}
.p2 {
  font-family: Arial, Helvetica, sans-serif;
}
.p3 {
  font-family: "Lucida Console", "Courier New", monospace;
}
```

In this example:

- The first paragraph uses the serif family, ideal for traditional or formal designs.
- The second paragraph uses sans-serif fonts, which appear clean and modern.
- The third paragraph uses monospace fonts, often used for coding or technical content.

Selecting the right font combination ensures visual harmony, accessibility, and a professional look for your website.

3.3.2.2 Generic Font Families

In CSS, fonts are categorized into five main generic font families, each with its own distinct style and purpose:

- Serif fonts : Feature small decorative strokes at the ends of each letter, giving text a formal and elegant appearance.
- Sans-serif fonts : Have smooth, clean edges without strokes, offering a modern and minimalistic look.
- Monospace fonts : Each character has a uniform, fixed width, producing a technical or typewriter-like effect.
- Cursive fonts : Designed to mimic natural handwriting, adding a personal and creative touch.
- Fantasy fonts : Include decorative or playful designs, often used for artistic or informal purposes.

Every font you use in CSS falls under one of these five generic font families.

Table 3.3.1 Generic font family with examples

Generic Font Family	Examples of Font Names
Serif	Times New Roman Georgia Garamond
Sans-serif	Arial Verdana Helvetica
Monospace	Courier New Lucida Console Monaco
Cursive	Brush Script MT Lucida Handwriting
Fantasy	Copperplate Papyrus

3.3.2.3 Web Safe Fonts

Understanding Web Safe Fonts

Web safe fonts are typefaces that are commonly available across most operating systems, browsers, and devices. They ensure that your website's text looks consistent, regardless of the platform being used. Since these fonts are pre-installed on most systems, they help maintain the intended design and readability of your webpages.

The Need for Fallback Fonts

Even though web safe fonts are widely supported, no font is guaranteed to work everywhere. Some devices or browsers might not have certain fonts installed. To address this, web developers use fallback fonts—a list of alternative fonts defined in the font-family property. If the first font in the list is unavailable, the browser automatically switches to the next option, ensuring your text remains legible. It is a best practice to always end this list with a generic font family such as serif, sans-serif, or monospace so that the browser can substitute a similar font style.

Example:

```
p {
  font-family: Tahoma, Verdana, sans-serif;
}
```

In this example, if *Tahoma* is not found, the browser will use *Verdana*, and if that too fails, it will revert to the generic *sans-serif* font.

Common Web Safe Fonts for HTML and CSS:

- Arial (sans-serif)
- Verdana (sans-serif)
- Tahoma (sans-serif)
- Trebuchet MS (sans-serif)
- Times New Roman (serif)
- Georgia (serif)
- Garamond (serif)
- Courier New (monospace)
- Brush Script MT (cursive)

Before publishing a website, always preview it across different browsers and devices. This ensures that the selected fonts render correctly and that the fallback fonts preserve readability and design consistency on all platforms.

3.3.2.3 Font Style and Font Weight

Font Style in CSS

The font-style property in CSS is used to define how the text should appear whether in a regular, italicized, or slanted form. It allows designers to add emphasis or stylistic variation to text without changing the font itself.

This property accepts the following values:

- normal – Displays text in its standard, upright form.
- italic – Renders text in an italicized style, often used for emphasis or quotations.
- oblique – Similar to italic but creates a slanted version of the text without using an actual italic font.

Example:

```
p.normal {
  font-style: normal;
}
p.italic {
  font-style: italic;
}
```

```
p.oblique {
  font-style: oblique;
}
```

In this example, different paragraph elements display the same text in varying styles normal, italic, and oblique to demonstrate their visual differences.

Font Weight in CSS

The font-weight property controls the thickness or boldness of text characters. It helps in improving readability, emphasizing headings, or distinguishing sections of content.

Possible values include:

- normal : The default thickness for most text.
- bold : Makes text appear thicker and more prominent.
- bolder and lighter : Adjust the weight relative to the parent element's font weight.
- Numeric values (100–900) : Provide fine-grained control, where 400 equals normal and 700 equals bold.

Example:

```
p.normal {
  font-weight: normal;
}
p.light {
  font-weight: lighter;
}
p.thick {
  font-weight: bold;
}
p.thicker {
  font-weight: 900;
}
```

Here, each paragraph demonstrates a different level of text thickness. Using font-weight effectively enhances hierarchy, draws attention to key content, and maintains visual balance across a web page.

3.3.2.4 Font Variant

The font-variant property in CSS is used to control whether the text should appear in small-caps style or in its normal form. Small-caps formatting transforms lowercase letters into smaller-sized uppercase letters, while existing uppercase letters remain unchanged but slightly larger in size. This style is often used for headings, titles, or to give text a formal and elegant appearance.

When applied, small-caps do not change the actual content of the text—only its presentation on the page.

Example:

```
p.normal {
  font-variant: normal;
}
p.small {
  font-variant: small-caps;
}
```

In this example, the paragraph with the class .normal displays text in its default form, while the paragraph with the class .small renders the same text using small capital letters. Using font-variant effectively can enhance readability and add a refined typographic touch to web designs.

3.3.2.5 Font Size

The font-size property in CSS defines how large or small the text appears on a webpage. It plays a key role in ensuring readability and maintaining visual hierarchy. Designers should always use proper HTML tags (like <h1>–<h6> for headings and <p> for paragraphs) instead of manipulating font sizes to simulate headings or body text.

Font size values can be absolute or relative:

- Absolute units such as px (pixels) give fixed and precise control. Keyword values like small, medium, and large represent predefined browser sizes.
- Relative units such as em, rem, and % adjust dynamically based on other font sizes, making designs flexible across devices.
- em sizes text relative to its parent's font size.
- rem refers to the root HTML element's font size, ensuring consistent scaling across the document.
- % scales the text in proportion to the parent element's size.

- vw (viewport width) adjusts the text size based on the browser's width, enabling responsive typography.

Example (using different units):

```
/* Using pixels */
h1 { font-size: 40px; }
p { font-size: 16px; }
/* Using em (relative to parent) */
body { font-size: 16px; }
h1 { font-size: 2.5em; }
/* Using rem (relative to root) */
html { font-size: 16px; }
h2 { font-size: 1.875rem; }
/* Using vw (responsive to viewport) */
h1 { font-size: 10vw; }
```

In short, for static precision use px, and for responsive design use em, rem, or vw. Choosing the right unit helps maintain accessibility, readability, and adaptability across all screen sizes.

3.3.3 Box Model and styling elements

The CSS Box Model is a fundamental concept in web design and layout. It describes how every HTML element is represented as a rectangular box that consists of four distinct layers — content, padding, border, and margin.

Components of the Box Model

Each element's box is made up of the following parts, starting from the innermost to the outermost layer:

- **Content** : This is where the text, images, or other elements are displayed.
- **Padding** : The transparent space surrounding the content; it creates a gap between the content and the border.
- **Border** : A visible line (or styled edge) that wraps around the padding and content.
- **Margin** : The outermost transparent area that separates the element from other surrounding elements.

The box model helps in defining spacing, borders, and the overall structure of elements in a webpage layout.

Example: Basic Box Model Demonstration

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

This example shows how padding, border, and margin combine around the content to form a complete box.

3.3.3.1 Width and Height in the Box Model

When defining the width and height of an element using CSS, these values apply only to the content area. The total space an element occupies on a page also includes padding and borders, while margins determine spacing between elements but are not included in the element's size.

Example: Calculating Total Element Size

```
div {
  width: 320px;
  height: 50px;
  padding: 10px;
  border: 5px solid gray;
  margin: 0;
}
```

Total Width Calculation:

320px (content width)

+ 20px (left + right padding)

+ 10px (left + right border)

= 350px total width

Total Height Calculation:

50px (content height)

+ 20px (top + bottom padding)

+ 10px (top + bottom border)

= 80px total height

Formulas

- **Total Element Width** = content width + left padding + right padding + left border + right border
- **Total Element Height** = content height + top padding + bottom padding + top border + bottom border

Note: Margins influence the space around the element but are not part of the element's actual dimensions. The element's size calculation ends at the border.

Recap

- CSS allows you to style and format text by changing its color, alignment, spacing, and transformation to make webpages more readable and attractive.
- The color property defines the color of text, and it can be set using color names, HEX values, or RGB values.
- The background-color property sets the background color behind text or elements to improve visibility and contrast.
- The text-align property controls how text is aligned horizontally (left, right, center, or justify) within an element.
- The text-align-last property defines how the last line of a text block is aligned.
- The vertical-align property adjusts how inline elements, such as images, align vertically with surrounding text.
- The direction and unicode-bidi properties control the text direction, especially for languages that read right-to-left like Arabic or Hebrew.
- The text-decoration property adds or removes decorative lines such as underline, overline, or line-through from text.
- The text-transform property changes the capitalization of text, such as uppercase, lowercase, or capitalize.
- The text-indent, letter-spacing, word-spacing, and line-height properties control text spacing, indentation, and line gaps for better readability.

- The text-shadow property adds shadow or glow effects to text, making it stand out visually.
- The font-family property specifies the typeface used for text and allows a list of fallback fonts in case the first is unavailable.
- CSS includes five generic font families — serif, sans-serif, monospace, cursive, and fantasy — each with a unique style and purpose.
- The font-style, font-weight, font-variant, and font-size properties control text appearance, thickness, small-caps effect, and size respectively.
- The CSS Box Model explains that every HTML element is treated as a rectangular box with four layers — content, padding, border, and margin — which determine spacing and layout on a webpage.

Objective Type Questions

1. Which CSS property sets the color of text?
2. Which property defines the background color of an element?
3. What property aligns text horizontally?
4. Which property adjusts spacing between letters?
5. What property is used to indent the first line of text?
6. Which property changes the direction of text?
7. What property adds decorative lines to text?
8. Which property converts text to uppercase or lowercase?
9. What property adds shadow effects to text?
10. Which property defines the typeface of text?
11. What CSS property controls the boldness of text?
12. Which property defines the size of the text?
13. What is the outermost layer in the CSS box model?
14. What is the innermost part of the CSS box model?
15. Which property controls the vertical alignment of elements?

Answers to Objective Type Questions

1. color
2. background-color
3. text-align
4. letter-spacing
5. text-indent
6. direction
7. text-decoration
8. text-transform
9. text-shadow
10. font-family
11. font-weight
12. font-size
13. margin
14. content
15. vertical-align

Assignments

1. Explain the use of the color and background-color properties in CSS with suitable examples.
2. Describe the different values of the text-align property and explain how each affects text placement on a webpage.
3. What is the purpose of the text-decoration property in CSS? Discuss its sub-properties with examples.
4. Explain the different text-transform options available in CSS. How do they help in improving text presentation?

5. Write short notes on letter-spacing, word-spacing, and line-height properties. Provide examples to illustrate their use.

6. Define the CSS Box Model. Explain its components and how they determine the total space occupied by an element.

7. Differentiate between absolute and relative font size units in CSS with examples.

8. What is the purpose of web safe fonts and fallback fonts in CSS? Explain with examples.

Reference

1. Negrino, T., & Smith, D. (2011). *JavaScript and Ajax for the web: Visual QuickStart guide* (8th ed.). Peachpit Press.

2. Felke-Morris, T. (2020). *Web development and design foundations with HTML5* (9th ed.). Pearson.

3. McFarland, D. S. (2015). *CSS: The missing manual* (4th ed.). O'Reilly Media.

Suggested Reading

1. Duckett, J. (2011). *HTML and CSS: Design and build websites*. Wiley.

2. Nixon, R. (2021). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (6th ed.). O'Reilly Media.

3. Castro, E., & Hyslop, B. (2013). *HTML5 and CSS: Visual QuickStart guide* (8th ed.). Peachpit Press.

4. Freeman, E. (2012). *Head First HTML and CSS* (2nd ed.). O'Reilly Media.

Layout and Positioning

Learning Outcomes

After completing this unit, learners will be able to:

- describe how the CSS overflow property controls extra content inside a box
- explain how the CSS float property moves items to the left or right on a webpage
- show how the CSS border-image property adds an image around a box
- use the CSS border-radius property to make the corners of a box round

Prerequisites

In web development, simply styling individual elements such as text, images, or buttons is not enough. To create an organized, visually appealing, and user-friendly webpage, it is essential to control where elements are placed and how they relate to one another on the page. This is why layout and positioning in CSS are crucial.

For example, consider designing a newspaper website. The header containing the site logo should remain at the top, while the navigation menu appears just below it, properly aligned. Articles must be arranged in columns, advertisements or sidebars positioned on the right, and the footer fixed at the bottom regardless of page length.

Without CSS layout and positioning, all elements would simply appear one after another vertically, making it impossible to create columns, fixed headers, sidebars, or responsive designs for different screen sizes. With CSS layout and positioning, developers can precisely control the placement of elements. Techniques such as Flexbox, Grid, and the position properties—static, relative, absolute, fixed, and sticky—make this possible.

Through effective use of these techniques, a webpage becomes structured, readable, and visually balanced. CSS layout and positioning transform unstructured content into a well-organized, professional, and responsive design—much like how a newspaper arranges text, images, and advertisements for clarity and accessibility.

Keywords

CSS Overflow, Float, Border-image, Border-radius, Layout, Styling

Discussion

3.4.1 CSS Overflow Property

The overflow property in CSS determines how content is handled when it exceeds the boundaries of its containing element. It allows developers to control whether the overflowing content is visible, hidden, or scrollable, ensuring better layout management and a cleaner design.

This property accepts the following values:

- visible : Default behavior; content is not clipped and extends outside the container.
- hidden : Extra content is clipped, and anything beyond the defined area is not visible.
- scroll : Adds both horizontal and vertical scrollbars, allowing users to view the hidden content.
- auto : Similar to scroll, but displays scrollbars only when the content actually overflows.

Example:

```
div {
  width: 200px;
  height: 65px;
  background-color: coral;
  overflow: auto;
}
```

In this example, scrollbars will appear only if the text or elements inside the div exceed its size.

1. Overflow: visible

When set to visible, the content flows outside the box without being clipped. This is the default setting and is useful when overflow does not disrupt the layout.

2. Overflow: hidden

With hidden, any extra content that goes beyond the container is cut off. This helps maintain clean layouts when only visible content should be displayed.

```
div {
  overflow: hidden;
}
```

3. Overflow: scroll

Using scroll ensures that scrollbars are always visible, even if the content fits perfectly inside. This gives users control to explore hidden sections manually.

```
div {
  overflow: scroll;
}
```

4. Overflow: auto

The auto value dynamically adds scrollbars only when content overflows, making it a practical option for responsive layouts.

5. Overflow-x and Overflow-y

You can control overflow separately for horizontal and vertical directions:

- **overflow-x** controls the left and right overflow.
- **overflow-y** controls the top and bottom overflow.

Example:

```
div {
  overflow-x: hidden; /* Hide horizontal overflow */
  overflow-y: scroll; /* Allow vertical scrolling */
}
```

In summary, the overflow property is essential for managing how content fits within a container, especially when dealing with responsive designs or dynamic content. It enhances user experience by maintaining order and readability within the layout.

3.4.2 CSS Float

The CSS float property defines how an element should be positioned (or "floated") within its container. It allows elements — such as images, text blocks, or divs — to align to the left or right, enabling surrounding content to wrap around them.

When the browser window is resized, floated elements maintain their position while the adjacent text automatically adjusts and wraps around them.

Purpose of the float Property

The float property is primarily used for content positioning and layout formatting. A common use is placing an image on one side of a container while text flows beside it.

Float Property Values

The float property accepts the following values:

Table 3.4.1 Float properties

Value	Description
left	Floats the element to the left of its container.
right	Floats the element to the right of its container.
none	Default value. The element does not float and remains in its normal position.
inherit	Inherits the float value from the parent element.

The float property is frequently used to wrap text around images for better visual presentation.

Example: Float Right

The float: right; value positions an element on the right side of its container, allowing surrounding text to flow on the left.

```
img {
  float: right;
}
```

Result:

An image is aligned to the right, and the paragraph text wraps neatly around it.

Example: Float Left

The float: left; value moves an element to the left side of its container so that text wraps to the right.

```
img {
  float: left;
}
```

Result:

The image appears on the left, while the text flows around its right side.

Example: Float None

Using float: none; removes floating behavior, meaning the element appears in its normal document flow position.

```
img {
  float: none;
}
```

Result:

The image is displayed exactly where it appears in the HTML structure, with no text wrapping around it.

Floating Elements Side by Side

By default, elements are block-level, meaning they occupy the full width of their container and start on a new line. However, by applying float: left;, multiple elements can be displayed next to each other horizontally.

```
div {
  float: left;
  padding: 15px;
}
.div1 {
  background: red;
}
.div2 {
  background: yellow;
}
.div3 {
  background: green;
}
```

Result:

The three blocks appear side by side, each aligned to the left, creating a clean horizontal layout.

The float property is a powerful CSS tool for aligning elements and creating flexible text-wrapping effects, especially when designing image layouts, sidebars, or multi-column sections.

3.4.3 CSS Border Images

The border-image property in CSS allows designers to replace a traditional solid or dashed border with an image, giving web elements a more customized and visually appealing appearance. Instead of using plain border colors, this feature uses an image that is sliced and applied around the edges of an element.

When applied, the border image is divided into nine sections (like a tic-tac-toe grid). The four corners of the image are placed at the corners of the element, while the edges and middle sections are either stretched or repeated to fill the border area, depending on the settings used.

The border-image property is a shorthand that combines several sub-properties:

- border-image-source : defines the path of the image file.
- border-image-slice : specifies how the image should be divided.
- border-image-width : sets the width of the image border.
- border-image-outset : determines how far the border image extends beyond the border box.
- border-image-repeat : defines whether the image should be stretched, repeated, or rounded.

Important: The element must have a standard border defined (e.g., border: 10px solid transparent;) for the border image to display correctly.

Example 1 – Using a Repeated Border Image:

```
#borderimg {
  border: 10px solid transparent; /* Required for border-image */
  padding: 15px;
  border-image: url(border.png) 30 round;
}
```

In this example, the image is sliced 30 pixels from each side and repeated (rounded) along the borders.

Example 2 – Using a Stretched Border Image:

```
#borderimg {
  border: 10px solid transparent; /* Required for border-image */
  padding: 15px;
  border-image: url(border.png) 30 stretch;
}
```

Here, the edges of the image are stretched instead of repeated to fit the border's dimensions.

Example 3 – Different Slice Values:

```
#borderimg1 { border-image: url(border.png) 50 round; }
#borderimg2 { border-image: url(border.png) 20% round; }
#borderimg3 { border-image: url(border.png) 30% round; }
```

Changing the slice value alters how the image is divided and displayed along the border.

Using border-image helps designers add creativity and uniqueness to web elements, such as decorative boxes, banners, and frames, enhancing the overall aesthetic of a webpage.

3.4.4 CSS Rounded Corners

The CSS border-radius property allows developers to create smooth, rounded corners for HTML elements, enhancing the visual appeal of web pages. It can be applied to elements that have a background color, border, or background image. By adjusting the radius value, you can control how curved each corner appears — from subtle rounding to perfect circles.

Basic Usage

The simplest use of border-radius applies a uniform curve to all four corners of an element. For example:

```
#div1 {
  border-radius: 25px;
  background-color: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}
```

This creates a box with softly rounded corners and a green background. The same property works equally well for elements that have borders or background images.

Corner-Specific Rounding

The border-radius property can take one to four values, allowing precise control over individual corners:

- One value: Applies the same radius to all corners.
- Two values: The first affects the top-left and bottom-right corners; the second affects the top-right and bottom-left corners.
- Three values: The first applies to the top-left corner, the second to the top-right and bottom-left, and the third to the bottom-right.
- Four values: Each value applies to one corner in clockwise order — top-left, top-right, bottom-right, and bottom-left.

Example:

#div1 { border-radius: 15px 50px 30px 5px; background: #04AA6D; }

This code assigns different rounding levels to each corner for a custom look.

Elliptical and Circular Shapes

You can also create elliptical or circular corners by using two values separated by a slash (/). The first value sets the horizontal radius, and the second defines the vertical radius.

Example:

```
#div1 { border-radius: 70px / 30px; }   /* Elliptical corners */

#div2 { border-radius: 50%; }           /* Perfect circle */
```

Using border-radius: 50% on a square element transforms it into a circle, while applying it to a rectangle produces an oval shape.

The border-radius property is a powerful CSS tool for creating soft, rounded, or even fully circular shapes without relying on images or complex markup. It improves design flexibility, reduces loading time, and contributes to a cleaner, more modern interface.

Recap

- The CSS overflow property controls how content is displayed when it exceeds the boundaries of its container.

The overflow property accepts four key values:

- visible (default) – content overflows visibly outside the box.

- hidden – extra content is clipped and invisible.
- scroll – adds scrollbars regardless of content size.
- auto – adds scrollbars only when content actually overflows.
- The overflow-x and overflow-y properties allow independent horizontal and vertical control of overflow behavior.
- Using overflow helps maintain clean, responsive layouts and prevents text or elements from spilling outside containers.
- The CSS float property positions elements to the left or right of their container, allowing surrounding text or inline elements to wrap around them.
- Float values include:
    - left – element floats to the left of the container.
    - right – element floats to the right.
    - none – element remains in normal document flow.
    - inherit – element inherits the float property from its parent.
- Floating is commonly used to wrap text around images or align multiple boxes horizontally within a container.
- By using float: left;, multiple elements can appear side by side, creating horizontal layouts.
- The border-image property replaces traditional solid borders with custom images, enhancing the visual appeal of web elements.
- The border image is divided into nine sections (four corners, four edges, and one center), allowing flexible stretching or repeating around the element.
- Key sub-properties of border-image include:
    - border-image-source – specifies the image file path.
    - border-image-slice – defines how the image is divided.
    - border-image-width – sets border thickness.
    - border-image-outset – extends the image beyond the border box.
    - border-image-repeat – controls whether the image is stretched, repeated, or rounded.
- A standard border (e.g., border: 10px solid transparent;) is required for border-image to display properly.
- The border-radius property allows developers to create rounded or circular corners, improving design smoothness and aesthetics.

- border-radius can take one to four values to round specific corners, or use percentage values (like 50%) to form perfect circles or ovals.
- Together, the properties overflow, float, border-image, and border-radius enhance layout control, visual design, and user experience — making webpages cleaner, more dynamic, and visually engaging.

Objective Type Questions

1. Which CSS property controls what happens when content exceeds its container's boundaries?
2. What is the default value of the overflow property?
3. Which overflow value hides extra content that goes beyond the container?
4. Which overflow value adds scrollbars only when the content overflows?
5. Which properties are used to handle overflow separately in horizontal and vertical directions?
6. What is the main purpose of the float property in CSS?
7. What is the default value of the float property?
8. What happens when multiple div elements are set with float: left;?
9. Which CSS property allows an image to be used as a border instead of a solid color?
10. What must be defined for the border-image property to work correctly?
11. Which sub-property defines how the border image is sliced?
12. Which sub-property controls whether the border image is stretched, repeated, or rounded?
13. Which CSS property is used to create rounded corners on an element?
14. What value of border-radius is used to create a perfect circle?
15. How many values can the border-radius property accept to define individual corner rounding?

Answers to Objective Type Questions

1. overflow
2. visible
3. hidden
4. auto
5. overflow-x and overflow-y
6. To align elements to the left or right within a container, allowing text or other elements to wrap around them.
7. none
8. The div elements appear side by side horizontally.
9. border-image
10. A standard border (e.g., border: 10px solid transparent;)
11. border-image-slice
12. border-image-repeat
13. border-radius
14. 50%
15. Four values (one for each corner in clockwise order)

Assignments

1. Explain the CSS overflow property in detail. Describe its purpose, different possible values, and how each value affects the layout and visibility of overflowing content. Include suitable examples.
2. Discuss the use of overflow-x and overflow-y properties. How do they differ from the overflow shorthand property? Provide examples demonstrating horizontal and vertical overflow handling.
3. Define the float property in CSS and explain its importance in web page layout design. Illustrate with examples how float: left, float: right, and float: none work, and describe a practical use case for each.

4. Write a detailed note on the CSS border-image property. Explain how it replaces traditional borders, describe its sub-properties (border-image-source, border-image-slice, border-image-width, etc.), and provide sample code.

5. Differentiate between stretched and repeated border images in CSS. How do the values stretch, repeat, and round affect the appearance of the border image? Support your answer with examples.

6. Explain the concept and usage of the CSS border-radius property. How can different values be applied to individual corners, and how is it used to create circular or elliptical shapes? Include examples.

7. Discuss how CSS overflow and float properties can be combined to manage complex web layouts. Explain how proper use of these properties improves visual structure and responsiveness, with an example layout.

Reference

1. Negrino, T., & Smith, D. (2011). *JavaScript and Ajax for the web: Visual QuickStart guide* (8th ed.). Peachpit Press.

2. Felke-Morris, T. (2020). *Web development and design foundations with HTML5* (9th ed.). Pearson.

3. McFarland, D. S. (2015). *CSS: The missing manual* (4th ed.). O'Reilly Media.

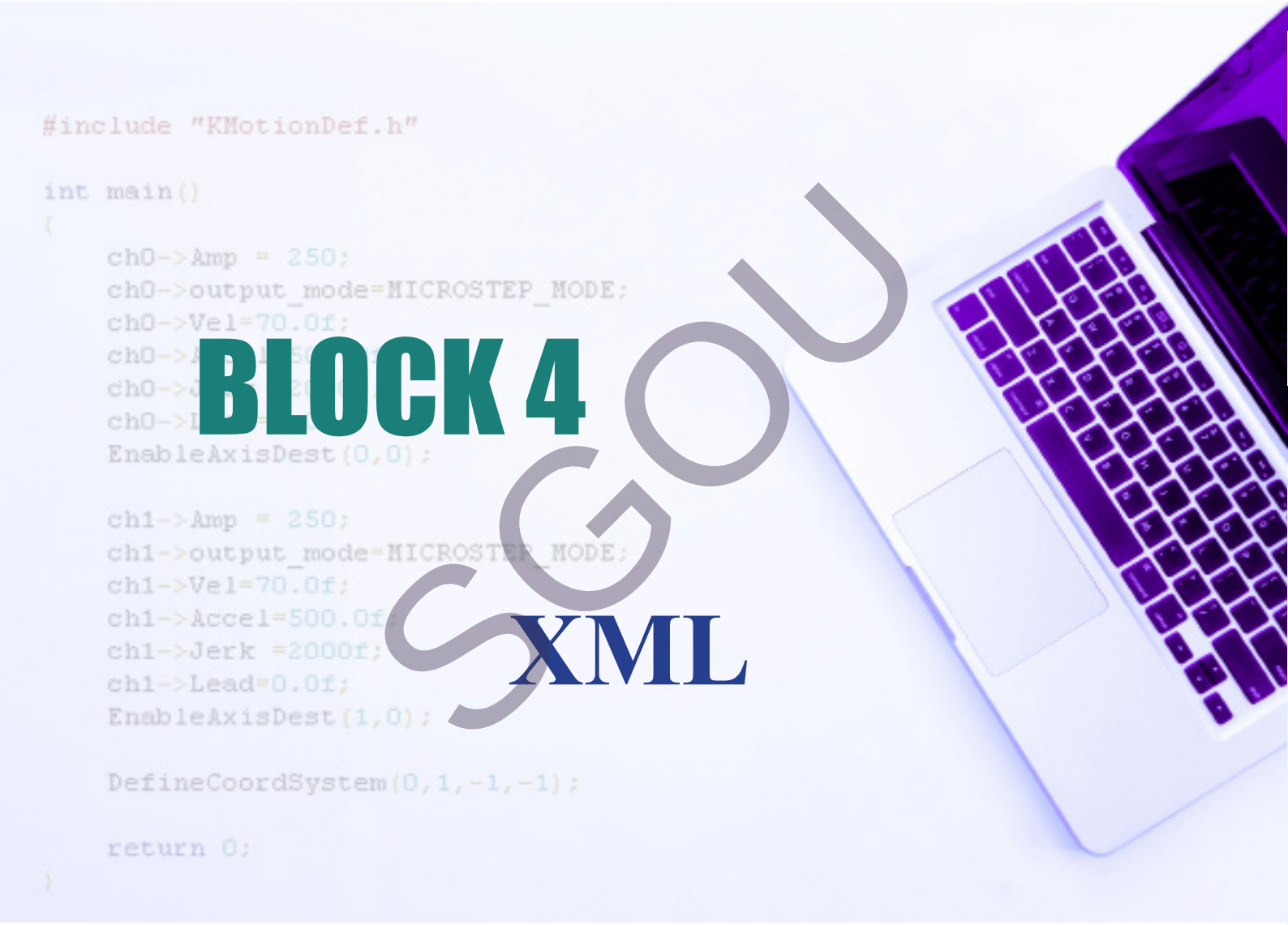
Suggested Reading

1. Duckett, J. (2011). *HTML and CSS: Design and build websites*. Wiley.

2. Nixon, R. (2021). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (6th ed.). O'Reilly Media.

3. Castro, E., & Hyslop, B. (2013). *HTML5 and CSS: Visual QuickStart guide* (8th ed.). Peachpit Press.

4. Freeman, E. (2012). *Head First HTML and CSS* (2nd ed.). O'Reilly Media.



BLOCK 4

XML

UNIT 1

XML Basics

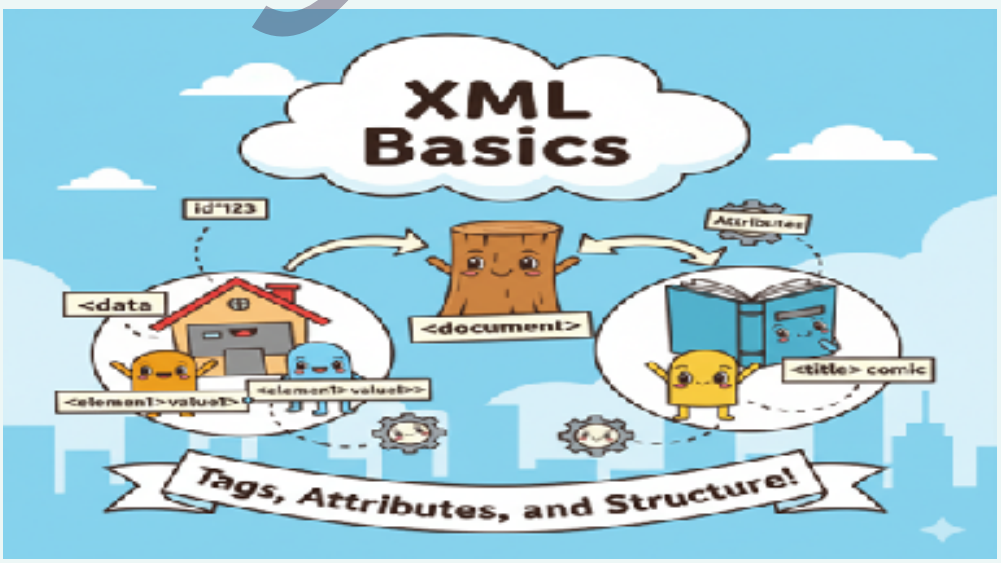
Learning OuWtcomes

After completing this unit, learners will be able to:

- explain the basic syntax rules to create well-formed XML documents
- differentiate between XML elements, tags, attributes and describe their significance in document structure
- use comments in XML to improve readability and documentation
- compare XML and HTML in terms of structure, purpose, and applications
- design simple XML documents to represent real-world data

Prerequisites

Before dive into the world of XML, you should have a solid grasp of some basic concepts. You don't need to be a coding expert, but familiarity with how files and folders are organized on a computer is a plus. If you've ever typed a sentence in a text editor or saved a document, you're already on the right track. This unit is designed for beginners, so we'll cover everything from scratch. The most important prerequisite is a willingness to learn how data can be structured in a simple yet powerful way.



XML is the backbone of data exchange on the web and in many applications. It's not about creating pretty websites like HTML, but rather about creating a universal format for data that both humans and machines can understand. By learning XML syntax rules, you'll gain a foundational skill that is essential for web development, software engineering, and data science. This unit will explore how elements, tags, and attributes work together to create a logical structure for any kind of data and also provide you with a powerful tool for tackling more complex tasks in data management.

This unit will transform how you think about data. You'll discover the simple logic behind XML structure, learn the difference between XML and HTML, and become fluent in using comments to make your code clear. Mastering these basics will not only give you a valuable skill but also provide a crucial stepping stone for understanding technologies like APIs and configuration files. Get ready to unlock the true potential of structured data and prepare yourself for a future in tech where data is the new gold!

Keywords

Syntax rules, structure, Elements, Tags, Attributes, Comments, HTML

Discussion

4.1.1 XML Fundamentals

Extensible Markup Language (XML) is a powerful and flexible language used for storing, transporting, and sharing data in a structured, human-readable, and machine-readable format. Unlike HTML, which focuses on displaying content on web pages, XML emphasizes describing the data itself. Users can create custom tags to represent different types of information, giving XML a level of adaptability that fits a wide variety of applications, from web services and databases to configuration files and content management systems. Its hierarchical structure organized with elements, attributes, and nested tags ensures that data is consistent and easy to interpret across platforms.

One of the main advantages of XML is its ability to separate data from presentation. This means that information can be shared and processed by different systems without worrying about how it will be displayed. For example, an XML file containing product details can be used by a website, a mobile application, or an internal database, all without changing the underlying data. This interoperability makes XML a cornerstone technology for businesses, developers, and organizations that need reliable, flexible data exchange between diverse systems.

Learning XML provides a strong foundation for working with structured data and modern technologies. It serves as the basis for many standards and tools, including SOAP (Simple Object Access Protocol) for web services for content syndication, and configuration management in software applications. By understanding XML, learners can create well-organized data formats, ensure seamless communication between

different applications, and develop skills that are highly relevant in data-driven industries. XML opens the door to efficient data management and integration in a wide range of digital environments.

Extensible Markup Language (XML) is a flexible, text-based language used to store, transport, and describe data in a structured format. It is both human-readable and machine-readable, allowing data to be shared and processed across different systems, platforms, and applications. Unlike HTML, which focuses on presentation, XML focuses on the meaning and organization of data, enabling interoperability between diverse software and systems. Some features of XML are:

1. **Self-descriptive:** XML tags clearly describe the data, making it easy to understand.

2. **Customizable tags:** Users can create their own tags to represent any type of data.

3. **Hierarchical structure:** Data is organized in a tree-like structure with nested elements.

4. **Platform-independent:** XML can be used across different operating systems and programming languages.

5. **Extensible:** It can be easily extended to include new data types or structures.

6. **Supports validation:** Data can be validated using DTD (Document Type Definition) or XML Schema to ensure correctness.

7. **Separation of data and presentation:** XML stores data separately from how it will be displayed, allowing flexible use.

4.1.2 XML Syntax Rules

XML (Extensible Markup Language) is a strict, text-based language used for storing, transporting, and structuring data. To ensure that an XML document is well-formed and can be correctly interpreted by machines, it must follow specific syntax rules. These rules help maintain data integrity and make XML documents predictable and interoperable.

1. **Root Element:** Every XML document must have exactly one root element, which contains all other elements in the document. This root element acts as the parent container for the entire XML structure. In this example <library> is the root element.

Syntax	Example
<root> <child> <subchild>.....</subchild> </child> </root>	<library> <book>XML Basics</book> </library>

2. **Closing Tags:** Every opening tag must have a corresponding closing tag. Self-closing tags are allowed for empty elements.

Example
<author>John Doe</author>

3. **Case Sensitivity:** XML is case-sensitive. <Title> and <title> are considered different elements, unlike HTML. Opening and closing tags must be written with the same case.

Example	
<Title>XML Basics</Title>	***<!-- Different from <title> -->***

4. **Entity References:** Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

Example	
<message> salary < 1000 </message>	***<!--This will generate an XML error.-->***

To avoid this error, replace the "<" character with an entity reference.

Example
<message>salary < 1000</message>

There are 5 pre-defined entity references in XML (Table 4.1.1).

Table 4.1.2 Entity references in XML

Entity References	Symbol	Description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	Apostrophe
"	"	quotation mark

5. **XML attribute values must always be quoted:** XML elements can have attributes in name/value pairs just like in HTML. In XML, the attribute values must always be quoted. All attribute values must be enclosed in quotes, either double (") or single (').

Example
<book id="101" genre='Technology'>XML Basics</book>

6. **XML Elements Must be Properly Nested:** Tags cannot overlap; the elements must be fully nested inside one another. In HTML, you might see improperly nested elements.

Example	
This text is bold and italic	***<!—Correct-->***
This text is bold and italic	***<!—Incorrect-->***

In XML, all elements must be properly nested within each other. In the example above, "*Properly nested*" simply means that since the <*i*> element is opened inside the <*b*> element, it must be closed inside the <*b*> element.

7. **The XML Prolog:** The XML prolog is optional. If it exists, it must come first in the document.This line is called the XML prolog.

Example
<?xml version="1.0" encoding="UTF-8"?>

XML documents can contain international characters, like Norwegian or French. To avoid errors, you should specify the encoding used, or save your XML files as UTF-8. UTF-8 is the default character encoding for XML documents.

8. **Whitespace:** XML ignores extra spaces and line breaks within elements unless they are part of the content.

9. **Well Formed XML:** XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

4.1.3 XML structure

XML (Extensible Markup Language) is a widely used markup language that represents data in a structured, text-based, and hierarchical format. Unlike HTML, which is focused on data presentation, XML is mainly designed for data storage and exchange. To achieve this, XML follows a well-defined structure consisting of declarations, elements, tags, attributes, comments, and rules that make a document well-formed and valid.

The XML declaration is an optional statement at the very beginning of an XML document that specifies important information about the document, such as the XML version and character encoding. Its primary purpose is to tell the XML parser how to read the document correctly.

Syntax	Example
<?xml version="1.0" encoding="UTF-8"?>	<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

where,

- **version:** Specifies the XML version being used, usually "1.0".

- **encoding:** Defines the character encoding, e.g., "UTF-8" or "ISO-8859-1".
- **standalone (optional):** Indicates whether the XML document relies on an external DTD (Document Type Definition). i.e.; standalone="yes"

4.1.3.1 Elements

In XML, elements are the fundamental building blocks of the document. An element usually consists of a *start tag*, some *content*, and an *end tag*. Elements can contain text, attributes, other nested elements, or even a mix of these, making them essential for representing structured information. Elements can also be empty (self-closing) if they do not hold any content. Since XML is case-sensitive, element names must be used consistently. Properly defined elements help in organizing data logically in a hierarchical manner.

Syntax	Example
<elementname>content</elementname>	<title>Introduction to XML</title> <author>John Smith</author>

In XML, nested elements refer to the practice of placing one element inside another to represent a *hierarchical structure* of data. The element that contains other elements is called the *parent element*, while the elements inside it are known as *child elements*. This nesting allows XML to organize information in a tree-like structure, making it easier to represent complex data.

Example
<book> <title>XML Basics</title> <author>John Smith</author> </book>

Output:

book

├── title : Introduction to XML

└── author : John Smith

Here, <*book*> is an element that acts as a parent and contains two child elements, <*title*> and <author>. The <*title*> element has the text content *"Introduction to XML"* while <*author*> has "John Smith." This shows how elements can group related information together in a structured format.

4.1.3.2 Tags

In XML, tags are used to mark the beginning and end of an element. They define the structure of the data and must always appear in pairs, consisting of a start tag and an end

tag, except for empty (self-closing) tags. A start tag marks the beginning of an element, while an end tag marks the close of an element. A start tag is written as <*tagname*> and an end tag as </*tagname*>. The content of the element is placed between these tags. XML is case-sensitive, which means <Book> and <book> are treated as different tags. Tags are used to define the boundaries of elements. Tags must be properly nested to ensure that the document is well-formed. XML also supports empty tags (self-closing) for elements without content.

Additionally, tags must always be properly nested without overlap to make the document well-formed.

Syntax	Example
<starttag>content</starttag>	<book>XML Guide</book>
<emptytag />	

In this example, <*book*> is the start tag and </*book*> is the end tag, with the content "XML Guide" inside them. The <*br* /> tag is an empty (self-closing) tag that does not contain any content. Proper use of tags ensures that XML parsers can correctly interpret and process the data.

Output:

book : XML Guide

linebreak : (empty)

4.1.3.3 Attributes

In XML, attributes are used to provide additional information about elements. They are always defined within the *start tag* of an element and are written as *name–value pairs*. Each attribute must have a unique name within the same element, and its value must always be enclosed in either single (' ') or double (" ") quotes. Attributes do not contain child elements but act as extra descriptive details about the element. They are especially useful for storing metadata, identifiers, or properties of data without being part of the main content.

Syntax	Example
<element attribute="value"> content</element>	<book id="b101" category="programming"> <title>XML Essentials</title> <author>John Smith</author> </book>

In this example, the element <book> has two attributes: *id="b101"* and *category="programming"*. These attributes describe properties of the book, such as its identification number and its subject category, while the child elements <*title*> and <*author*> store the actual content. Attributes help add details to elements without interrupting the element's main data structure.

Output:

book (id="b101", category="programming")

├── title : XML Essentials

└── author : John Smith

A valid XML document is not only well-formed but also conforms to a predefined set of rules described in a *DTD (Document Type Definition)* or *XML Schema.* This validation ensures that the document structure and data types follow the expected rules, which is essential in applications like web services and data exchange.

The structure of XML is built around a *tree-like hierarchy* where a single root element contains nested child elements, attributes, and text content. Comments, CDATA sections, and processing instructions enhance the flexibility of XML documents. By following well-formedness and validation rules, XML ensures that data remains consistent, portable, and easy to process across different platforms.

4.1.3.4 Comments

In XML, comments are used to add notes, descriptions, or explanations within the code that are meant for human readers but ignored by the XML parser. They are helpful for documenting the structure of the XML document, clarifying the purpose of elements, or temporarily disabling certain parts during development. A comment in XML always begins with <!-- and ends with -->. Comments can be placed anywhere in the document except inside tags or attribute values. Since they are ignored during parsing, they do not affect the data output or structure of the XML file.

Syntax	Example
<!-- This is a comment -->	<library> <!-- This section contains book details --> <book id="b101"> <title>XML Basics</title> <author>John Smith</author> </book> </library>

In this example, the comment <!-- This section contains book details --> explains the purpose of the following element block. Even though it is present in the XML file, it will not be displayed or processed as part of the actual data.

Output:

library

└── book (id="b101")

    ├── title : XML Basics

    └── author : John Smith

(Comment is ignored by the parser)

4.1.4 XML vs HTML

XML (Extensible Markup Language) and HTML (HyperText Markup Language) are both markup languages used in web and data technologies, but they serve different purposes. Understanding the differences between these two languages is essential for web development and data management. The key differences between XML and HTML lie in their purpose, tag rules, structure, and handling of content and presentation. XML is designed for data storage and transport, while HTML is meant for displaying content on web pages. In terms of tags, XML uses user-defined, case-sensitive tags and requires documents to be well-formed, whereas HTML relies on predefined, case-insensitive tags and follows a more lenient syntax. Regarding structure, XML enforces a strict, hierarchical arrangement of elements, while HTML is more flexible and primarily focuses on the visual layout of content. Finally, XML separates content from presentation, making it ideal for data processing, whereas HTML combines content with presentation, ensuring it is displayed correctly in browsers.

Table 4.1.2 Comparison of XML and HTML

Feature	**XML** (Extensible Markup Language)	**HTML** (HyperText Markup Language)
Purpose	Storing, transporting, and structuring data	Displaying and formatting content on web pages
Tag Definition	User-defined tags	Predefined tags (like <p>, <h1>,)
Case Sensitivity	Case-sensitive e.g.: <Book> ≠ <book>	Case-insensitive e.g. <P> = <p>
Syntax Rules	Strict; must be well-formed (all tags closed, properly nested)	Lenient; browsers can render even with errors
Hierarchy	Supports nested, hierarchical data	Less strict; mainly linear content structure
Attributes	Can add metadata to elements	Used for styling or element properties
Content vs Presentation	Focuses on data (content separated from presentation)	Combines content with visual presentation

Output	Structured data that applications can process	Formatted text displayed in a browser

In conclusion, understanding the basics of XML including its syntax rules, structure, elements, tags, attributes, and comments is essential for creating well-formed and structured documents. XML provides a flexible and extensible way to store and transport data, while its strict syntax ensures consistency and reliability. Unlike HTML, which focuses on displaying data in web browsers, XML emphasizes data organization and interoperability across different systems. Mastery of these fundamentals equips learners to design XML documents that are both readable by humans and processable by machines, laying a strong foundation for advanced XML applications and data exchange.

Recap

- XML (Extensible Markup Language) is a markup language used to store, transport, and structure data.
- XML is platform-independent and readable by both humans and machines.
- Every XML document can optionally start with an XML declaration.

  e.g., <?xml version="1.0" encoding="UTF-8"?>.
- Root Element: Each XML document must have exactly one root element that contains all other elements.
- Elements are the basic building blocks of XML and include a start tag, content, and end tag. Elements can contain text, attributes, or other nested elements.
- Nested Elements show parent-child relationships and create a hierarchical structure.
- Tags mark the start and end of elements and are case-sensitive. XML tags must be properly nested, and overlapping tags are not allowed.
- Empty elements (self-closing) are used when no content is present.

  e.g., <element />.
- Attributes provide additional information about elements and are included in the start tag.
- Attribute values must be enclosed in quotes (single ' ' or double " ").
- Multiple attributes can be added to an element, but each must be unique within the element.
- Elements can contain plain text content, which represents the actual data.

- Special characters like <, >, and & in content must be escaped as entities (<, >, &).
- Comments are used for documentation and are ignored by the XML parser.

  e.g., <!-- This is a comment -->.
- A well-formed XML document must have a single root, proper nesting, all tags closed, and quoted attributes.
- A valid XML document is well-formed and also conforms to a DTD (Document Type Definition) or XML Schema.
- XML is designed for data storage and transport, whereas HTML is for displaying content on web pages.
- XML enforces a hierarchical structure, while HTML is flexible and mainly linear for layout purposes.
- Understanding XML basics including elements, tags, attributes, comments, and syntax rules is essential for data exchange, web services, and structured information processing.
- Knowledge of XML vs HTML helps developers decide whether to use XML for data storage or HTML for content presentation.

Objective Type Questions

1. What does XML stand for?
2. What is the top-level element in an XML document called?
3. XML tags are ……..
4. What kind of element ends with />?
5. Which declaration is placed at the top of an XML file?
6. …… are used to provide additional information about elements.
7. What character represents “&” in XML text?
8. Name one predefined tag in HTML.
9. What is the structure type of XML?
10. What is used to describe rules for XML validation?
11. XML ignores white spaces within elements. (true/ false)

12. What is the smallest building block of XML?

13. What is the symbol of a comment in XML?

14. Which language is used for displaying web pages?

15. What symbol is used for *less than* in XML text?

16. What type of XML structure can contain child elements?

17. ……. are used to mark the beginning and end of an element.

18. XML can contain multiple root elements. (true/ false)

19. What must every XML element have to be well-formed?

20. What is the start of a processing instruction in XML?

Answers to Objective Type Questions

1. Extensible Markup Language
2. Root
3. case-sensitive
4. Empty
5. XML declaration
6. Attributes
7. &
8. <p>
9. Hierarchical
10. DTD/Schema
11. false
12. Element
13. <!-- -->
14. HTML

15. <

16. Parent element

17. Tags

18. False

19. Closing tag

20. <?

Assignments

1. Write a well-formed XML document for a single book with the following details: Title, Author, and Year of Publication. Include a root element.

2. Design an XML document for a library system that includes multiple books. Each book should have a title, author, publication year, and a unique ID attribute. Make sure your XML is well-formed and demonstrates nested elements and attributes.

3. Explain the difference between XML and HTML with examples. Include at least one example XML element and one HTML element to show how XML focuses on data structure.

4. List and explain at least 5 XML syntax rules with examples. For each rule, give one correct and one incorrect example to illustrate what makes XML well-formed or invalid.

Reference

1. Holzner, S. (2023). *XML: A Beginner's Guide: Go Beyond the Basics with Ajax, XHTML, XPath 2.0, XSLT 2.0, and XQuery*. McGraw-Hill Education.

2. Bocso, L. (2025). *Mastering XML: Practical Applications and Modern Techniques*. Wiley.

3. Gange, A. (2025). *XML Essentials: A Developer's Guide*. O'Reilly Media.

4. Smith, M. (2025). *XML for Beginners: An Introduction to Markup Languages*. Addison-Wesley.

Suggested Reading

1. W3Schools – XML Tutorial https://www.w3schools.com/xml/
2. MDN Web Docs – Introduction to XML https://developer.mozilla.org/en-US/docs/Web/XML
3. TutorialsPoint – XML Tutorial https://www.tutorialspoint.com/xml/index.html
4. GeeksforGeeks – XML Basics https://www.geeksforgeeks.org/xml/

UNIT 2

XML Components

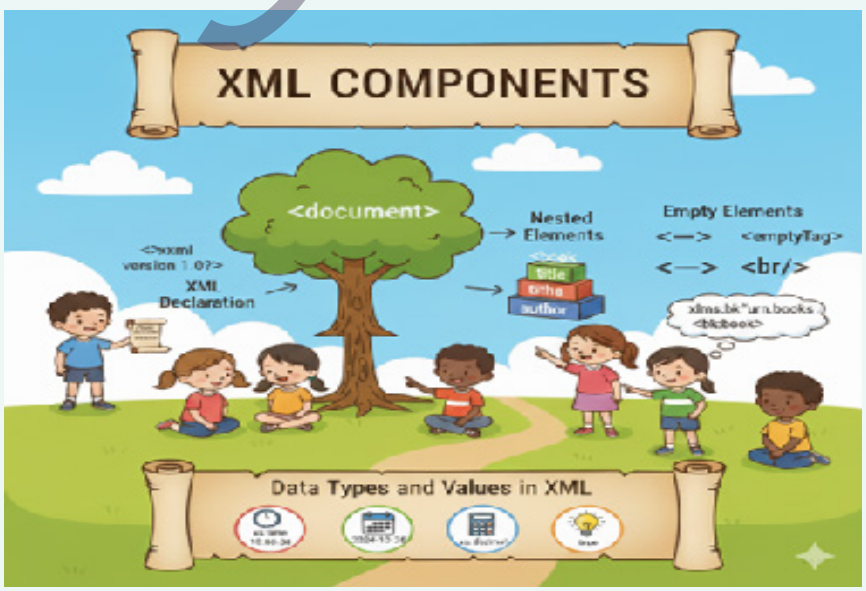
Learning Outcomes

After completing this unit, learners will be able to:

- understand the XML declaration and its purpose
- identify root, nested, and empty elements in an XML document
- use XML namespaces to avoid naming conflicts
- know data types and assign values to XML elements and attributes
- define correct XML documents following proper syntax and rules

Prerequisites

Before learning XML components, it's essential to understand the idea of structured data. XML is like a digital blueprint that organizes information in a clear hierarchy. Imagine a library: each book has a title, author, and publication year. Without a proper structure, it would be impossible to locate a book quickly. Similarly, XML provides a way to organize data so computers and humans can easily access and interpret it. This basic understanding of structured data prepares learners to handle more complex XML components.



Next, you should be familiar with the concept of elements and nesting. Elements are like boxes that store data, and sometimes these boxes are placed inside other boxes (nested elements). For example, consider an online shopping website: a customer order (root element) contains multiple items (nested elements), and each item may have details like price, quantity, and description. Some elements may be empty, like a "discount" field if no discount applies. Recognizing these patterns in real life helps learners grasp XML structures easily.

Finally, it's helpful to know about namespaces and data types. Namespaces prevent confusion when two different elements have the same name, similar to having street names in different cities. Data types ensure that values are correctly interpreted numbers, text, or dates. For instance, in a hotel booking system, "check-in date" must be treated as a date, while "room number" is a number. Understanding these prerequisites ensures learners can create accurate, error-free XML documents and manage real-world data efficiently.

Keywords

XML Declaration, Root, Nested, Empty Elements, Namespace, Data Types

Discussion

4.2.1 XML Components

XML (eXtensible Markup Language) is a flexible and widely used markup language designed to store, organize, and transport data in a structured, readable format. Every XML document can include an XML declaration, which specifies the version and encoding, followed by a single root element that contains all other elements. Inside the root, data is organized using nested elements**,** which create a hierarchical structure, and empty elements, which act as placeholders when no content is present. Elements can also include attributes to provide additional information, such as IDs, types, or categories. For example, a <*book*> element may contain <*title*> and <*author*> as nested elements, while an id attribute uniquely identifies the book. Comments can also be added to document or annotate the XML code without affecting the data.

XML also supports namespaces**,** which prevent naming conflicts when combining elements from different sources, ensuring unique identification of each element. Although XML itself does not enforce data types, standards like XML Schema allow defining types for elements and attributes, such as strings, numbers, and dates. This makes XML highly suitable for data exchange between different systems, including web services, databases, and configuration files. Overall, XML's components declarations, root, nested and empty elements, attributes, namespaces, comments, and typed values work together to create a flexible, structured, and interoperable format for storing and sharing information.

4.2.2 XML Declaration

The XML declaration helps parsers understand how to read and process the document. Its typical format is an XML prolog that includes the body, XML Declaration, and XML DTD (Document Type Definition). The XML prolog should always appear at the start of the document if it is present. The shortest XML declaration is created by including only the XML 1.0 Version, which is the default version. There are seven different character-encoding schemes, with UTF-8 being the default. There may be some encoding issues if it is absent. In XML, PCDATA stands for Parsed Character Data. It refers to the text content within an XML element that is intended to be parsed by an XML parser.

Syntax of XML Declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Syntax of DTD:

```
<!DOCTYPE root-element [<!element-declarations>]>
```

Example:

```
<!DOCTYPE website [
  <!ELEMENT website (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<website>
  <name>SHOPPING</name>
  <company>ABC</company>
  <phone>011-24533221</phone>
</website>
```

Every XML document must have exactly one root element, which encloses all other elements in the file. The root element acts as the parent of the entire document structure and provides a logical container for the data. Without a root element, the document will not be well-formed. Here, *<library>* is the root element containing the *<book>* element.

Syntax	Example
<root> <!-- child elements --> </root>	<library> <book>XML Basics</book> </library>

4.2.3 Root Element

The root element is the single, top-level element that contains all other elements in an XML document. Every XML document must have exactly one root element**,** making it the main container for the entire structure. Without a root element, the XML document is considered invalid.

Syntax	Example
<rootElement> <!-- Child elements go here --> </rootElement>	<library> <book> <title>XML Basics</title> <author>John Doe</author> </book> </library>

Here, *<library>* is the root element. *<book>* is a child element nested inside the root. Think of the root element like the main folder on a computer that holds all its subfolders and files inside it. The root element is essential in an XML document because there can only be one root element per file. It encloses all other elements, including nested elements, empty elements, and attributes, ensuring that the document maintains a hierarchical and well-formed structure**.**

4.2.4 Nested Elements

Nested elements are elements that are placed inside another element in an XML document. They help organize data in a hierarchical structure, making it easier to represent complex relationships between different pieces of information. The element that contains another element is called the *parent element*, while the element inside is called the *child element*.

Syntax	Example
<parentElement> <childElement> <!-- Further nested elements or content --> </childElement> </parentElement>	<library> <book> <title>XML Basics</title> <author>John Doe</author> </book> </library>

In this example, <book> is a child of the root element <library>, and <title> and <author> are children of <book>.

Nested elements allow structured data representation, making it easier to organize complex information. They can have multiple levels of nesting depending on the complexity of the data, and proper nesting is essential for the XML document to be well-formed. Think of nested elements like boxes within boxes, where each box contains related items in an organized way.

4.2.5 Empty elements

Empty elements are XML elements that do not contain any content or child elements. They are often used as placeholders or to represent optional data that may be added later. Empty elements are written using a self-closing tag.

W	Example
<elementName />	<book> <title>XML Basics</title> <summary /> <!-- Empty element for missing summary --> </book>

<*summary* /> is an empty element because it does not contain any text or nested elements. Empty elements are self-closing, using the / at the end, and are useful for optional data or when no value is available. They help ensure that the XML document remains well-formed without leaving any tags unclosed. Think of empty elements like empty boxes on a shelf, ready to be filled when needed.

4.2.6 XML namespace

An XML namespace is a method used to avoid naming conflicts when combining elements from different XML documents. It provides a way to distinguish elements and attributes that may have the same name but come from different sources. Namespaces are defined using the xmlns attribute. There are two types: Default Namespace and Prefixed Namespace.

4.2.6.1 Default Namespace Declaration

Default Namespace Declaration (Figure 4.2.1) in XML namespaces assigns a default namespace to all unprefixed components inside a given scope. This signifies that elements without a prefix are presumed from the given namespace. The "xmlns" element is used to declare the default namespace.

Syntax	Example
<root xmlns="http://example.com/ns"> <child>Content</child> </root>	<?xml version="1.0" encoding="UTF-8"?> <library xmlns="http://example.com/library"> <book> <title>XML Basics</title> <author>John Doe</author> </book> <book> <title>Advanced XML</title> <author>Jane Smith</author> </book> </library>

Example is to demonstrate the XML library catalog file structure in XML.

Fig. 4.2.1 Default Namespace

4.2.6.2 Prefixed Namespace Declaration

In XML namespaces, the Prefixed Namespace Declaration (Figure 4.2.2) method assigns a prefix to a namespace URI. This allows elements and attributes from that namespace to be identified with the specified prefix. Prefixed namespaces are especially beneficial when items from different namespaces appear in the same XML document.

Syntax	Example
<root xmlns:prefix="http://example.com/ns"> <prefix:child>Content</prefix:child> </root>	<?xml version="1.0" encoding="UTF-8"?> <catalog xmlns:bk="http://example.com/books" xmlns:auth="http://example.com/authors"> <bk:book> <bk:title>XML Basics</bk:title> <auth:author>John Doe</auth:author> </bk:book> <bk:book> <bk:title>Advanced XML</bk:title> <auth:author>Jane Smith</auth:author> </bk:book> </catalog>

Example is to demonsrtate the books catalog file structure in XML.

Fig. 4.2.2 Prefixed Namespace

4.2.7 Data types and values in XML

XML (eXtensible Markup Language) is primarily designed to store and transport data in a structured format. While XML itself does not enforce strict data types, it allows elements and attributes to contain various kinds of data values such as text, numbers, dates, and custom types. To enforce data types and validate content, XML Schema (XSD) or Document Type Definition (DTD) is used. Different types are listed below:

1. **Text Data:** The simplest form of data in XML. Any element or attribute can contain string/text. Here, <*title*> and <*author*> hold text values.

Example
<book> <title>XML Basics</title> <author>John Doe</author> </book>

2. **Numeric Data**: XML elements can store integers, decimals, or floating-point numbers. Useful for prices, quantities, or IDs. Here <price> is a decimal, <pages> is an integer.

Example
<book> <price>25.50</price> <pages>300</pages> </book>

3. **Date and Time Data:** XML can store dates and times in standard formats, often validated using XML Schema. This ensures consistent representation of dates across systems.

Example
<book> <publishedDate>2025-10-03</publishedDate> </book>

4. **Boolean Data:** Represents true/false values.

Example
<book> <available>true</available> </book>

5. **Custom/Complex Data Types:** Using XML Schema (XSD), elements can be restricted to specific patterns or enumerations.

Example
<book> <genre>Fiction</genre> <!-- Can be restricted to a set of values like Fiction, Non-Fiction, Science --> </book>

6. **Attributes and Data Values:** Attributes also store values of different types.

Example
<book id="B001" price="25.50" available="true"> <title>XML Basics</title> </book>

To sum up, the key components of XML including the XML declaration, root element, nested and empty elements, namespaces, and data types form the foundation for creating structured, readable, and reliable XML documents. Each component contributes to the document's clarity and functionality, ensuring that data is organized hierarchically, conflicts are avoided, and values are accurately represented. A solid understanding of these elements equips learners and developers to design XML documents that are both well-formed and easily shareable across different applications and platforms.

Recap

- XML (eXtensible Markup Language) is a flexible text-based format that is mainly used to store, transport, and share structured data across different systems.
- An XML document must always be well-formed, meaning it should follow the proper syntax rules like correct nesting, matching start and end tags, and a single root element.
- The XML declaration generally appears at the beginning of an XML document and provides information about the version, encoding, and whether the document is standalone or linked to an external source.
- XML supports different versions, but the most widely used and supported version is XML 1.0, which ensures compatibility with most applications.
- The encoding attribute in the XML declaration (commonly UTF-8 or UTF-16) ensures that characters, including special symbols, are represented correctly across systems.
- Although the XML declaration is technically optional, including it improves clarity and prevents confusion when the document is processed on different platforms.
- Every XML document must have a *root element*, which acts as the top-level container for all other elements within the document.
- The root element defines the *scope of the document's structure*, ensuring that all data and nested elements fall under a single unified hierarchy.
- Without a root element, an XML document becomes invalid, since there is no single structure to represent the data properly.
- Nested elements are elements placed inside other elements, and they help XML represent complex relationships and hierarchical data.
- The use of nested elements allows XML to model real-world structures, such as books containing chapters, or orders containing items, in a natural way.
- Proper nesting of elements is essential; if tags overlap incorrectly or close in the wrong order, the XML will not be well-formed.
- Empty elements are elements that do not contain text, attributes, or nested elements, and they are often used when data is optional or not currently available.
- These empty elements are written using a self-closing tag syntax like *
*, ensuring the document remains valid without requiring an explicit closing tag.
- Empty elements are extremely useful in cases where placeholders are needed, such as marking a break, an image tag, or missing values in structured data.

- XML namespaces are used to prevent naming conflicts when multiple XML vocabularies are combined into a single document.
- Namespaces are declared using the *xmlns* attribute, which assigns a unique URI to distinguish between elements that may otherwise have the same name.
- A default namespace can be defined to apply automatically to elements without prefixes, simplifying the syntax of XML documents.
- A prefixed namespace uses a short identifier, such as *<xsd:element>,* to clearly specify the vocabulary an element belongs to.
- Namespaces are especially valuable when integrating XML documents from different applications, industries, or standards, ensuring elements do not clash.
- XML stores content as text, but it supports the representation of different **data types**, such as strings, numbers, dates, and boolean, depending on schema definitions.
- By default, XML does not enforce data types; instead, all values are treated as text unless validated through schemas like DTD or XML Schema Definition (XSD).
- XML Schema (XSD) provides powerful mechanisms to define the structure, allowed elements, attributes, and data types of an XML document.
- Using schemas and defined data types helps maintain data integrity, consistency, and clarity, especially when XML is used in business transactions or data exchange.

Objective Type Questions

1. What does XML stand for?
2. XML is primarily used for _____.
3. Which of the following is optional in an XML document?
   a. Root element
   b. XML declaration
   c. Nested element
   d. Empty element
4. How many root elements can an XML document have?
5. The XML declaration is usually placed at the _____ of the document.

6. Which attribute specifies character encoding in XML declaration?

7. The default encoding for XML is _____.

8. What does the standalone attribute in XML declaration define?

9. Elements inside another element are called _____.

10. Proper nesting of elements ensures that XML is _____.

11. An element with no content or child elements is called a _____ element.

12. Which symbol is used to close an empty element?

13. What is the purpose of XML namespaces?

14. Which attribute is used to declare an XML namespace?

15. What is a default namespace?

16. What is a prefixed namespace?

17. In XML, all values are treated as _____ by default.

18. Which XML technology is commonly used to define data types?

19. XML Schema is written in _____.

20. A boolean data type in XML can have which two values?

21. What is the purpose of defining data types in XML?

22. Which of the following is NOT an XML data type?
    a. String
    b. Integer
    c. Decimal
    d. Picture

23. The `<br />` element in XML is an example of a _____.

24. XML is _____ case-sensitive.

25. An XML document that follows all syntax rules is called _____.

Answers to Objective Type Questions

1. eXtensible Markup Language
2. Storing and transporting data
3. b) XML declaration
4. Only one
5. Beginning (top)
6. encoding
7. UTF-8
8. Whether the XML depends on an external DTD/schema
9. Nested elements
10. Well-formed
11. Empty element
12. / before > (e.g., <tag />)
13. To avoid element name conflicts
14. xmlns
15. A namespace applied automatically to elements without a prefix
16. A namespace with a prefix attached to elements (e.g., <ns:tag>)
17. Text (string)
18. XML Schema Definition (XSD)
19. XML
20. true or false
21. To ensure consistency and validity of values
22. d) Picture
23. Empty element
24. Always
25. Well-formed document

Assignments

1. Explain the importance of the XML declaration in an XML document. Discuss its attributes (version, encoding, and standalone) with examples.

2. Describe the role of the root element in XML. Why is it mandatory to have exactly one root element? Support your answer with a real-life example in XML format.

3. Differentiate between nested elements and empty elements in XML. Provide suitable syntax examples to highlight their use and importance in structuring data.

4. What are XML namespaces and why are they necessary? Explain the difference between default and prefixed namespaces with examples showing their syntax and output.

Reference

1. Holzner, S. (2023). *XML: A Beginner's Guide: Go Beyond the Basics with Ajax, XHTML, XPath 2.0, XSLT 2.0, and XQuery*. McGraw-Hill Education.

2. Bocso, L. (2025). *Mastering XML: Practical Applications and Modern Techniques*. Wiley.

3. Gange, A. (2025). *XML Essentials: A Developer's Guide*. O'Reilly Media.

4. Smith, M. (2025). *XML for Beginners: An Introduction to Markup Languages*. Addison-Wesley.

Suggested Reading

1. W3Schools – XML Tutorial https://www.w3schools.com/xml/

2. MDN Web Docs – Introduction to XML https://developer.mozilla.org/en-US/docs/Web/XML

3. TutorialsPoint – XML Tutorial https://www.tutorialspoint.com/xml/index.html

4. GeeksforGeeks – XML Basics https://www.geeksforgeeks.org/xml/

Document Structure

Learning Outcomes

After completing this unit, learners will be able to:

- explain the structure of an XML document including declaration, root, child elements, attributes, and text content
- differentiate between well-formed and valid XML documents
- understand the role of DTD and XSD in defining and validating XML documents
- apply XML validation in real-world applications like e-commerce, databases, and web services

Prerequisites

In today's fast-paced professional world, meetings play a vital role in decision-making and collaboration. However, they often generate large volumes of unstructured discussions that are difficult to revisit or analyze later. Without proper documentation, important decisions, deadlines, and responsibilities can easily be lost or misunderstood. To address this challenge, automated meeting summarization workflows have become increasingly important. These systems help organizations transform lengthy discussions into concise summaries and actionable insights, thereby saving time and enhancing productivity. A basic understanding of audio processing, transcription tools, and natural language processing (NLP) is beneficial before studying this workflow, as these technologies constitute its core foundation.

Keywords

XML, Well-Formed, Valid, Root Element, Child Element, Attributes, PCDATA, CDATA, DTD, XSD, Schema, Namespace, Validation, Parser, Interoperability

Discussion

4.3.1 Introduction to XML document structure

XML, or Extensible Markup Language, is a flexible, text-based language designed by the W3C in 1998 for representing and exchanging structured information. Unlike HTML, which is primarily concerned with how data is displayed on a web page, XML focuses on describing the meaning of data. It provides a universal format that is both human readable and machine readable, making it ideal for sharing information across diverse systems, platforms, and applications. For example, <student><name>Ravi</name><age>20</age></student> immediately conveys that the information represents a student named Ravi who is 20 years old.

The usefulness of XML lies in its document structure, which ensures that data is stored in a consistent, logical, and hierarchical format. Every XML document typically begins with an optional declaration, such as <?xml version="1.0" encoding="UTF-8"?>, which specifies the XML version and character encoding. At the heart of the document is the root element, which acts as the container for all other elements. For instance, in a university database, <university> may serve as the root, enclosing multiple <student> records. Each child element represents specific data items, such as <name>, <department>, or <marks>, while attributes provide additional details like roll="101" inside a <student> tag.

XML content can be expressed in two ways: parsed character data (PCDATA), which is interpreted by the XML parser, and character data (CDATA), which is ignored by the parser and can contain special characters without conflict. Comments can also be added for documentation using the syntax <!-- comment -- >, and reserved characters like <, >, and & are represented with entity references such as <, >, and &. Together, these elements form a well structured document that resembles a tree when visualized, with the root at the top, branches for child elements, and leaves for actual data values.

XML documents can be classified as well formed or valid. A well formed document follows the basic syntax rules of XML, including having exactly one root element, properly nested and closed tags, and quoted attribute values. A valid document, on the other hand, is not only well formed but also adheres to additional constraints defined by a schema such as a Document Type Definition (DTD) or an XML Schema Definition (XSD). For example, while <book><title>XML Basics</title></book> is well-formed, it becomes valid only if it satisfies the rules specified in the associated schema about what elements and attributes are allowed.

In practice, XML plays a significant role in many real-world applications. In e-commerce, it is used to represent product catalogs and orders. In the banking sector, it facilitates the exchange of payment and transaction information. Healthcare relies on XML standards such as HL7 to exchange patient data between hospitals and laboratories. Even in education, XML is used to store and transfer student records and exam results. The clear structure of XML documents ensures accuracy, consistency, and portability, which is why XML remains one of the most important standards for structured data exchange.

4.3.1.1 Structure of an XML Document

An XML (eXtensible Markup Language) document follows a specific structure that makes it both machine readable and human readable. The structure ensures that data is organized systematically and can be exchanged across different platforms and applications. A well-formed XML document typically consists of several components such as an XML declaration, root element, child elements, attributes, and textual content. Each of these plays a significant role in defining the meaning and usability of the document.

1. XML Declaration (Optional but Recommended)

The XML declaration is usually the first line in an XML document. Although optional, it is strongly recommended because it provides important details about the document to XML parsers.

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- **version** → Specifies the version of XML being used (commonly "1.0").
- **encoding** → Indicates the character encoding used in the document (e.g., "UTF-8", "ISO-8859-1").
- **standalone (optional)** → Specifies whether the document depends on external definitions (DTD). "yes" means it is standalone, "no" means it requires external definitions.

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

2. Root Element

Every XML document must have exactly one root element that encloses all other elements. This root element acts as the top level container and defines the scope of the document.

Example:

```
<students>
  <student>John</student>
  <student>Mary</student>
</students>
```

Here, <students> is the root element.

3. Child Elements

Child elements are nested inside the root element (or inside other elements). They represent hierarchical data, allowing XML to structure information in tree form.

Example:

```
<student>
  <name>John</name>
  <age>21</age>
  <course>Computer Science</course>
</student>
```

- <student> → Parent element
- <name>, <age>, <course> → Child elements of <student>

4. Text/Data

The text or data inside XML elements represents the actual information being stored. XML does not focus on how the data looks but on what the data means.

Example:

```
<name>John</name>
Here, "John" is the text value of the <name> element.
XML also allows storing structured or mixed content:
<note>
  Remember to complete the <b>assignment</b> by tomorrow.
</note>
```

5. Attributes (Optional)

Attributes are name value pairs that provide additional information about elements. They appear inside the start tag of an element. Attributes are always enclosed in quotation marks (" " or ' ').

Example:

```
<student id="101" course="CS">John</student>
```

- id="101" → Attribute specifying the student's unique identifier.
- course="CS" → Attribute giving additional information about the student.
- "John" → Text content of the <student> element.

Attributes should generally be used for metadata (descriptive information), while main data should be kept inside elements.

A complete XML document combining all these components looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student id="101">
    <name>John</name>
    <age>21</age>
    <course>Computer Science</course>
  </student>
  <student id="102">
    <name>Mary</name>
    <age>22</age>
    <course>Mathematics</course>
  </student>
</students>
```


- **Declaration:** Defines XML version and encoding.
- **Root Element (<students>):** Encapsulates all student records.
- **Child Elements (<student>):** Each student is represented individually.
- **Attributes (id):** Provide unique identifiers.
- **Text/Data:** Contains actual names, ages, and courses.

The structure of an XML document is hierarchical, starting with an optional declaration, followed by a single root element containing child elements, text/data, and optional attributes. This structure makes XML flexible, platform-independent, and widely used for representing structured data in applications like databases, web services, and configuration files.

An XML document can be described in two ways: well formed or valid. A well formed XML document follows the basic syntax rules of XML, such as having a single root element, properly nested tags, quoted attributes, and closed tags. However, being well-formed alone does not guarantee that the data follows a particular structure or business logic. For that, XML provides validation mechanisms.

A valid XML document is one that is not only well formed but also conforms to a defined structure described by a Document Type Definition (DTD) or an XML Schema Definition (XSD). In other words:

- **Well formed XML** → Correct syntax.
- **Valid XML** → Correct syntax and follows the rules defined in a DTD or XSD.

This ensures that the XML data is both structurally correct and semantically meaningful, making it reliable for real-world applications such as databases, web services, and configuration files.

4.3.2 Well Formed XML Documents

An XML document is considered well formed when it follows the basic syntax rules specified by the World Wide Web Consortium (W3C) XML standard. These rules ensure that the document can be correctly parsed and understood by any XML parser, making it reliable for data exchange across applications, platforms, and systems. A well-formed XML does not guarantee semantic correctness (what the data means), but it guarantees structural correctness (how the data is represented).

4.3.2.1 Characteristics of a Well Formed XML Document

1. Single Root Element

Every XML document must have exactly one root element (also called the document element). This root encloses all other elements in the document. Without a single root, the structure becomes invalid.

Example (Correct):

```
<bookstore>
  <book>
    <title>XML Basics</title>
  </book>
</bookstore>
```

Here, <bookstore> is the root element.

Incorrect Example:

```
<book></book>
<author></author>
```

This is invalid because there are two top-level elements.

2. Properly Nested Tags

Elements must be properly nested within each other. Overlapping tags are not allowed.

Wrong Nesting:

```
<b><i>XML</b></i>
```

Correct Nesting:

```
<b><i>XML</i></b>
```

Proper nesting ensures that the document follows a hierarchical tree structure.

3. Case Sensitivity

XML is strictly case sensitive. For example, <Book> and <book> are treated as two different elements.

Example:

```
<Book>Correct</Book>
<book>Different</book>
```

Here, <Book> and <book> are not the same element, unlike HTML where case is often ignored.

4. All Tags Must Be Closed

Every opening tag in XML must have a corresponding closing tag. Failing to close a tag makes the document invalid.

Correct:

```
<author>John</author>
```

Incorrect:

```
<author>John
```

For empty elements, XML provides a self-closing syntax:

<linebreak />

5. Attributes Must Be Quoted

Attribute values must always be enclosed in either single (' ') or double (" ") quotes.

Correct:

```
<book category="Programming">
  <title>Learn XML</title>
</book>
```

Incorrect:

```
<book category=Programming>
```

This rule avoids ambiguity and ensures parser compatibility.

6. Proper Use of Special Characters

Since characters like <, >, and & have special meanings in XML, they cannot be used directly as data. Instead, they must be replaced with predefined entity references:

- < → <
- > → >
- & → &
- " → "
- ' → '

Example:

```
<note>
  <text>5 &lt; 10</text>
</note>
```

7. XML Declaration (Optional but Recommended)

An XML document generally begins with a declaration, which specifies the XML version, character encoding, and whether the document is standalone.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- **version** → Specifies the XML version (usually "1.0").
- **encoding** → Defines the character encoding (e.g., "UTF-8", "ISO-8859-1").
- **standalone** → "yes" if no external resources (like DTD) are required, "no" if they are.

8. Whitespace Handling

XML parsers preserve whitespace (spaces, tabs, newlines) inside element content. This means that unless whitespace is explicitly ignored or normalized, it will be treated as data.

Example of a Well-Formed XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <book id="B1">
    <title>XML Fundamentals</title>
    <author>John Smith</author>
    <price currency="USD">39.95</price>
  </book>
  <book id="B2">
    <title>Advanced XML</title>
    <author>Jane Doe</author>
    <price currency="USD">49.95</price>
  </book>
</library>
```


This document is well formed because:

- It has a single root element <library>.
- Tags are properly nested and closed.
- Attributes are quoted.
- No illegal characters are used.

Example of a Not Well Formed XML Document

```
<library>
  <book id=B1>
    <title>XML Fundamentals<title>
    <author>John Smith</Author>
</library>
```

Errors in this document:

- Attribute id is not quoted.
- <title> tag is not properly closed.
- Case mismatch in <author> vs </Author>.
- Root element is not properly structured.

Real-Life Application Example

Consider an online bookstore that uses XML to exchange book data between its website and a supplier. If the XML is not well-formed, the parser will reject the data, and books will not be displayed or processed correctly. For example, companies like Amazon, Flipkart, or library management systems rely on well formed XML for ensuring smooth data exchange and integration across different systems.

Thus, writing a well formed XML document is the foundation for building reliable, consistent, and portable data-driven applications.

4.3.3 Valid XML Documents

An XML document can be described in two ways: well-formed or valid. A well formed XML document follows the basic syntax rules of XML, such as having a single root element, properly nested tags, quoted attributes, and closed tags. However, being well-formed alone does not guarantee that the data follows a particular structure or business logic. For that, XML provides validation mechanisms.

A valid XML document is one that is not only well formed but also conforms to a defined structure described by a Document Type Definition (DTD) or an XML Schema Definition (XSD). In other words:

- **Well-formed XML** → Correct syntax.
- **Valid XML** → Correct syntax and follows the rules defined in a DTD or XSD.

This ensures that the XML data is both structurally correct and semantically meaningful, making it reliable for real world applications such as databases, web services, and configuration files.

4.3.3.1 Characteristics of a Valid XML Document

1. Well Formedness

Before validation, the document must satisfy all the rules of a well-formed XML:

- One root element.
- Properly nested and closed tags.
- Case sensitivity.
- Attributes quoted.
- Special characters properly encoded.

2. Validation Against a Defined Structure

Validation means checking whether the XML document follows a predefined structure or grammar. This structure can be defined using:

- **DTD (Document Type Definition):** Defines elements, attributes, and their relationships.
- **XSD (XML Schema Definition):** A more powerful and flexible schema language than DTD. It supports data types, namespaces, and more complex rules.

A valid XML document must match the rules specified in either a DTD or XSD.

Real-Life Applications of Valid XML

1. **Web Services:** SOAP messages must be valid XML according to WSDL/ XSD definitions.
2. **E-commerce:** Online stores (like Amazon, Flipkart) validate product XML data against schemas before displaying.
3. **Databases:** Many databases use XML for data import/export, requiring schema validation.
4. **Configuration Files:** Valid XML ensures applications load settings without errors.

4.3.4 Document Type Definition (DTD)

A Document Type Definition (DTD) is a set of rules that defines the structure, elements, and attributes of an XML document. It serves as a blueprint or schema for XML data, specifying which elements may appear, their order, the attributes they can have, and whether certain elements are mandatory or optional. By using a DTD, XML documents can be validated to ensure they conform to the expected structure an essential requirement for reliable data exchange between applications.

4.3.4.1 Purpose of DTD

The main purposes of using a DTD are:

1. **Validation:** Ensures the XML document adheres to the defined structure.
2. **Consistency:** Maintains uniformity across multiple XML documents of the same type.
3. **Interoperability:** Facilitates sharing of XML data between different systems with confidence that the structure will be recognized correctly.
4. **Documentation:** Serves as a formal description of the XML data format, making it easier for developers and systems to understand expected content.

4.3.4.2 Types of DTD

1. Internal DTD

An internal DTD is defined within the XML file itself, usually at the beginning, inside the <!DOCTYPE> declaration. This type of DTD is suitable for small documents or when the structure is specific to a single XML file.

Syntax Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book [
   <!ELEMENT book (title, author, price)>
   <!ELEMENT title (#PCDATA)>
   <!ELEMENT author (#PCDATA)>
   <!ELEMENT price (#PCDATA)>
]>
<book>
   <title>XML Basics</title>
   <author>John Smith</author>
   <price>39.95</price>
</book>
```

Explanation:

- <!DOCTYPE book [...]> → Declares the document type for <book>.
- <!ELEMENT book (title, author, price)> → Specifies that <book> contains exactly <title>, <author>, and <price> in that order.
- <!ELEMENTtitle(#PCDATA)>→<title>containsparsedcharacterdata(text).

Internal DTDs are simple to use but cannot be shared easily across multiple XML files.

2. External DTD

An external DTD is stored in a separate .dtd file and linked to the XML document. This is useful when multiple XML files need to follow the same structure, promoting reusability and consistency.

Syntax Example:

External DTD file (book.dtd):

```
<!ELEMENT book (title, author, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

XML Document Linking External DTD (book.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book>
  <title>Advanced XML</title>
  <author>Jane Doe</author>
  <price>49.95</price>
</book>
```

Explanation:

- <!DOCTYPE book SYSTEM "book.dtd"> → Links the XML file to an external DTD named book.dtd.

- Any changes made in book.dtd are automatically applied to all XML documents referencing it, making it efficient for large systems or multiple documents.

Advantages of Using DTD

1. **Standardization:** Ensures all XML files follow the same format.
2. **Validation:** Helps detect errors like missing elements, wrong order, or incorrect nesting.
3. **Reusability:** External DTDs can be applied to multiple XML files.
4. **Documentation:** Acts as a clear reference for developers to understand the structure of XML data.

Real Life Applications of DTD

- **E-commerce:** Ensures product XML data from suppliers adheres to a standard structure.
- **Publishing:** Used in news agencies or book catalogs to maintain consistent metadata.
- **Web Services:** Validates SOAP messages against a DTD before processing.
- **Configuration Management:** Ensures configuration files follow expected formats for ap

4.3.5 XML Schema Definition (XSD)

XML Schema Definition (XSD) is a powerful way to define the structure, content, and data types of an XML document. Unlike DTD, which primarily focuses on the structure and order of elements, XSD allows developers to specify detailed constraints on the data itself, making XML validation more robust and precise. XSD is written in XML syntax, which makes it self descriptive and compatible with XML parsers.

1. Defining Data Types

XSD supports a variety of built-in data types such as string, integer, decimal, date, boolean, and more. This ensures that the data in XML elements or attributes is of the expected type.

Example:

<xs:element name="age" type="xs:integer"/>

This ensures that the <age> element can only contain integer values.

2. Restricting Length, Pattern, and Value Ranges

XSD allows precise control over data through facets:

- **Length:** Restrict the number of characters in a string.
- **Pattern:** Enforce a specific format using regular expressions.
- **Min/Max Values:** Set limits for numeric data.

Example:

```
<xs:element name="zipCode">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{5}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Here, <zipCode> must be a 5-digit number.

3. Namespaces for Avoiding Conflicts

XSD supports XML namespaces, which allow multiple XML vocabularies to coexist without element name conflicts. Namespaces are particularly useful in large systems or web services where several schemas may interact.

Example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.example.com/books"
        xmlns:bk="http://www.example.com/books"
        elementFormDefault="qualified">
```

Advantages of XSD over DTD

1. **Data Type Support:** XSD allows defining integers, decimals, dates, and other types, whereas DTD treats all data as text.
2. **Better Validation:** XSD can restrict values using patterns, length, or min/max constraints.
3. **Namespaces:** Prevents element name conflicts in large XML systems.
4. **Extensibility:** XSD can define complex types, sequences, choices, and nested structures.

5. **XML-Based Syntax:** XSD itself is written in XML, making it readable and processable by standard XML parsers.

Example of XSD and Valid XML

XSD File (book.xsd):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="price" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Document (book.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="book.xsd">
  <title>Advanced XML</title>
  <author>Jane Doe</author>
  <price>49.95</price>
</book>
```

This XML document is valid because it follows the structure and data types defined in the XSD.

Real Life Applications of XSD

1. **Web Services:** Ensures SOAP or RESTful messages follow strict formats and data types.

2. **Financial Systems:** Validates transaction XML with specific numeric and date constraints.

3. **E-Commerce:** Guarantees product and order XML data following correct data types and ranges.

4. **Healthcare Systems:** Ensures medical records in XML conform to complex schemas with specific formats.

XSD is a powerful and flexible schema language for XML documents. It allows specifying data types, value constraints, and namespaces, making XML validation more precise than DTD. XSD is widely used in web services, financial systems, healthcare, and e-commerce for reliable and standardized data exchange.

4.3.6 XML Validation

XML Validation is the process of checking whether an XML document is both well-formed (follows XML syntax rules) and valid (follows a specific structure defined by a schema such as DTD or XSD). Validation is essential when XML is used for data exchange between applications, databases, and systems, ensuring that the data structure and content meet predefined rules.

- **Well formed XML:** Follows XML syntax rules (proper nesting, closing tags, quoted attributes, etc.).
- **Valid XML:** A well formed XML that also conforms to a DTD (Document Type Definition) or XSD (XML Schema Definition).

How Validation Works

1. Well-Formed Check
   - Performed by the XML parser.
   - Ensures proper syntax and structure.

Example of well-formed XML:

```
<student>
  <name>John</name>
  <age>22</age>
</student>
```

2. Validation Check
   - Performed against a DTD or XSD.

Ensures that elements, attributes, and data types match the schema definition.

Example with DTD validation:

```
<!DOCTYPE student [
  <!ELEMENT student (name, age)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT age (#PCDATA)>
]>
<student>
  <name>John</name>
  <age>22</age>
</student>
```

This XML is valid because it matches the DTD definition.

Benefits of XML Validation

1. Prevents Errors in Data Exchange
    - When systems communicate (e.g., web services, APIs, or e-commerce platforms), invalid XML could break communication. Validation ensures that the data structure is always correct.
    - Example: An online bookstore validating product listings before displaying them.
2. Ensures Data Integrity and Consistency
    - Validation guarantees that all required elements and attributes are present and that data follows the correct format.
    - Example: A banking XML file must always contain <accountNumber> and <balance> in the correct type (numeric).
3. Facilitates Interoperability Between Systems
    - Different applications can easily exchange data if they validate against the same schema.
    - Example: Healthcare systems exchanging patient data using HL7 XML standards validated through XSD.
4. Improves Reliability and Trustworthiness of Applications
    - Applications depending on XML data won't crash due to missing or malformed elements.
    - Ensures smooth workflow in enterprise systems.

Real Life Applications of XML Validation

- Web Services: SOAP/XML messages are validated against WSDL/XSD before being processed.
- E-Commerce: Product catalog XML is validated to prevent incomplete or incorrect listings.
- Databases: Data import/export in XML format is validated to maintain consistency.
- Configuration Files: Valid XML ensures software applications load settings correctly.

Example: XSD Validation

XSD File (student.xsd):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="age" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML File (student.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="student.xsd">
  <name>John</name>
  <age>22</age>
</student>
```

This XML is valid because it follows the structure and data types defined in the XSD.

Comparison of Well Formed vs Valid XML

Table 4.3.1 Comparison of Well-Formed vs Valid XML

Feature	Well-Formed XML	Valid XML
Syntax	Must follow XML syntax rules	Must follow XML syntax + DTD/XSD rules
Structure	Root element + nested tags	Matches DTD/XSD definitions
Parser Requirement	XML parser	XML parser + validator
Examples	Any correctly nested XML	XML conforming to DTD/ XSD

Recap

- XML is a markup language used for data exchange and representation.
- Structure includes: declaration, root, child elements, attributes, and text.
- Well formed XML → follows syntax rules.
- Valid XML → well formed + follows rules of DTD/XSD.
- DTD defines elements and attributes structure.
- XSD provides stronger validation with data types and constraints.
- XML validation ensures accuracy, interoperability, and reliability in real-world applications.

Objective Type Questions

1. Who developed XML in 1998?
2. What is the top most element in an XML document called?
3. Which XML section is optional but recommended at the beginning?
4. Which type of XML data is ignored by the parser and can hold special characters?
5. XML is strictly _____ (case sensitive or case-insensitive?).
6. Which schema language is more powerful, DTD or XSD?

7. What is the file extension of a DTD file?
8. Which organization standardized XML?
9. In XML, attributes must always be enclosed in _____.
10. Which XML component helps avoid element name conflicts in large systems?

Answers to Objective Type Questions

1. W3C
2. Root element
3. XML Declaration
4. CDATA
5. Case-sensitive
6. XSD
7. .dtd
8. W3C
9. Quotes
10. Namespace

Assignments

1. Define XML and explain its importance in data exchange with real life applications.
2. Differentiate between well formed XML and valid XML with suitable examples.
3. Write an XML document structure for a library database containing book records (title, author, price).
4. Explain DTD with an example. Differentiate between internal and external DTD.

5. Write a short note on XSD. Explain its advantages over DTD with an example.

6. Why is XML validation important in industries like banking, healthcare, and e-commerce? Illustrate with examples.

Reference

1. Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing* (3rd ed.). Pearson.

2. OpenAI. (2023). *Whisper: Robust Speech Recognition via Neural Networks.* Retrieved from https://openai.com/research/whisper

3. McTear, M. (2020). *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots.* Springer.

Suggested Reading

1. Manning, C. D., Raghavan, P., & Schütze, H. (2009). *Introduction to Information Retrieval*. Cambridge University Press.

2. Huang, X., Baker, J., & Reddy, R. (2014). *A Historical Perspective of Speech Recognition*. Communications of the ACM, 57(1), 94–103.

3. Articles on AI-powered productivity tools in *Harvard Business Review*.

XML Technologies and Tools

Learning Outcomes

After completing this unit, learners will be able to:

- define the purpose of XML and its supporting technologies
- describe the role of XPath, XSLT, XQuery, and DOM in working with XML documents
- identify the features, advantages, and limitations of each XML tool
- explain real-life applications of XML technologies in web, database, and business contexts

Prerequisites

In today's digital world, information is stored and exchanged in large volumes across different platforms. The challenge is that this information is often scattered, unorganized, and difficult to process efficiently. For example, online shopping sites, digital libraries, or banking systems deal with thousands of records that must be searched, updated, or displayed instantly. Without a systematic way to represent and manage such structured data, handling these operations becomes slow, error-prone, and confusing for both machines and users.

To overcome these issues, XML and its supporting technologies such as XPath, XSLT, XQuery, and DOM are studied. Studying these tools is important because XML alone is not sufficient for practical use, real-world applications require ways to navigate, query, transform, and manipulate XML documents. For example, in e-commerce platforms, product details are stored in XML, but XPath is needed to extract product information, and XSLT can transform that data into an HTML web page for customers. Similarly, XQuery is used in digital libraries to search millions of XML-based records, while DOM helps make web applications interactive, such as dynamically updating shopping cart totals. Thus, learning these technologies enables students to connect XML with real-life applications in web, business, and database systems.

Keywords

Transformation, Querying, Parsing, Node, Attribute, Tree Structure

Discussion

4.4.1 Introduction to XML Technologies and Tools

XML (Extensible Markup Language) is a platform-independent, structured data format designed to store, transport, and represent information in a human-readable and machine-processable manner. Although XML itself only provides a framework for representing data, working with XML in real-world applications requires additional technologies and tools to query, transform, navigate, and manipulate XML documents.

Over time, several standardized tools have been developed to enhance the usability of XML, the most important being:

- **XPath (XML Path Language):** A query language for navigating and selecting parts of an XML document.
- **XSLT (Extensible Stylesheet Language Transformations):** A transformation language for converting XML data into different formats such as HTML, text, or other XML documents.
- **XQuery (XML Query Language):** A functional language designed to query and extract data from XML, similar to how SQL queries work for relational databases.
- **DOM (Document Object Model):** A programming interface that represents XML (and HTML) documents as a hierarchical tree structure, allowing applications to create, access, modify, or delete elements dynamically.

These tools are essential because XML is widely used in:

- **Web applications** (for data interchange, e.g., SOAP, RSS feeds).
- **Databases** (storing semi-structured or hierarchical data).
- **Configuration files** (used in software, frameworks, and operating systems).
- **Business communication** (e.g., invoices, e-commerce transactions).

4.4.2 XPath (XML Path Language)

XPath, short for XML Path Language, is a standardized query language developed by the W3C for navigating and retrieving information from XML documents. Since XML represents data in a tree-like structure with nested elements, attributes, and text nodes, XPath provides a systematic way to locate and extract specific pieces of information.

Instead of manually searching through large XML files, XPath allows users to write simple path expressions that directly point to the required data, much like how a file path points to a particular file in a computer directory.

The primary reason for using XPath is that XML documents can often be large and complex, making manual data extraction impractical. XPath facilitates efficient navigation through different levels of the XML tree, allowing users to query elements, attributes, and specific text values. It is widely employed in areas such as XML data processing, web services, configuration management, and automation testing tools like Selenium, where it helps locate elements on web pages. By providing a precise mechanism for node selection, XPath ensures both accuracy and efficiency in data handling.

Syntax Basics

XPath uses a path-like notation, similar to file paths in a computer system. Here are the key syntax elements:

- / : Selects from the root node of the XML document.

  Example: /library/book selects all <book> elements directly under <library>.

- // : Selects nodes from the current node, regardless of their location in the document.

  Example: //title selects all <title> elements anywhere in the XML document.

- . : Refers to the current node.

  Example: ./author selects the <author> element relative to the current node.

- .. : Refers to the parent node of the current node.

  Example: ../title selects the <title> element of the parent node.

- @ : Selects attributes of elements.

  Example: //book[@id='b1'] selects the <book> element with attribute id="b1".

Features of XPath

1. **Path-like notation:** XPath expressions resemble file paths, making them easy to understand and use.

2. **Forward and backward navigation:** You can move between parent, child, ancestor, descendant nodes in the XML tree.

3. **Predicates and filters:** Use square brackets [ ] to refine searches.

   Example: //book[price>300] selects <book> elements with <price> greater than 300.

4. **Built-in functions:** Functions like count(), text(), contains(), starts-with() help perform advanced queries.

5. **Direct attribute selection:** Use @ to access attributes easily.

6. **Axes for advanced navigation:** Access nodes like preceding-sibling, following-sibling, ancestor, descendant, etc.

7. **Foundation for other technologies:** XPath is the backbone of XSLT and XQuery.

8. **Absolute and relative paths:** Supports both /root/element and //element selection.

Advantages of XPath

- Simple and intuitive syntax similar to file paths, easy to learn and apply.
- Powerful filtering and conditional selection using predicates.
- Supports functions for text matching, counting, and positional queries.
- Enables precise extraction of required data, saving time and resources.
- Works seamlessly with other XML tools like XSLT and XQuery.
- Platform- and application-independent; can be used with any XML-based system.
- Widely used in automation and testing frameworks (e.g., Selenium for locating web elements).

Limitations of XPath

- Works only with XML documents; not directly applicable to JSON or other formats.
- For large XML documents, complex queries can become slow and resource-heavy.
- Case-sensitive, which may cause issues if element and attribute names are not carefully matched.
- Cannot modify or update XML documents; only retrieves data (requires XSLT for transformations).
- May require additional knowledge of XML schemas (DTD/XSD) for complex queries.

Example

Consider XML:

```
<library>
  <book id="b1">
    <title>XML Fundamentals</title>
    <author>John Smith</author>
  </book>
  <book id="b2">
    <title>Data Science with Python</title>
    <author>Jane Doe</author>
  </book>
</library>
```

XPath Queries:

- /library/book/title

Output

XML Fundamentals

Data Science with Python

- //book[@id='b2']/author

Output

Jane Doe

- /library/book/title

/ means start from the root.

/library → selects the <library> element.

/book → selects all <book> child elements of <library>.

/title → selects the <title> element inside each <book>.

- //book[@id='b2']/author

//book → selects all <book> elements anywhere in the document.

[@id='b2'] → filters the <book> element that has id="b2".

/author → selects the <author> child of that book.

4.4.3 XSLT (Extensible Stylesheet Language Transformations)

XSLT (Extensible Stylesheet Language Transformations) is a language used to transform XML documents into other formats. It takes XML data and converts it into HTML, plain text, or another XML document. Think of XML as raw ingredients (data) and XSLT as the cooking recipe that tells how to present the data in the final dish (output).

XSLT is primarily used to transform XML data into various formats such as HTML for displaying information on web pages, to generate text reports from XML, to restructure XML data for exchange between systems, and to filter or sort XML content based on specific conditions. It facilitates the separation of data (XML) from its presentation (HTML, text, etc.), thereby enhancing system flexibility and reusability.

Some important features of XSLT are that it is template-based, meaning it uses templates and rules to decide how elements in XML should appear in the output. It also supports XPath expressions, which are powerful ways to navigate and select parts of the XML document. XSLT allows sorting and filtering of data, making it possible to rearrange or limit the output according to certain conditions. Another feature is that it supports multiple output formats, so the same XML can be transformed into HTML, text, or XML depending on the requirement. Overall, XSLT makes XML more practical by turning raw data into meaningful and user-friendly presentations.

Features of XSLT

1. **Template-based transformation**
   - Uses rules called templates to decide how each XML element should appear.

     Example: <xsl:template match="book">...</xsl:template>

2. **Supports XPath**
   - Uses XPath expressions to find and select specific elements in XML.

     Example: book/title selects the title of each book.

3. **Sorting and Filtering**
   - Can sort data alphabetically, numerically, or by conditions.

     Example: <xsl:sort select="title"/>

4. **Multiple Output Formats**
   - Can produce HTML(forweb), Text(forreports), or XML(forsystemexchange).

5. **Separation of Data and Presentation**

- Keeps XML for storing data and XSLT for presentation. This makes it flexible and reusable.

Working of XSLT

The working of XSLT involves four main components that work together to transform XML data into a desired format. First, we have the XML document, which acts as the input and contains the raw data to be transformed. Next is the XSLT stylesheet, which holds the set of rules and templates that define how the XML data should be displayed or structured in the output. Then comes the XSLT processor, a software engine that reads both the XML document and the XSLT stylesheet, applies the rules, and carries out the transformation. Finally, the processor generates the output, which can be in different formats such as HTML (for web pages), plain text (for reports), or another XML document (for data exchange).

The transformation process involves four parts:

1. **XML Document** : The input data (e.g., book information).
2. **XSLT Stylesheet** : The set of rules/templates to transform XML.
3. **XSLT Processor** : The engine that applies the stylesheet to XML.
4. **Output :** The final result (HTML, Text, or XML).

Advantages of XSLT

- Separates data from presentation.
- Easy to reuse rules for different XML data.
- Supports filtering, sorting, and grouping.
- Works well with other XML technologies like XPath.

Limitations of XSLT

- Learning curve can be a bit hard for beginners.
- Complex transformations may become lengthy.
- Browser support is mostly limited to XSLT 1.0.

Example

XML:

```
<book>
  <title>XML Basics</title>
  <author>John Smith</author>
</book>
```

XSLT Stylesheet:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Book Information</h2>
        <p>Title: <xsl:value-of select="book/title"/></p>
        <p>Author: <xsl:value-of select="book/author"/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Output (HTML):

```
<h2>Book Information</h2>
<p>Title: XML Basics</p>
<p>Author: John Smith</p>
```

Explanation of Example

1. XML file stores raw data: book title and author.
2. The XSLT file has a template that tells how to display data.
   - <xsl:value-of select="book/title"/> picks up the title from XML.
   - <xsl:value-of select="book/author"/> picks up the author.

3. The processor applies the XSLT to XML.

4. Output is an HTML page showing book details neatly.

4.4.4 XQuery (XML Query Language)

XQuery is a powerful query language specially designed for working with XML data. Just like SQL is used for querying relational databases, XQuery is used to query, search, and manipulate data stored in XML documents. It allows developers and analysts to extract meaningful information from large XML datasets easily. XQuery is built on top of XPath, which means it can navigate through XML structures efficiently while also providing more advanced query capabilities.

We use XQuery because XML is widely used for storing and transferring structured data across different systems. For example, online bookstores, financial systems, and web services often store data in XML format. With XQuery, we can filter, sort, join, and transform XML data just like we do with SQL in relational databases. This makes it very useful for applications that depend on XML databases, SOAP-based web services, and content management systems.

Features of XQuery

Query works very much like SQL, but is designed specifically for querying XML documents. It is built on top of Xpath which allows it to easily navigate through XML trees. XQuery provides powerful features such as filtering, sorting, joining and restructuring XML data based on the user's needs. It can be used both for querying XML documents and XML databases. Another important feature is its flexibility to transform XML into other formats like HTML, plain text or JSON

Advantages of XQuery

1. **Powerful Querying** : Provides SQL-like power to query XML documents, making it easy to search and manipulate data.

2. **Platform Independent** : Works on any system that uses XML, ensuring cross-platform compatibility.

3. **Integration** : Can be integrated with web services, XML databases, and business applications.

4. **Data Transformation** : Converts XML into other formats such as HTML or JSON for display or further processing.

5. **Human Readable** : The syntax is clear and easier for developers who already know SQL or XPath.

Limitations of XQuery

1. **Performance Issues** : For very large XML files, XQuery can be slower compared to optimized SQL queries on relational databases.

2. **Complex Syntax for Beginners** : While easier than raw XML parsing, it may still be difficult for beginners to master.

3. **Dependency on XML** : Works only with XML data, so it is not useful if data is stored in other formats like CSV or relational databases.

4. **Limited Tool Support** : Compared to SQL, fewer commercial tools provide full-fledged XQuery support.

Example

XML:

```
<library>
  <book>
    <title>XML Fundamentals</title>
    <price>300</price>
  </book>
  <book>
    <title>Python Programming</title>
    <price>500</price>
  </book>
</library>
```

XQuery:

```
for $x in /library/book
where $x/price > 300
return $x/title
```

Output:

Python Programming

4.4.5 DOM (Document Object Model)

The Document Object Model (DOM) is a programming interface that represents XML or HTML documents as a tree structure. In this structure, each part of the document—such as elements, attributes, and text is treated as a node. This allows programmers to easily access, navigate, and modify the contents of XML or HTML documents using different programming languages like JavaScript, Java, Python, and C#.

In simple words, DOM acts as a bridge between the document and programming languages, making documents dynamic and interactive.

We use the Document Object Model (DOM) when we need to work with XML or HTML documents in a flexible and interactive way. It is especially useful in applications that require dynamic changes—such as updating XML data without reloading the entire document or manipulating HTML elements on web pages in real time (for example, changing text, colors, or layouts using JavaScript).

For example, in XML-based applications, DOM can be used to add new elements, delete old ones, or update attribute values. Similarly, in websites, DOM allows developers to make web pages interactive.

Features of DOM

- **Language-Independent:** Provides a standard API that can be used with many languages (JavaScript, Java, Python, etc.).
- **Tree Representation:** Represents XML or HTML documents as a hierarchical tree of nodes.
- **Full Document Access:** Allows navigation to any part of the document (element, attribute, or text).
- **Modifications Possible:** Nodes can be added, updated, or deleted at runtime.
- **Dynamic Updates:** Useful for web browsers and XML applications that need real-time changes.

Advantages of DOM

1. **Easy Navigation** : The tree structure makes it simple to move through elements and attributes.
2. **Dynamic Modifications** : Content can be added, changed, or removed at any time.
3. **Cross-Language Support** : Works with multiple programming languages.
4. **Good for Small/Medium Documents** : Provides full flexibility for documents that are not too large.

Limitations of DOM

1. **Memory Intensive** : For very large XML/HTML documents, storing everything in memory as a tree can be costly.
2. **Performance Issues** : Parsing and manipulating large documents can be slower compared to event-based models like SAX.

3. **Complex for Beginners** : Navigating and modifying nodes can be confusing for new learners.

4. **Not Always Needed** : If only reading simple XML data, DOM may be overkill compared to simpler methods.

Example (JavaScript DOM with XML)

```
// Parse XML string
var parser = new DOMParser();
var xmlString = "<book><title>XML Basics</title></book>";
var xmlDoc = parser.parseFromString(xmlString, "text/xml");
// Access element
var title = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
console.log(title); // Output: XML Basics
```

Table 4.4.1 Comparison Table of XML Technologies

Technology	Purpose	Based On	Output	Example Use
XPath	Navigate and select nodes	Path expressions	Node(s)	Selecting book titles
XSLT	Transform XML into other formats	Templates + XPath	XML/HTML/ Text	Convert XML to HTML
XQuery	Query XML data	XPath + SQL-like syntax	XML or data set	Query XML database
DOM	Programmatic access to XML	Tree structure API	XML object model	Modify XML dynamically

Real-Life Applications

- **XPath:** Extract specific information (e.g., price of a product in XML feed).
- **XSLT:** Convert XML-based reports into readable HTML pages.
- **XQuery:** Retrieve structured data from XML databases like BaseX, eXist-db.
- **DOM:** Manipulate XML in web applications dynamically (like JavaScript handling HTML).

Recap

- XML by itself is a data representation language; additional tools make it usable in applications.
- XPath: Used for selecting specific nodes or attributes from XML documents.
- XSLT: A stylesheet language that transforms XML into other formats such as HTML or text.
- XQuery: A query language for XML, works like SQL for databases, enabling filtering, sorting, and joining.
- DOM: A programming interface representing XML/HTML as a tree structure, allowing modifications.
- Comparison: XPath (navigation), XSLT (transformation), XQuery (querying), DOM (programming access).
- Real-life use: Web data exchange, XML-based databases, business communication, and web page manipulation.

Objective Type Questions

1. What structure does the DOM use to represent XML or HTML documents?
2. Which XML tool is used to navigate and select specific nodes in a document?
3. What is the main purpose of XSLT in XML processing?
4. XQuery is similar to which language but designed for XML?
5. What type of API does DOM provide?
6. In DOM, how is each element, attribute, and text represented?
7. Which XML technology allows filtering, joining, and transforming data?
8. Which technology helps in making real-time dynamic updates to XML documents?
9. Which XML technology uses templates and rules to restructure XML content?
10. On which base technology is XQuery built?

Answers to Objective Type Questions

1. Tree structure
2. XPath
3. Transforming XML documents into other formats such as HTML, Text, or XML
4. SQL
5. Language-independent API
6. Node
7. XQuery
8. DOM
9. XSLT
10. XPath

Assignments

1. Explain the role of XPath in navigating XML documents with suitable examples.
2. Write an example XSLT stylesheet to transform XML student data into an HTML table.
3. Compare XQuery with SQL. Give one real-life use case where XQuery is preferable.
4. Discuss the advantages and limitations of DOM with an example.
5. Prepare a comparison table of XPath, XSLT, XQuery, and DOM in your own words.

Reference

1. Grimmer, A., & Petersen, S. (2023). *Beginning XML: From Novice to Professional.* Apress.

2. Harold, E. R., & Means, W. S. (2022). *XML in a Nutshell.* O'Reilly Media.

3. Kay, M. (2022). *XSLT 3.0 Programmer's Reference*. Wiley.

Suggested Reading

1. Chamberlin, D., Robie, J., & Florescu, D. (2013). *XQuery: A query language for XML*. Morgan Kaufmann.

2. Apparao, V., Byrne, P., Champion, M., Isaacs, I., Jacobs, I., Le Hors, A., … Wilson, C. (2000). *Document Object Model (DOM) Level 2 Core Specification*. World Wide Web Consortium (W3C).

3. Tidwell, D. (2008). *Beginning XML* (4th ed.). Wrox.

Practicing Web Development

Introduction

Web technology has become the foundation of modern communication, business, and education. The HTML and JavaScript Lab introduces students to the essential tools and techniques of web development. HTML (HyperText Markup Language) defines the structure and content of a webpage, allowing designers to create headings, paragraphs, tables, images, and links. With the integration of Cascading Style Sheets (CSS), students learn to enhance visual presentation through color, layout, and typography. JavaScript, the scripting language of the web, adds interactivity and functionality then turning static pages into dynamic applications. By mastering these technologies, students gain the skills to build responsive, visually appealing, and user-friendly websites. This lab encourages creativity, experimentation, and critical thinking—preparing students for real-world web development, software engineering, and digital design challenges.

Objectives of the HTML and JavaScript Lab

- Understand the structure and syntax of HTML for webpage creation.
- Design and style web pages using Cascading Style Sheets (CSS).
- Incorporate multimedia elements like images, audio, and video into webpages.
- Apply JavaScript for interactive functionalities such as form validation and dynamic content.
- Develop the ability to combine HTML, CSS, and JavaScript in building complete websites.
- Encourage logical thinking, creativity, and problem-solving through practical exercises.

Tools and Software Used for Web Programming

The following tools are commonly used in this lab to design, develop, and test web applications:

1. **Integrated Development Environments (IDEs)**

- **Visual Studio Code (VS Code):** A lightweight, feature-rich editor ideal for web development.
- **Sublime Text:** Fast and simple text editor for writing clean HTML, CSS, and JavaScript.
- **Notepad++:** A basic editor suitable for beginners learning HTML and JavaScript syntax.
- **Brackets:** An open-source editor designed specifically for front-end development.

- **CodePen / JSFiddle:** Online platforms for experimenting with HTML, CSS, and JavaScript code interactively.

2. Web Browsers

Google Chrome, Mozilla Firefox, Microsoft Edge: Used to test and debug web applications.

3. Testing and Validation Tools

- W3C HTML Validator : for validating HTML syntax.
- CSS Validator : for checking style sheet correctness.
- Browser Developer Tools : for inspecting code and debugging JavaScript.

Scope of the Lab

This lab provides a strong foundation in the basics of web development. It enables students to design structured, visually appealing, and interactive web pages using HTML, CSS, and JavaScript. Through step-by-step experiments, students learn to create everything from simple pages to functional websites. The lab focuses on hands-on practice, bridging theory with real-world application. It also encourages exploration of modern web technologies, responsive design, and front-end frameworks—skills that are essential for careers in software development, UI/UX design, and data visualization.

Expectations from Students

1. Students must actively participate in all lab sessions by engaging in coding exercises, discussions, and practical applications to deepen their understanding of Python. They should come prepared by reviewing the lab manual and any assigned materials beforehand to approach tasks with confidence.

2. Consistency in completing and submitting lab assignments on time is essential for reinforcing programming skills and building technical proficiency.

3. Collaboration with peers is encouraged, as working on group projects and coding challenges fosters teamwork and enhances problem-solving abilities.

4. Students should take the initiative to explore beyond the structured exercises, ask questions, and apply their learning to real-world applications.

5. Maintaining discipline and focus throughout the lab sessions will ensure maximum learning and retention of concepts.

6. They should demonstrate curiosity and engagement by experimenting with Python's capabilities and integrating their knowledge into practical coding projects.

7. By fulfilling these expectations, students will develop a strong foundation in Python programming that will be valuable for future studies and professional endeavors.

8. Students are encouraged to bring their own datasets or find publicly available ones to explore during lab sessions. They should also document their analysis process and share visual reports or dashboards as part of project submissions.

Lab Guidelines

1. Read and understand the lab manual, experiment guidelines, and instructor directions before starting any task.

2. Keep the workspace clean and organized to prevent any interference during practical exercises.

3. Use only the approved software and tools specified by the instructor or lab administrator.

4. Ensure that all lab exercises and coding assignments are completed and submitted within the given deadlines.

5. Write original code and follow ethical programming practices, avoiding plagiarism.

6. Participate actively in discussions and collaborative activities to improve understanding and problem-solving skills.

7. Seek help from the instructor or peers when facing challenges to clarify doubts and enhance learning.

8. Regularly save and back up coding projects to prevent data loss due to unexpected errors or system failures.

9. Explore beyond the basic exercises by experimenting with new concepts and applying Python to real-world scenarios.

10. Submit lab assignments and reports within the specified deadlines to avoid penalties.

11. Ensure all submitted work is original, avoiding plagiarism or copying from others to maintain academic integrity.

12. Prepare for viva or oral exams to effectively demonstrate understanding of the experiments conducted.

13. Non-compliance with lab guidelines, misuse of lab equipment, or engaging in unethical practices will result in disciplinary measures, which may include:

- Warnings.
- Deduction of grades.
- Suspension from lab sessions.

14. All data used in lab projects should comply with data privacy laws and ethical guidelines. Students must avoid using sensitive or personal information unless explicit permission is granted and anonymization measures are applied.

Code of Conduct

- Be respectful to instructors, lab assistants, and peers.
- Always seek help when required, but do not disrupt others' work.
- Strive to develop problem-solving skills and explore advanced features of DBMS tools.
- Respect the integrity of datasets and avoid misrepresenting results.
- Clearly label visualizations, cite sources, and maintain transparency in analyticmethods.

Setting up the Web Technology Lab

To practice HTML, CSS, and JavaScript effectively, students need a proper development environment. This section guides you through setting up the necessary tools and software.

Step 1: Install a Code Editor

A code editor is required to write and test HTML, CSS, and JavaScript code. Some recommended editors are:

1. **Visual Studio Code (VS Code)**
   - A lightweight and powerful editor with extensive extensions for HTML, CSS, and JavaScript.
   - Supports live preview, debugging, and Git integration.
   - Ideal for both beginners and advanced users.
2. **Sublime Text**
   - Fast and user-friendly editor with syntax highlighting.
   - Suitable for quick edits and learning HTML and CSS basics.

3. **Brackets**

   - Open-source editor designed for front-end development.
   - Offers live preview and visual tools for HTML and CSS design.

4. **Notepad++**

   - A simple editor for Windows users, suitable for learning HTML, CSS, and JavaScript syntax.

5. **Online Editors (CodePen, JSFiddle, JSBin)**

   - Cloud-based platforms for writing and running HTML, CSS, and JavaScript code without installing any software.
   - Useful for experimentation and sharing small projects.

Step 2: Install a Web Browser

Modern web browsers are essential to run and test your web pages. Recommended browsers include:

- **Google Chrome** : Fast, secure, and supports developer tools for debugging.
- **Mozilla Firefox** : Provides robust development tools and extensions.
- **Microsoft Edge** : Integrates well with Windows systems and has built-in developer tools.

Step 3: Verify Your Environment

1. Open your chosen code editor and create a new HTML file.
2. Save the file with a .html extension (e.g., index.html).
3. Open the file in a web browser to verify that your environment is correctly set up.
4. Test simple HTML code like headings, paragraphs, and links to ensure everything works properly.

Step 4: Optional Tools and Extensions

To enhance your coding experience, the following tools and extensions are recommended:

- **Live Server Extension (VS Code)** : Launches a live preview of your HTML page in the browser.
- **Emmet Plugin** : Speeds up HTML and CSS coding with shortcuts and templates.
- **Linting Tools (HTMLHint, CSSLint, JSHint)** : Check code quality and catch errors early.

Experiment No: 1

Title: Practicing basic HTML tags, text tags, paragraph styles, headings, and lists

Objectives:

- To understand the structure of an HTML document
- To learn and implement basic HTML tags such as headings, paragraphs, and text formatting tags
- To create and practice different types of lists (ordered, unordered, and definition lists)
- To develop the ability to design simple structured web pages using basic HTML elements

Theory

1.1.1 Introduction to HTML

HTML (HyperText Markup Language) is the standard language used to create web pages. It defines the structure of web content using elements represented by tags. An HTML document is composed of elements such as headings, paragraphs, text formatting, and lists.

Basic structure of an HTML document:

```
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <!-- Content goes here -->
</body>
</html>
```

1.1.2 Basic HTML Tags

1. **Headings (<h1> to <h6>)**

- Used to define headings of different sizes.

- <h1> is the largest heading, <h6> is the smallest.

Example:

```
<h1>Main Heading</h1>
<h2>Sub Heading</h2>
<h3>Smaller Heading</h3>
```

2. Paragraphs (<p>)

- Defines blocks of text.

Example:

```
<p>This is a simple paragraph in HTML.</p>
```

3. Text Formatting Tags

- <b> or → Bold text
- <i> or → Italic text
- <u> → Underline
- <mark> → Highlight text
- <sup> / <sub> → Superscript / Subscript

Example:

```
<p>This is <b>bold</b>, <i>italic</i>, and <u>underlined</u> text.</p>
```

4. Paragraph Styles

- Inline styles or CSS can be used to change paragraph appearance.

Example:

```
<p style="color:blue; font-size:18px; text-align:center;">
  This is a styled paragraph.
</p>
```

5. Lists in HTML

- Ordered List (<ol>) – Numbered items.
- Unordered List (<ul>) – Bulleted items.

- Definition List (<dl>) – Term and description pairs.

Example:

```
<h3>Ordered List</h3>
<ol>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ol>
<h3>Unordered List</h3>
<ul>
  <li>Apple</li>
  <li>Banana</li>
  <li>Orange</li>
</ul>
<h3>Definition List</h3>
<dl>
  <dt>HTML</dt>
  <dd>HyperText Markup Language</dd>
  <dt>CSS</dt>
  <dd>Cascading Style Sheets</dd>
</dl>
```

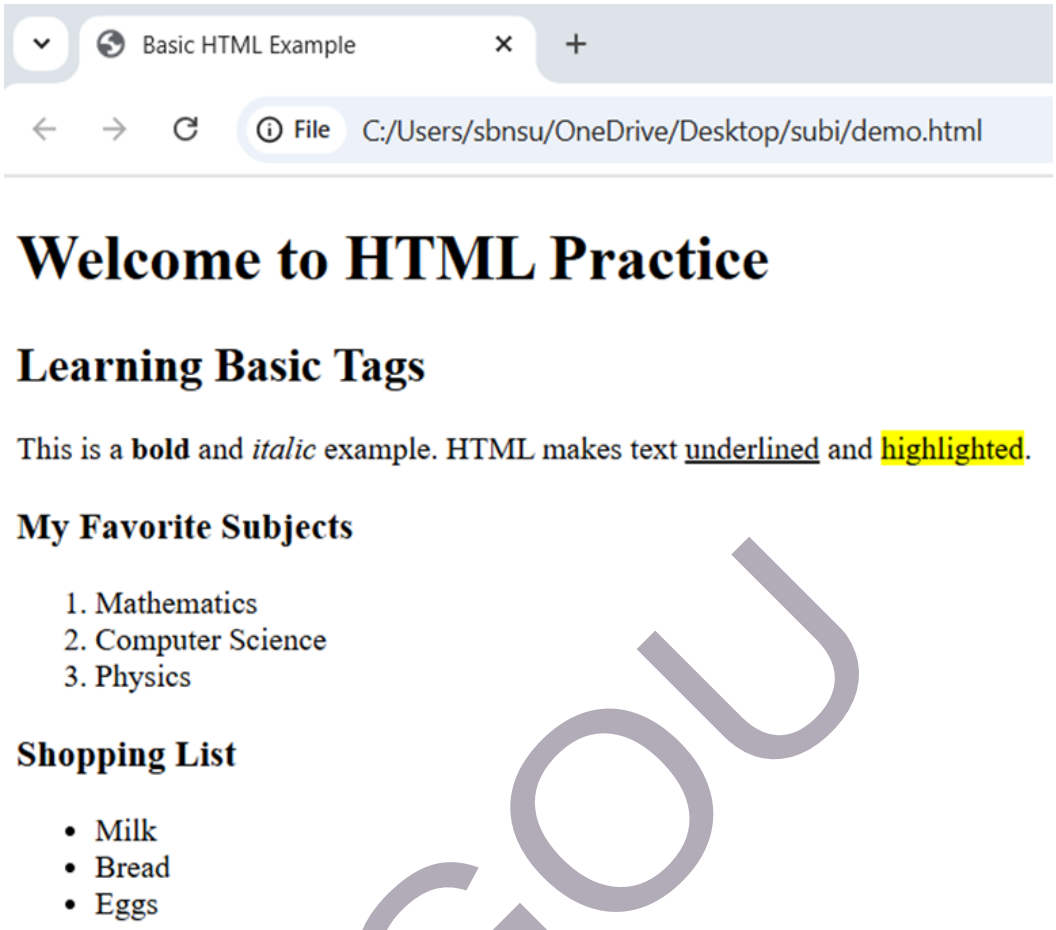

1.1.3 Sample questions

1. Write an HTML program to demonstrate the use of basic HTML tags, text formatting, and lists. Your program should include:

    1. A main heading (<h1>) and a subheading (<h2>).
    2. A paragraph showing bold, italic, <u>underlined</u>, and <mark>highlighted</mark> text.
    3. An ordered list (<ol>) of your favorite subjects.
    4. An unordered list (<ul>) of shopping items.
    5. A suitable <title> inside the <head>.

```
<html>
<head>
  <title>Basic HTML Example</title>
</head>
<body>
  <h1>Welcome to HTML Practice</h1>
  <h2>Learning Basic Tags</h2>
  <p>This is a <b>bold</b> and <i>italic</i> example.
    HTML makes text <u>underlined</u> and <mark>highlighted</mark>.
  </p>
  <h3>My Favorite Subjects</h3>
  <ol>
    <li>Mathematics</li>
    <li>Computer Science</li>
    <li>Physics</li>
  </ol>
  <h3>Shopping List</h3>
  <ul>
    <li>Milk</li>
    <li>Bread</li>
    <li>Eggs</li>
  </ul>
</body>
</html>
```

Output



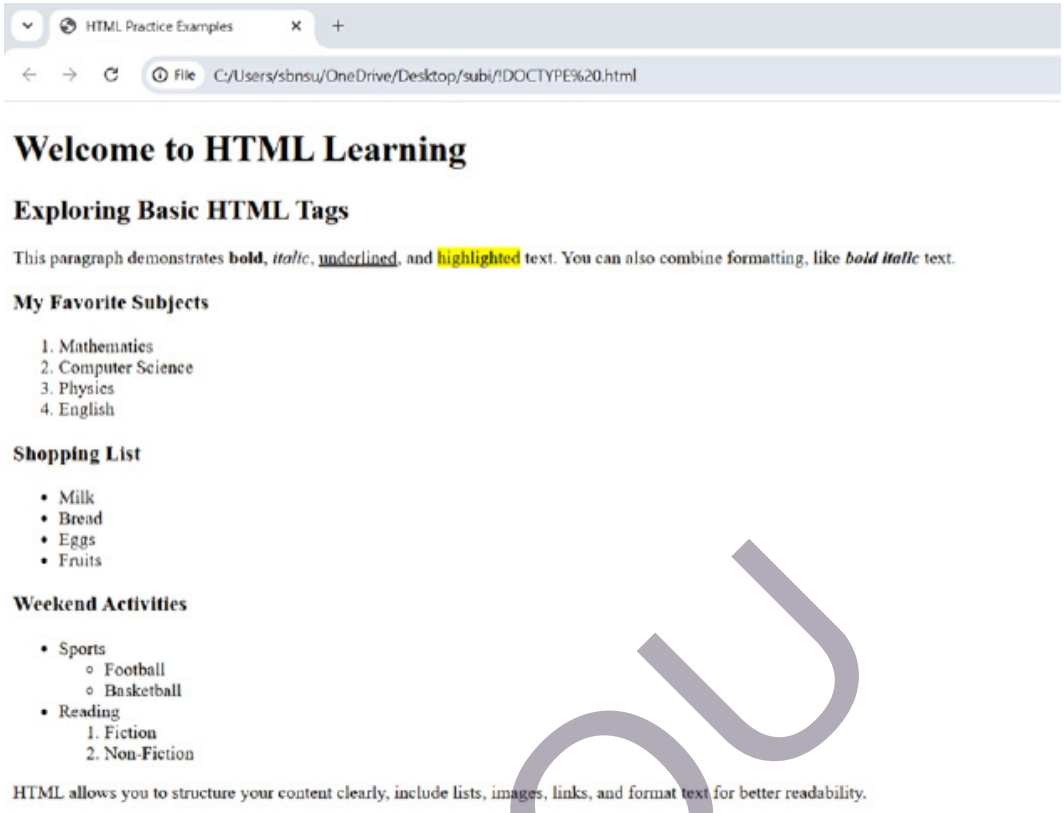
2. Write an HTML program to demonstrate the use of basic HTML tags, text formatting, and lists. Your program should include the following:

   1. A main heading (<h1>) and a subheading (<h2>).
   2. A paragraph that shows text in bold, italic, <u>underlined</u>, and <mark>highlighted</mark> formats.
   3. An ordered list (<ol>) of at least three of your favorite subjects.
   4. An unordered list (<ul>) of at least three shopping items.
   5. A suitable <title> inside the <head> section.

```
<html>
<head>
   <title>HTML Practice Examples</title>
</head>
<body>
   <!-- Main Heading and Subheading -->
   <h1>Welcome to HTML Learning</h1>
   <h2>Exploring Basic HTML Tags</h2>
   <!-- Paragraph with text formatting -->
   <p>
      This paragraph demonstrates <b>bold</b>, <i>italic</i>,
      <u>underlined</u>, and <mark>highlighted</mark> text.
      You can also combine formatting, like <b><i>bold italic</i></b> text.
   </p>
     <!-- Ordered List -->
   <h3>My Favorite Subjects</h3>
   <ol>
      <li>Mathematics</li>
      <li>Computer Science</li>
      <li>Physics</li>
      <li>English</li>
   </ol>
   <!-- Unordered List -->
   <h3>Shopping List</h3>
   <ul>
      <li>Milk</li>
      <li>Bread</li>
      <li>Eggs</li>
      <li>Fruits</li>
   </ul>
   <!-- Nested Lists -->
```

```
    <h3>Weekend Activities</h3>
    <ul>
        <li>Sports
            <ul>
                <li>Football</li>
                <li>Basketball</li>
            </ul>
        </li>
        <li>Reading
            <ol>
                <li>Fiction</li>
                <li>Non-Fiction</li>
            </ol>
        </li>
    </ul>
    <!-- Another Paragraph -->
    <p>
        HTML allows you to structure your content clearly, include lists, images, links,
        and format text for better readability.
    </p>
</body>
</html>
```

Output



1.1.4 Lab Practice Questions

1. Write an HTML program that displays your name using all heading tags (<h1> to <h6>).

2. Create an HTML page with two paragraphs:

   - First paragraph in blue color and center aligned.
   - Second paragraph in red color and bold font.

3. Write an HTML program that demonstrates the following text styles: bold, italic, underline, superscript, and subscript.

4. Design an HTML page that displays an ordered list of your top 5 favorite movies and an unordered list of your hobbies.

Experiment No:2

Title: Tables in HTML, nested tables, Frames in HTML, nested frames, Link and Anchor Tags

Objectives:

- understand the structure and usage of HTML tables to organize and present tabular data effectively
- demonstrate the ability to create nested tables within an HTML document for complex layouts
- learn how to use HTML frames and nested frames to divide the browser window into multiple independent sections
- implement hyperlinks using anchor (<a>) tags to navigate between web pages and within the same page
- apply attributes such as href, target, border, and frameset to control the behavior and appearance of links and frames

Theory

1.2.1 Overview

1. **Tables in HTML**
   - Tables are created using the <table> tag.
   - Rows are defined with <tr>, and cells with <td> (data) or <th> (header).
   - Attributes like border, cellspacing, cellpadding, width, and height control appearance.
2. **Nested Tables**
   - A table placed within a <td> cell of another table is called a nested table.
   - Useful for creating complex layouts or grouping related data visually.
3. **Frames in HTML**
   - Frames divide the browser window into multiple sections, each loading a separate HTML document.
   - <frameset> tag replaces <body> and defines frame structure.
   - frame> tag loads individual pages within each frame.

4. **Nested Frames**

   - Framesets can be nested to create complex layouts (e.g., splitting vertically and then horizontally).
   - Requires careful use of <frameset> within another <frameset>.

5. **Links and Anchor Tags**

   - The <a> tag is used to create hyperlinks. Syntax: Link Text
   - href defines the target URL.
   - target attribute specifies where to open the link (_blank, _self, _parent, _top).
   - Anchor tags can also create internal page navigation using the id attribute and #.

Sample Questions

1. Create an HTML table with 3 rows and 3 columns.The first row should be headers.

```
<html>
<head><title>Simple Table</title></head>
<body>
 <table border="1">
  <tr>
   <th>Name</th>
   <th>Age</th>
   <th>City</th>
  </tr>
  <tr>
   <td>John</td>
   <td>25</td>
   <td>New York</td>
  </tr>
  <tr>
   <td>Jane</td>
   <td>30</td>
   <td>Los Angeles</td>
  </tr>
 </table>
</body>
</html>
```

Output

Name	Age	City
John	25	New York
Jane	30	Los Angeles

2. Create a table where one of the cells contains another table.

```
<html>
<head><title>Nested Table</title></head>
<body>
 <h3>Nested Table Example</h3>
 <table border="1">
  <tr>
   <td>Main Table - Row 1, Cell 1</td>
   <td>
    <table border="1">
     <tr>
      <td>Nested Row 1</td>
     </tr>
     <tr>
      <td>Nested Row 2</td>
     </tr>
    </table>
   </td>
  </tr>
 </table>
</body>
</html>
```


Output

Nested Table Example

Main Table - Row 1, Cell 1	Nested Row 1 Nested Row 2

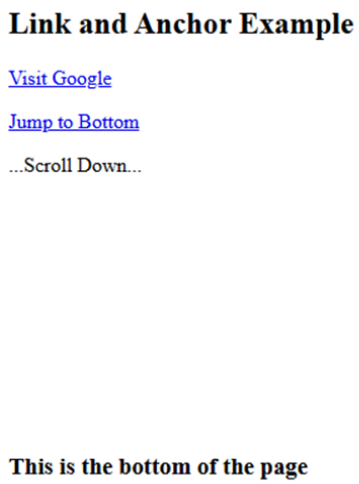
3. Create a layout where the left frame stays fixed and the right side is split into two horizontal frames.

```
<html>
<frameset cols="30%,70%">
  <frame src="left.html">
  <frameset rows="50%,50%">
    <frame src="top.html">
    <frame src="bottom.html">
  </frameset>
</frameset>
</html>
```

4. Create a webpage with links : one to another website ,and one to an internal section using an anchor tag.

```
<html>

<head><title>Links and Anchors</title></head>

<body>

  <h2>Link and Anchor Example</h2>

  <!-- External link -->

  <p><a href="https://www.google.com" target="_blank">Visit Google</a></
p>

  <!-- Anchor link to internal section -->

  <p><a href="#bottom">Jump to Bottom</a></p>

  <p>...Scroll Down...</p>

  <br><br><br><br><br><br><br><br><br><br>

  <h3 id="bottom">This is the bottom of the page</h3>

</body>

</html>
```

Output



Lab Practice Questions

1. Design a nested table where one of the table cells contains another table showing detailed information like subject marks.
2. Use an anchor tag to jump to a specific section within a long webpage.
3. Create hyperlinks that:
   - Open another webpage in the same tab.
   - Open a link in a new tab.
   - Download a file from a given URL.
4. Create a frame-based structure with three sections: a header at the top, a menu on the left, and content on the right.
5. Link multiple pages together using the <a> tag to simulate navigation between Home, About, and Contact pages.

Experiment No: 3

GRAPHICS ON WEBPAGES

Objectives

- To understand the methods of embedding multimedia elements such as images, audio, and video into web pages using HTML tags
- To learn and apply HTML attributes (*src, controls, autoplay, loop,* etc.) for proper display and control of multimedia content
- To demonstrate the integration of graphics in various formats (*JPEG, PNG, GIF*) for enhancing the visual appeal of web pages
- To explore embedding techniques for different video and audio formats using <*video*> and <*audio*> elements and external sources
- To design an interactive and media-rich web page that effectively combines text, graphics, video, and sound for improved user experience

Theory

A web page becomes more attractive and engaging when it includes multimedia elements such as images, audio, and video. These elements help in better presentation of information and improve user experience. HTML provides specific tags to include multimedia content easily on web pages. The <*img*> tag is used to display images, the <*audio*> tag is used to play sound or music, and the <*video*> tag is used to play video clips. By combining these elements, a web designer can create interactive and visually appealing web pages.

1.3.1 Multimedia Content on Web Pages

Multimedia elements such as images, audio, and video play a key role in making web pages more interactive, engaging, and visually appealing**.** They help convey information more effectively than plain text and enhance the overall user experience. HTML provides specific tags to include multimedia content on web pages:

- **<img>** : This tag is used to display images on a web page.
- **<audio>** : This tag is used to play sound or music on the web page.
- **<video>** : This tag is used to play video clips directly on the web page.

By using these tags and attributes, a web designer can combine text with multimedia to create a web page that is informative, dynamic, and visually appealing to users.

1.3.1.1 Including Image in a Web Page

Images make a web page more attractive and informative. HTML allows you to add images using the <*img*> tag. The image can be in formats like JPEG, PNG, GIF, or SVG. Attributes such as *src* (to specify the image path), alt (to provide alternative text if the image cannot be displayed), and *width/height* (to set the size of the image) help in proper placement and display.

HTML Tag: <img>

Syntax: <img src="image_path" alt="alternate_text" width="value" height="value" title="tooltip_text">

Attributes of <*img*> Tag: In Table 1.3.1 describes common attributes of <*img*> tag.

Table 1.3.1 Attributes of <*img*> Tag

Attribute	Description
src	Specifies the path of the image file.
alt	Provides alternative text if the image cannot be displayed.
width	Sets the width of the image.
height	Sets the height of the image.
title	Provides a tooltip text when hovering over the image.

Example: Write the HTML code to display an image called flower.jpg with width 300px, height 200px, an alternate text "Red Rose" with a paragraph "This is an example of adding an image to a webpage." and a heading "My Favourite Flower".

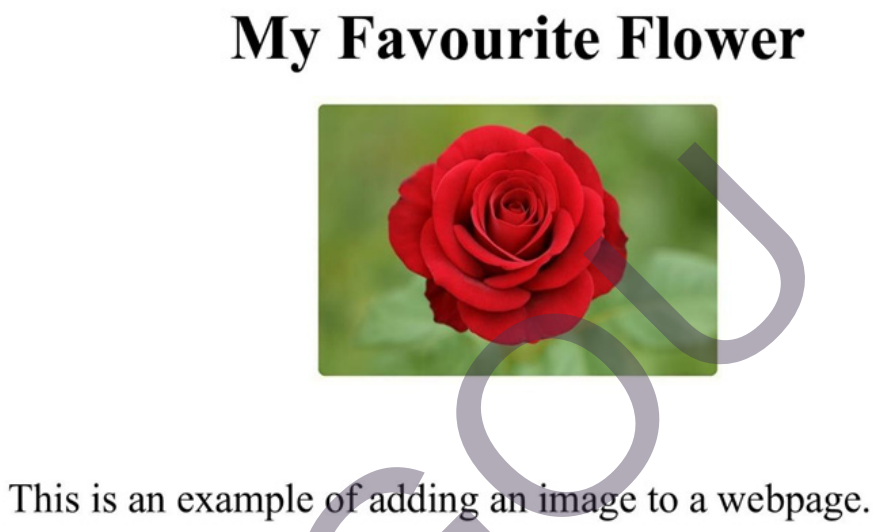
HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Image Example</title>
</head>
<body>
   <h1>My Favorite Flower</h1>
   <!-- Image tag -->
   <img src="flower.jpg" alt="Red Rose" width="300" height="200" >
   <p>This is an example of adding an image to a webpage.</p>
</body>
</html>
```

Here,

- *<img>* tag is used to include an image.
- *src="flower.jpg"* specifies the path of the image file.
- *alt="Red Rose"* provides the text that will be displayed if the image does not load.
- The *width and height* attributes set the size of the image.

Output:



Exercise Questions

Write the HTML code to display an image called mountain.jpg with width 500px, height 300px, and an alternate text “Snowy Mountain” with heading “Snowy Mountain”.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Mountain Image Example</title>
</head>
<body>
   <h1>Snowy Mountain</h1>
><img src="mountain.jpg" alt="Snowy Mountain" width="500" height="300"
</body>
</html>
```

Output:

1.3.1.2 Including Audio in a Web Page

HTML allows you to add audio (like songs, sound effects, or voice clips) using the <audio> tag.The <audio> element lets browsers play sound files directly without needing extra plugins. You can include audio files in formats like MP3, WAV, or OGG. Attributes such as controls (to show play/pause buttons), autoplay (to start automatically), and loop (to repeat continuously) allow better control over audio playback.

HTML Tag: <audio>

This tag is used to embed audio content in a webpage.

Syntax:

<audio src="audiofile.mp3" controls></audio>

or (using nested <source> elements):

<audio controls>

<source src="audiofile.mp3" type="audio/mpeg">

<source src="audiofile.ogg" type="audio/ogg">

Your browser does not support the audio element.

</audio>

Attributes of *<audio>* Tag: In Table 1.3.2 describes common attributes of *<audio>* tag.

Table 1.3.2 Attributes of <*audio*> Tag

Attribute	Description
src	Specifies the path (URL) of the audio file.
controls	Adds play, pause, and volume controls.
autoplay	Automatically starts playing when the page loads.
loop	Repeats the audio file continuously.
muted	Starts the audio in a muted state.
preload	Specifies if/how the audio should be loaded when the page loads. Options: auto, metadata, none.

Example: Write the HTML code to include basic audio player on webpage.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Audio Example 1</title>
</head>
<body>
   <h2>Play Background Music</h2>
   <audio src="music.mp3" controls></audio>
</body>
</html>
```

Output:

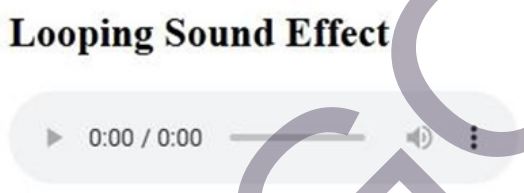
Exercise Questions

Write the HTML code to display an audio player that plays a file named sound.mp3 automatically when the webpage loads audio with autoplay and loop.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Audio Example 2</title>
</head>
<body>
   <h2>Looping Sound Effect</h2>
   <audio src="sound.mp3" controls autoplay loop></audio>
</body>
</html>
```

Output:



1.3.1.3 Including Video in a Web Page

The <video> tag in HTML is used to embed video files (like .mp4, .ogg, .webm) directly into a webpage. It allows users to play, pause, and control video playback without external plugins. Similar to audio, the controls attribute provides play/pause functionality, autoplay starts the video automatically, loop repeats the video, and width/height sets the display size.

HTML Tag: <video>

Used to embed video content in a webpage.

Syntax:

<video src="video.mp4" controls></video>

or (using nested <source> tags for multiple formats):

<video controls width="500" height="300">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">

Your browser does not support the video tag.

</video>

Attributes of <*video*> Tag: In Table 1.3.3 describes common attributes of <*video*> tag.

Table 1.3.3 Attributes of <*video*> Tag

Attribute	Description
Src	Specifies the path (URL) of the video file.
Controls	Adds play, pause, and volume controls.
Autoplay	Starts playing the video automatically when the page loads.
Loop	Repeats the video continuously.
Muted	Starts the video with no sound.
Poster	Specifies an image to show while the video is downloading or before playing.
width / height	Specifies the size of the video player.
Preload	Specifies how the video should be loaded: auto, metadata, or none.

Example: Write the HTML code to include basic video player on webpage.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Video Example 1</title>
</head>
<body>
   <h2>My Favorite Movie Clip</h2>
   <video src="clip.mp4" controls width="400" height="250"></video>
</body>
</html>
```

Output:

Exercise Questions

Write the HTML code to display an video player that plays a file named background. mp4 automatically when the webpage loads video with autoplay, loop, and poster image.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Video Example 2</title>
</head>
<body>
   <h2>Looping Background Video</h2>
  <video src="background.mp4" width="500" height="300" controls autoplay
loop muted poster="preview.jpg"></video>
</body>
</html>
```

Output :

1.3.1.4 <embed> Tag in HTML

The <embed> tag in HTML is used to embed external content such as audio, video, PDF files, Flash files, or other multimedia objects directly into a webpage. It acts as a container for external applications or interactive content.

HTML Tag: <embed>

It is an empty tag, meaning it doesn't require a closing tag.

Syntax: <embed src="filename" type="filetype" width="width" height="height">

Attributes of *<embed>* Tag: In Table 1.3.4 describes common attributes of <*embed*> tag.

Table 1.3.4 Attributes of <*embed*> Tag

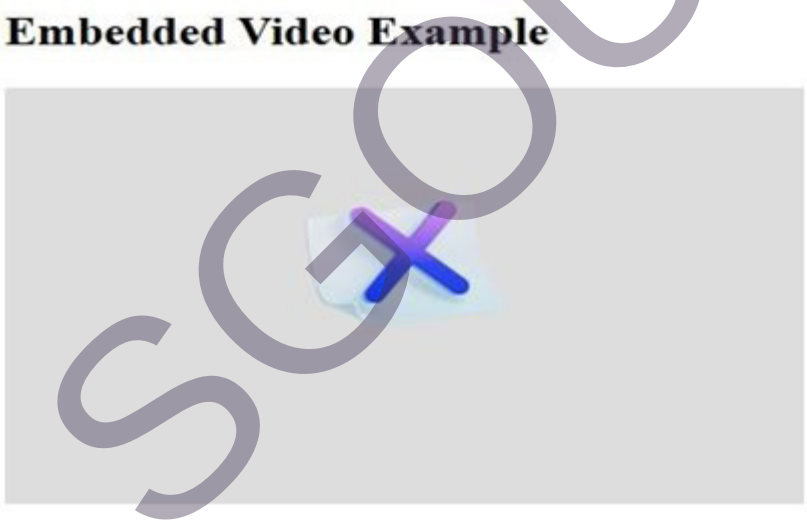
Attribute	Description
src	Specifies the path (URL) of the file to be embedded.
type	Specifies the MIME type (e.g., video/mp4, application/pdf, audio/mpeg).
width	Sets the width of the embedded content.
height	Sets the height of the embedded content.
autoplay	Automatically plays the file when the page loads (for media files).

Example: Write the HTML code to embed a video file to webpage.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Embed Example 1</title>
</head>
<body>
   <h2>Embedded Video Example</h2>
   <embed src="movie.mp4" type="video/mp4" width="400" height="250">
</body>
</html>
```

Output:



Exercise Questions

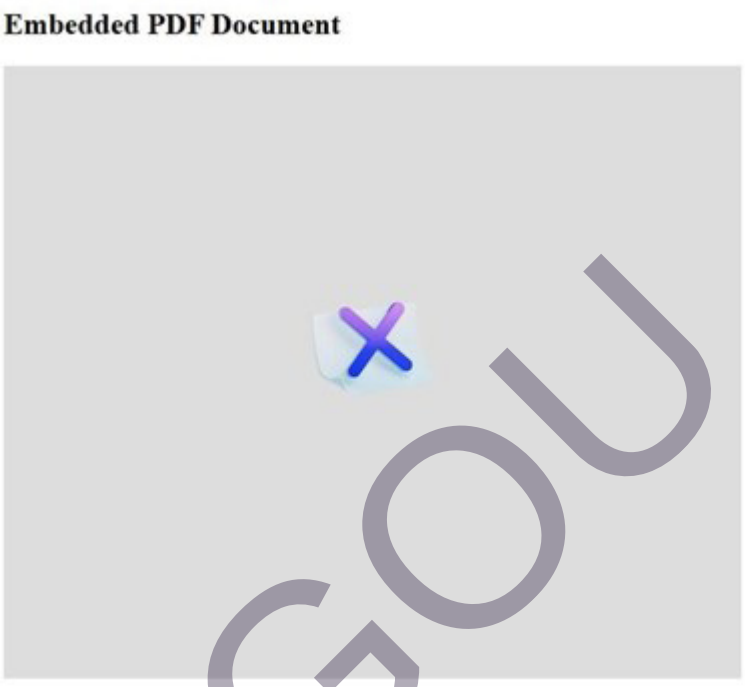
Write the HTML code to embed a PDF file to a webpage.

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Embed Example 2</title>
</head>
<body>
```

```
  <h2>Embedded PDF Document</h2>
 <embed src="document.pdf" type="application/pdf" width="600" height="500">
</body>
</html>
```

Output:



1.3.1.5 Procedure

General steps to add graphics to web page as follows:

1. Open a text editor (e.g., Notepad).
2. Write the HTML code using multimedia tags.
3. Save the file with a .html extension (e.g., media.html).
4. Open the file in a web browser.
5. Verify that the image, audio, and video appear and play correctly.

Lab Practice Questions

1. Write the HTML code to display an image called sunset.png with width 400px, height 250px, and an alternate text *"Beautiful Sunset*" with heading *"Beautiful Sunset"*.

2. Write the HTML code to display an image called beach.png with width 600px**,** height 400px**,** and an alternate text *"Sunny Beach"* with a paragraph *"how beautiful is this!"* and a heading *"Sunny Beach - My Favourite Image"*.

3. Write the HTML code to display an audio player that plays a file named *song. mp3* automatically when the webpage loads and repeats it continuously.

4. Write the HTML code to display a video called *nature.mp4* with controls, width 600px, height 350px, and a poster image *nature.jpg.*

5. Write the HTML code to embed a video file named *car moving.mp4* in a webpage with width 500px and height 300px.

Experiment No: 4

Title: Layers and Image Maps

Objectives:

- Create web pages using layers to control element positioning
- Design interactive image maps with clickable regions
- Implement CSS to style and manipulate layered content
- Develop navigation menus using image maps

Theory

1.4.1 Introduction

In web design, layers and image maps play an important role in creating visually rich and interactive web pages. Layers allow designers to position and control multiple elements independently, much like arranging transparent sheets one over another. Using layers, different components such as text, images, and buttons can be placed precisely on a page to achieve dynamic layouts. Image maps, on the other hand, transform a single image into multiple clickable regions, each linking to a different destination or triggering specific actions through JavaScript. Together, layers and image maps enable developers to design interactive interfaces, enhance user engagement, and provide seamless navigation experiences within a webpage.

1.4.2 Create web pages using layers to control element positioning

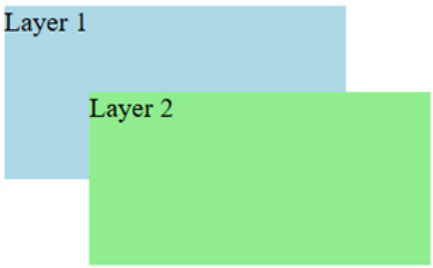
In web design, a layer refers to an HTML element that can be placed above or below other elements on a page. Using CSS positioning properties like absolute, relative, and fixed, we can control where elements appear. The z-index property allows us to manage the stacking order of these layers. By mastering layers, you can create visually rich and well-organized web pages where elements don't just sit in a flow but can overlap creatively.

Sample Program

```
<!DOCTYPE html>
<html>
<head>
<style>
#layer1 {
```

```
  position: absolute;
  top: 50px;
  left: 50px;
  width: 200px;
  height: 100px;
  background-color: lightblue;
  z-index: 1;
}
#layer2 {
  position: absolute;
  top: 100px;
  left: 100px;
  width: 200px;
  height: 100px;
  background-color: lightgreen;
  z-index: 2;
}
</style>
</head>
<body>
<div id="layer1">Layer 1</div>
<div id="layer2">Layer 2</div>
</body>
</html>
```

Expected Outcome:



Two overlapping colored boxes appear, with Layer 2 on top due to a higher z-index.

1.4.3 Design interactive image maps with clickable regions

An image map is an image with defined areas that act as clickable links. Each area can redirect users to different web pages or sections. Image maps are useful for creating interactive graphics, navigation menus, or diagrams. Using the <map> and <area> tags in HTML, we can define shapes like rectangles, circles, or polygons over an image and assign links to them, making static images interactive for users.

Sample Program

```
<!DOCTYPE html>
<html>
<body>
<img src="world-map.jpg" usemap="#worldmap" width="500" height="300">
<map name="worldmap">
 <area shape="rect" coords="34,44,150,150" href="asia.html" alt="Asia">
      <area shape="circle" coords="300,150,50" href="africa.html" alt="Africa">
       <area shape="poly" coords="400,50,450,50,450,100,400,100" href="europe.html" alt="Europe">
</map>
</body>
</html>
```

Expected Outcome:

Clicking on different regions of the image will redirect to corresponding web pages like asia.html or africa.html.

1.4.4 Implement CSS to style and manipulate layered content

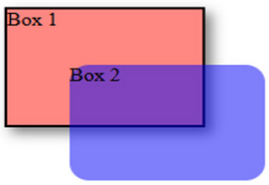
CSS allows us to style layers to make them visually attractive and distinct. We can control backgrounds, borders, shadows, opacity, and rounded corners to enhance readability and aesthetic appeal. By combining layering with CSS styling, web designers can create overlapping elements, pop-ups, floating menus, and other modern design effects that improve user experience. Styling layers also helps in organizing content without cluttering the page.

Sample Program

```
<!DOCTYPE html>
<html>
<head>
<style>
#box1 {
  position: absolute;
```

```
 top: 50px;
  left: 50px;
  width: 150px;
  height: 100px;
  background-color: rgba(255,0,0,0.5);
  border: 2px solid black;
  box-shadow: 5px 5px 10px gray;
  z-index: 1;
}
#box2 {
  position: absolute;
  top: 100px;
  left: 100px;
  width: 150px;
  height: 100px;
  background-color: rgba(0,0,255,0.5);
  border-radius: 15px;
  z-index: 2;
}
</style>
</head>
<body>
<div id="box1">Box 1</div>
<div id="box2">Box 2</div>
</body>
</html>
```

Expected Outcome:



Two semi-transparent boxes with shadows and rounded corners appear, showing styled layered content.

1.4.5 Develop navigation menus using image maps

Navigation is a critical part of any website. By using image maps, an image can be turned into a clickable menu, where different areas link to different pages. This technique allows designers to create graphical menus that are both interactive and visually appealing. Unlike text-based menus, image maps offer more creative freedom, allowing buttons, icons, or diagrams to guide users intuitively through the website.

Sample Program

```
<!DOCTYPE html>
<html>
<body>
<h2>Navigation Menu using Image Map</h2>
<img src="menu.jpg" usemap="#navmenu" width="600" height="200">
<map name="navmenu">
    <area shape="rect" coords="0,0,200,200" href="home.html" alt="Home">
    <area shape="rect" coords="200,0,400,200" href="about.html" alt="About">
    <area shape="rect" coords="400,0,600,200" href="contact.html" alt="Contact">
</map>
</body>
</html>
```

Expected Outcome:

Clicking different sections of the menu image navigates to the respective web pages, creating an interactive graphical menu.

1.4.6 Lab Practice Questions

1. Create a simple webpage that displays three layers overlapping each other using different background colors. Use CSS position and z-index properties to control their stacking order.

2. Develop an HTML page using layers where one image overlaps another, and display a caption text over the top image using absolute positioning.

3. Create an image map of your country's map with clickable regions for at least three states. Each region should link to a separate HTML page containing the state's name.

4. Develop a campus layout image map where clicking on different buildings (library, canteen, lab, etc.) takes the user to relevant pages.

5. Make an image map with mixed shapes (rectangle, circle, and polygon) to understand how different coordinates affect the clickable regions.

Experiment No: 5

Development of a Website Involving a Variety of Tools Practiced (HTML, JavaScript, XML, and CSS)

Objectives

- To apply HTML for webpage structure and content organization
- To use CSS for styling, layout design, and visual enhancement of web pages
- To integrate JavaScript or PHP for interactivity and dynamic functionality
- To demonstrate practical understanding of front-end and basic back-end web development concepts

Theory

1.5.1 Introduction

Website development involves designing and creating interactive web pages using various web technologies. HTML provides the basic structure of the webpage by defining elements like text, images, and links. CSS enhances the visual appearance by adding styles, colors, and layouts for a consistent and attractive design. JavaScript introduces interactivity and dynamic behavior such as form validation and responsive menus. XML is used for organizing and managing structured data within the website. Multimedia elements, hyperlinks, and navigation menus improve user experience and accessibility. A well-designed website should be responsive, user-friendly, and visually consistent across all devices.

Software / Tools Required:

1. Text Editor (e.g., Visual Studio Code / Notepad++)
2. Web Browser (e.g., Google Chrome / Mozilla Firefox)
3. Basic knowledge of HTML, CSS, JavaScript, and XML

By combining these technologies, developers can create responsive, interactive, and data-driven websites.

Algorithm / Procedure:

1. Create a project folder named MyWebsite.
2. Inside the folder, create the following files:

- index.html (for main structure)
- style.css (for styling)
- script.js (for JavaScript functions)
- data.xml (for storing sample data)

3. In index.html, write the HTML structure including header, navigation menu, content section, and footer.
4. Link style.css in the HTML file using the <link> tag.
5. Write CSS rules in style.css to style the web page elements.
6. Write JavaScript code in script.js to handle user interactions (e.g., button click or form validation).
7. Create an data.xml file to define and store sample data (e.g., product or user information).
8. Use JavaScript (via XMLHttpRequest or fetch API) to load and display XML data dynamically in the webpage.
9. Save all files and open index.html in a browser to test the website.

Program / Code Example:

1.5.2 Sample Program

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>My Sample Website</title>
  <link rel="stylesheet" type="text/css" href="style.css">
  <script src="script.js"></script>
</head>
<body>
  <header>
    <h1>Welcome to My Website</h1>
    <nav>
      <a href="#">Home</a>
```

```
 <a href="#">About</a>
    <a href="#">Contact</a>
  </nav>
 </header>
 <section id="content">
  <h2>Product List</h2>
  <button onclick="loadXML()">Load Data</button>
  <div id="output"></div>
 </section>
 <footer>
  <p>© 2025 My Website. All rights reserved.</p>
 </footer>
</body>
</html>
```

style.css

```
body {
  font-family: Arial, sans-serif;
  background-color: #f5f5f5;
  margin: 0;
  padding: 0;
}
header {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 15px;
}
nav a {
  color: white;
```

```
 margin: 0 10px;
  text-decoration: none;
}
section {
  padding: 20px;
  text-align: center;
}
button {
  padding: 10px 15px;
  margin-top: 10px;
  cursor: pointer;
}
```

script.js

```
function loadXML() {
  let xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    let xmlDoc = this.responseXML;
    let items = xmlDoc.getElementsByTagName("product");
    let output = "<ul>";
    for (let i = 0; i < items.length; i++) {
       let name = items[i].getElementsByTagName("name")[0].childNodes[0].
nodeValue;
        let price = items[i].getElementsByTagName("price")[0].childNodes[0].
nodeValue;
      output += "<li>" + name + " - $" + price + "</li>";
    }
    output += "</ul>";
    document.getElementById("output").innerHTML = output;
  };
  xhttp.open("GET", "data.xml", true);
  xhttp.send();
}
```

data.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
  <product>
    <name>Keyboard</name>
    <price>1200</price>
  </product>
  <product>
    <name>Mouse</name>
    <price>600</price>
  </product>
  <product>
    <name>Monitor</name>
    <price>8000</price>
  </product>
</products>
```

Note : A functional and styled web page that displays product data dynamically fetched from an XML file using JavaScript.

Result:

A website was successfully developed using HTML for structure, CSS for styling, JavaScript for interactivity, and XML for data management.

1.5.3 Lab Practice Questions

1. Create a webpage using HTML and CSS that display a homepage for a small business. (Use CSS to style the background, text color, fonts and layouts).

2. Develop a web page that display product details from an XML file and use javascript to load and display data.

3. Create a web page that loads employee details (Name, position, department) from an XML file and display them on the screen using javascript.

Experiment No: 6

Title: Cascading Style sheets

Objectives:

- understand the purpose and importance of CSS.
- differentiate between inline, internal, and external CSS.
- apply CSS properties to format text, backgrounds, borders, and layouts.
- use selectors effectively to style specific HTML elements.
- develop structured and visually appealing web pages.

Theory

1.6.1 Introduction

Cascading Style Sheets (CSS) is a style sheet language used to describe how HTML elements are displayed on the screen, paper, or other media. It separates content (HTML) from presentation (design), allowing web developers to maintain a consistent look and feel across multiple pages. CSS controls aspects such as colors, fonts, layout, spacing, and responsiveness of a webpage. The term "Cascading" refers to the order of priority , if multiple styles apply to the same element, the one defined last or with higher specificity will take effect. Without CSS, every HTML page must define its own layout and style, making web design repetitive and time-consuming. CSS allows web designers to:

- Maintain consistency across pages
- Reuse styles in multiple documents
- Improve website performance
- Enhance user experience with better visuals and responsive design

1.6.2 CSS Syntax

A CSS rule consists of a selector and a declaration block.

selector {
property: value;
}
	Example:
	h1 {
	color: blue;
	font-size: 24px;
	}
	♦ h1 is the selector (applies to all <h1> tags)
	♦ color and font-size are properties
	♦ blue and 24px are values

1.6.3 Types of CSS

Inline CSS Defined directly within an HTML tag using the style attribute.	This is inline CSS

Internal (Embedded) CSS
Defined inside the <style> tag within the <head> section of an HTML document.
	<head>
	<style>
	p {
	color: green;
	font-size: 20px;
	}
	</style>
	</head>

External CSS Defined in a separate .css file and linked using the <link> tag.	<link rel="stylesheet" type="text/css" href="style.css">

1.6.4 CSS selectors

Selector Type	Example	Description
Universal Selector	* { margin:0; }	Selects all elements
Element Selector	p { color:blue; }	Selects all elements
ID Selector	#header {background:gray}	Selects element with specific ID
Class Selector	.title {font-size:20px;}	Selects elements with a given class

Group Selector	h1, h2, h3 { color:navy; }	Groups multiple elements
Descendant Selector	Div p { color:green; }	Styles <p> inside

1.6.5 CSS Properties

Properties	Examples

Text Formatting	p {
	color: darkblue;
	font-family: Arial;
	font-size: 18px;
	text-align: justify;
	}

Backgrounds	body {
	background-color: lightyellow;
	background-image: url('pattern.jpg');
	background-repeat: no-repeat;
	background-size: cover;
	}

Borders and Padding	div {
	border: 2px solid black;
	padding: 10px;
	margin: 20px;
	}

Links Styling	a:link { color: blue; } a:hover { color: red; text-decoration: underline; } a:visited { color: purple; }

1.6.6 CSS Box Model

Example	Output

<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 250px;
  background-color: lightblue;
  padding: 20px;
  border: 5px solid blue;
  margin: 30px;
}
</style>
</head>
<body>
 CSS Box Model Example
 This box has content, padding, border, and margin.
</body>
</html>
	CSS Box Model Example
	
	This box has content, padding, border, and margin.

1.6.7 CSS Colors and Backgrounds

Example	Output

<!DOCTYPE html>
<html>
<head>
<style>
body {
background-color: lightyellow;
}
 	
h1 {
color: darkgreen;
background-color: lightgray;
text-align: center;
}
 	
p {
background-image: url('https://www.w3schools.com/css/img_lights.jpg');
background-size: cover;
color: white;
padding: 15px;
}
</style>
</head>
<body>
 	
 CSS Colors and Backgrounds
 This paragraph uses a background image and white text color.
 	
</body>
</html>
	

Sample questions

1. Write an HTML and CSS program to create a simple web page that changes the background color of the page, sets the heading color to white, and styles the paragraph text with red color, Verdana font, and larger size.

```
<!DOCTYPE html>
<html>
<head>
<title>Simple CSS Example</title>
<style>
/* Set background color for the whole page */
body {
  background-color: lightblue;
}
/* Style the heading */
h1 {
  color: white;            /* Heading text color */
  text-align: center;      /* Center align the heading */
}
/* Style the paragraph */
p {
  font-family: Verdana;    /* Font type */
  font-size: 20px;         /* Font size */
  color: red;              /* Text color */
}
</style>
</head>
<body>
<h1>My First CSS Example</h1>
<p>This is a paragraph demonstrating simple CSS styling.</p>
</body>
</html>
```

Output

2. Write an HTML program to create a paragraph and style it using inline CSS (change text color, font size, and font family).

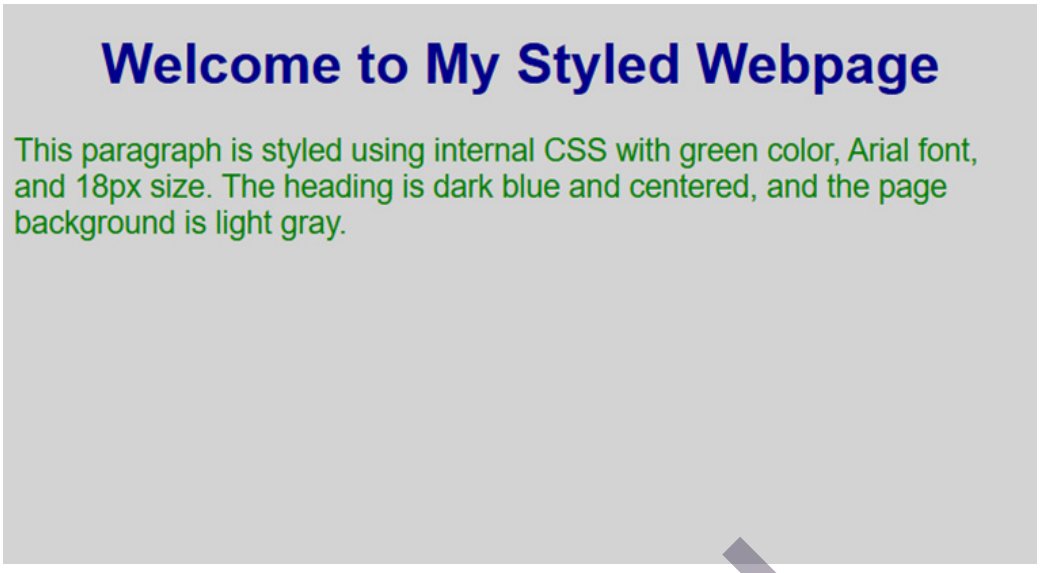
```
<!DOCTYPE html>
<html>
<head>
<title>Inline CSS Example</title>
</head>
<body>
<h2>Inline CSS Demo</h2>
<p style="color: blue; font-size: 18px; font-family: Arial;">
This paragraph is styled using <b>inline CSS</b> inside the HTML tag.
</p>
</body>
</html>
```

Output

3. Write an HTML program to style a webpage using internal CSS: Set the page background to light gray. Style the heading with dark blue color and center alignment. Style the paragraph text with green color, Arial font, and 18px size

```
<!DOCTYPE html>
<html>
<head>
<title>Internal CSS Example</title>
<style>
/* Set background color for the whole page */
body {
  background-color: lightgray;
  font-family: Arial, sans-serif;
}
/* Style the heading */
h1 {
  color: darkblue;        /* Dark blue text */
  text-align: center;     /* Center align the heading */
}
/* Style the paragraph */
p {
  color: green;           /* Paragraph text color */
  font-size: 18px;        /* Font size */
  font-family: Arial;     /* Font type */
}
</style>
</head>
<body>
<h1>Welcome to My Styled Webpage</h1>

<p>This paragraph is styled using internal CSS with green color, Arial font, and 18px size.

The heading is dark blue and centered, and the page background is light gray.</p>
</body>
</html>
```

Output



4. Create a webpage that links to an external CSS file. Set the page background color to light blue. Style the heading with dark red color and center it. Style paragraph text with green color, Verdana font, and 18px size

index.html

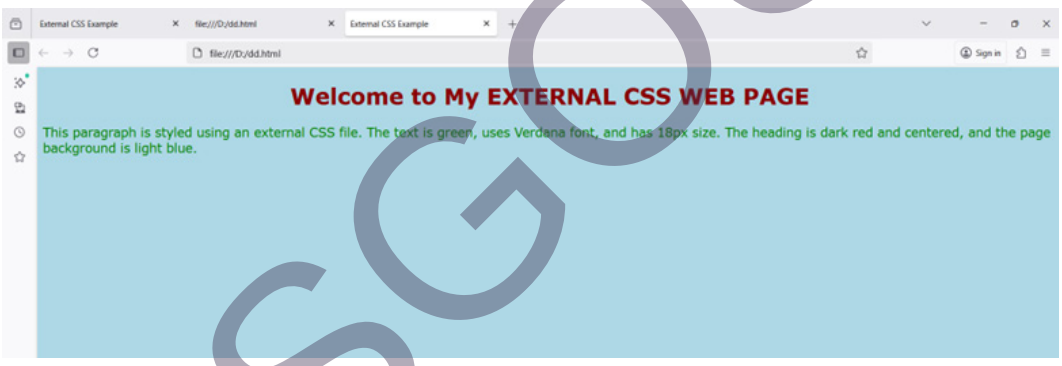
```
<!DOCTYPE html>
<html>
<head>
<title>External CSS Example</title>
<!-- Link to external CSS file -->
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<h1>Welcome to My Webpage</h1>
<p>This paragraph is styled using an external CSS file. The text is green, uses Verdana font, and has 18px size. The heading is dark red and centered, and the page background is light blue.</p>
</body>
</html>
```

style.css

```
/* Set page background color */
body {
    background-color: lightblue;
```

```
    font-family: Verdana, sans-serif;
}
/* Style heading */
h1 {
    color: darkred;
    text-align: center;
}
/* Style paragraph */
p {
    color: green;
    font-size: 18px;
    font-family: Verdana, sans-serif;
}
```

Output

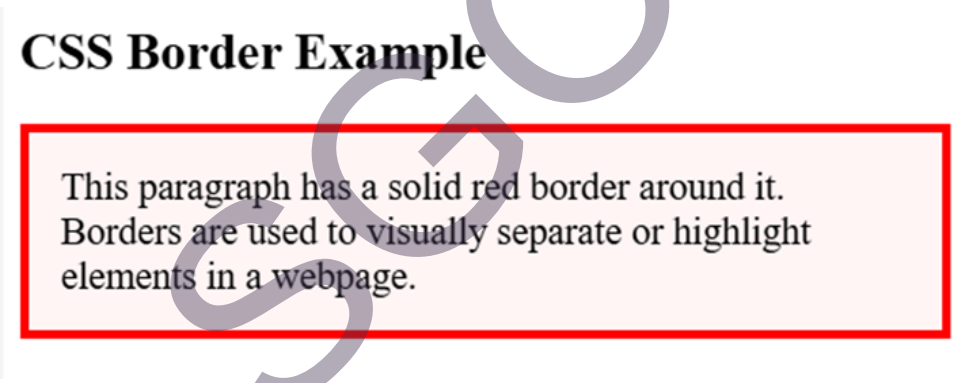


5. Write an HTML and CSS program to display a paragraph with a solid red border around it.

```
<!DOCTYPE html>
<html>
<head>
<title>Solid Red Border Example</title>
<style>
p {
  border-style: solid;     /* Defines the border style */
  border-width: 4px;       /* Sets the thickness of the border */
  border-color: red;       /* Sets the border color */
```

```
  padding: 15px;          /* Adds space between text and border */
  width: 400px;           /* Sets the width of the paragraph box */
  font-size: 18px;        /* Increases the text size */
  background-color: #fff5f5;  /* Light background to highlight the border */
}
</style>
</head>
<body>
<h2>CSS Border Example</h2>
<p>This paragraph has a solid red border around it.
Borders are used to visually separate or highlight elements in a webpage.</p>
</body>
</html>
```

Output



6. Write an HTML program to display different types of borders (dotted, dashed, solid, double, groove, ridge, inset, outset, none, hidden, and mixed) using CSS.

```
<!DOCTYPE html>
<html>
<head>
<title>Border Styles Example</title>
<style>
p.dotted {
  border-style: dotted;
```

```
  border-color: red;
  border-width: 3px;
  padding: 10px;
}
p.dashed {
  border-style: dashed;
  border-color: green;
  border-width: 3px;
  padding: 10px;
}
p.solid {
  border-style: solid;
  border-color: blue;
  border-width: 3px;
  padding: 10px;
}
p.double {
  border-style: double;
  border-color: purple;
  border-width: 4px;
  padding: 10px;
}
p.groove {
  border-style: groove;
  border-color: orange;
  border-width: 5px;
  padding: 10px;
}
```

```
p.ridge {
  border-style: ridge;
  border-color: brown;
  border-width: 5px;
  padding: 10px;
}
p.inset {
  border-style: inset;
  border-color: darkcyan;
  border-width: 4px;
  padding: 10px;
}
p.outset {
  border-style: outset;
  border-color: darkmagenta;
  border-width: 4px;
  padding: 10px;
}
p.none {
  border-style: none;
  color: gray;
  padding: 10px;
}
p.hidden {
  border-style: hidden;
  color: gray;
  padding: 10px;
}
```

```
p.mix {
  border-style: dotted dashed solid double;
  border-color: red green blue purple;
  border-width: 4px;
  padding: 10px;
}
body {
  font-family: Arial;
  background-color: #f9f9f9;
}
</style>
</head>
<body>
<h2 style="text-align:center;">CSS Border Styles Demonstration</h2>
<p>This example shows how to use the <b>border-style</b> property to display different types of borders around text.</p>
<p class="dotted">This is a <b>dotted</b> border — often used for highlighting notes.</p>
<p class="dashed">This is a <b>dashed</b> border — suitable for separating content sections.</p>
<p class="solid">This is a <b>solid</b> border — commonly used for boxes and containers.</p>
<p class="double">This is a <b>double</b> border — gives a formal appearance.</p>
<p class="groove">This is a <b>groove</b> border — appears as if carved into the page.</p>
<p class="ridge">This is a <b>ridge</b> border — appears raised from the page.</p>
<p class="inset">This is an <b>inset</b> border — makes the content look pressed in.</p>
```

```
<p class="outset">This is an <b>outset</b> border — makes the content look raised up.</p>

<p class="none">This is with <b>no border</b> applied.</p>

<p class="hidden">This is a <b>hidden border</b> (similar to none).</p>

<p class="mix">This is a <b>mixed</b> border — combines multiple border styles in one element.</p>

</body>

</html>
```

Output

7. Write an HTML and CSS program to display an image that rotates when the user hovers the mouse over it.

Index.html

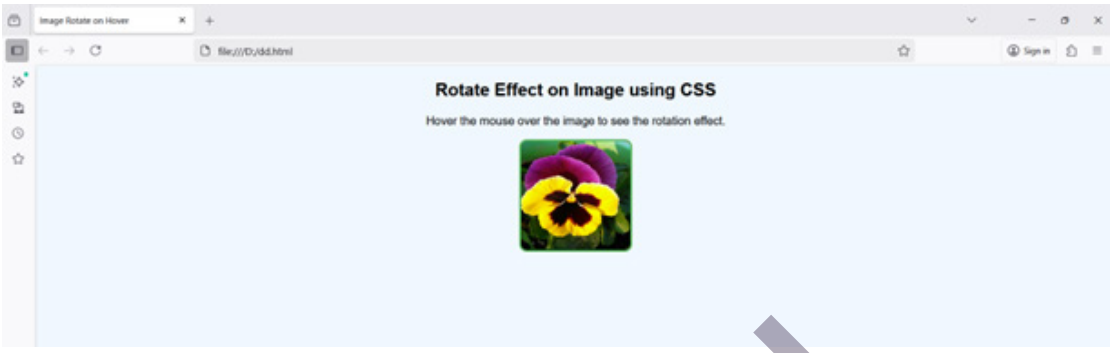
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title>Image Rotate on Hover</title>
<!-- Linking external CSS file -->
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<h2>Rotate Effect on Image using CSS</h2>
<p>Hover the mouse over the image to see the rotation effect.</p>
<!-- Image with hover rotate class -->
<figure class="hover-rotate">
        <img src="https://www.w3resource.com/html-css-exercise/html-css-practical-exercises/flower-2.jpeg" height="150" width="150" alt="Flower Image">
</figure>
</body>
</html>
```

Style.css

```
body {
  background-color: #f0f8ff;
  text-align: center;
  font-family: Arial, sans-serif;
}
.hover-rotate img {
  transition: transform 1s ease;

  border: 3px solid #4CAF50;
  border-radius: 10px;
}
```

```
.hover-rotate img:hover {
 transform: rotate(360deg);
}
```

Output

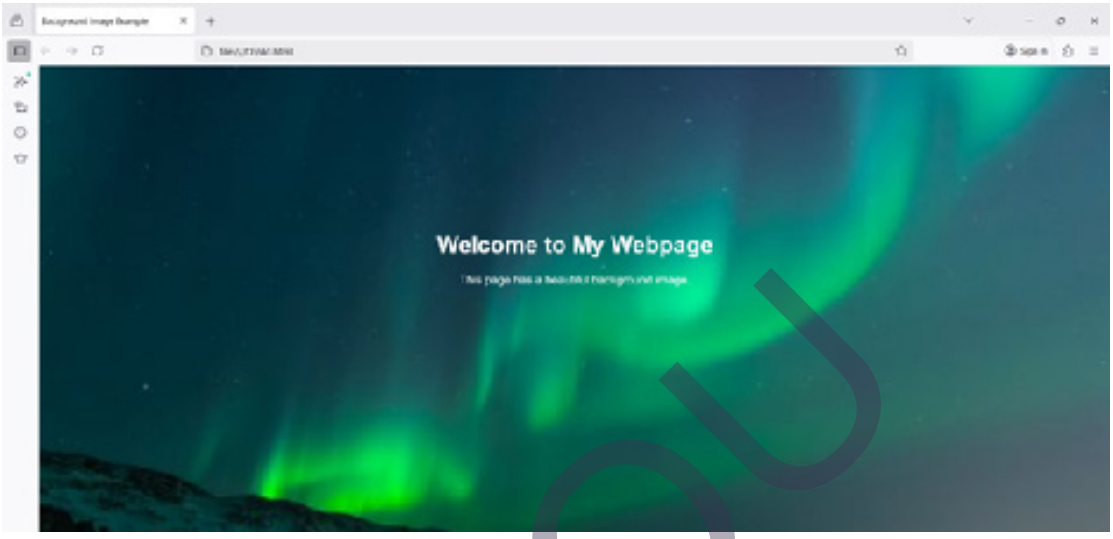


8. Write an HTML and CSS program to set an image as the background of a webpage.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title>Background Image Example</title>
<style>
body {
  background-image: url('https://www.w3schools.com/css/img_lights.jpg');
  background-size: cover;      /* Makes the image cover the entire page */
  background-repeat: no-repeat; /* Prevents image repetition */
  background-attachment: fixed; /* Keeps the image fixed when scrolling */
  font-family: Arial, sans-serif;
  color: white;
  text-align: center;
  padding-top: 200px;
}
</style>
</head>
<body>
```

```
<h1>Welcome to My Webpage</h1>
<p>This page has a beautiful background image.</p>
</body>
</html>
```

Output



Lab Practice Questions

1. Write an HTML and CSS program to display a paragraph in italic style and font size 20px.

2. Write an HTML and CSS program to style the page with cream background, heading text centered and dark orange, and paragraph text purple, italic, and font size 20px.

3. Write an HTML and CSS program to display a heading in blue color and paragraph text in green color using internal CSS.

4. Write an HTML and CSS program to set the background color of the page to light yellow.

5. Write an HTML and CSS program to create a webpage with a decorative page border. Use CSS properties to set the style, width, and color of the border around the entire webpage.

6. Create a web page using internal CSS where: Body background is light yellow., Heading is dark red, center-aligned, and bold., Paragraph text is blue, italic, and 20px font size.

7. Design a webpage using external CSS to: Apply sky blue background., Style heading with orange color, centered, and bold., Paragraph text in red, italic, and 18px size.

8. Write an HTML and CSS program to display an image that enlarges (zooms in) when the user hovers the mouse over it.

9. Write an HTML and CSS program to display a background image aligned to the bottom-right corner of the webpage.

10. Write an HTML and CSS program to display a heading in “Courier New” font and a paragraph in “Times New Roman” with 20px font size.

Experiment No: 7

HTML forms and Fields

Objectives

- To understand the structure and purpose of HTML forms
- To learn various attributes of the <form> tag
- To identify and use different types of form controls such as text boxes, radio buttons, checkboxes, dropdown lists, and buttons
- To apply fieldsets and legends for grouping related form elements
- To perform basic client-side validation using HTML5 attributes
- To understand form submission and data handling

Theory

1.7.1 Fundamentals of HTML Forms and Fields

HTML forms are used to collect user input on web pages. A form allows users to enter data that can be sent to a server for processing, such as login details, feedback, or registration information. Forms are created using the <form> tag, which acts as a container for various input elements.

1.7.1.1 Attributes of the <form> Tag

The <form> tag supports several attributes that define its behavior:

- **action:** Specifies where to send the form data when submitted.
- **method:** Defines how to send the data — GET (URL parameters) or POST (request body).
- **target:** Determines where to display the response after submitting (e.g., _ blank, _self).
- **enctype:** Specifies how the form data should be encoded (used when uploading files).
- **autocomplete:** Enables or disables browser autofill suggestions.

Common form control elements, description and its example shown in Table 7.1.1.

Table 7.1.1 Common Form Controls

Form Element	Description	Example
<input type="text">	Creates a single-line text box for user input.	Name: <input type="text" name="username">
<input type="password">	Creates a field that hides characters entered by the user (for passwords).	Password: <input type="password" name="userpass">
<input type="radio">	Allows the user to select one option from a group of choices.	<input type="radio" name="gender" value="male"> Male
<input type="checkbox">	Allows the user to select multiple options from available choices.	<input type="checkbox" name="hobby" value="Music"> Music
<select> and <option>	Creates a drop-down menu from which the user can choose one option.	<select><option>CSE</option> <option>ECE</option></select>
<textarea>	Provides a multi-line area for users to enter text (like comments or messages).	<textarea rows="4" cols="30"></textarea>
<input type="submit">, <input type="reset">, <button>	Used to submit, reset, or perform actions within a form.	<input type="submit" value="Submit">

1.7.1.2 Fieldsets and Legends

In HTML forms, it is often necessary to organize related form controls into logical groups to make the form more readable and user-friendly. The <fieldset> and <legend> elements help achieve this.

The <fieldset> tag is used to group related elements within a form by drawing a box around them. This visual separation helps users understand which fields belong together, such as “Personal Details,” “Contact Information,” or “Login Information.” Grouping related fields improves form structure, readability, and accessibility, especially for long or complex forms.

The <legend> tag is used to provide a caption or title for the grouped fields defined by the <fieldset>. The legend usually appears at the top of the fieldset box and briefly describes the purpose or category of the enclosed form controls.

1.7.1.3 Form Validation (Client-Side)

Form validation is an important feature in web development that ensures users enter correct and complete data before submitting a form. Client-side validation is performed by the web browser, without the need to send data to the server first. This helps in reducing errors, improving user experience, and saving time.

HTML5 provides several built-in validation attributes that can be applied directly to form elements. These attributes make it easier to check for missing or incorrect information before the form is submitted. When a validation rule is not satisfied, the browser automatically displays an error message and prevents form submission until the issue is corrected.

Some commonly used HTML5 validation attributes are:

- **required** : Ensures that the field must be filled before the form is submitted. If a required field is left blank, the browser prompts the user to complete it.
- **pattern** : Specifies a regular expression that defines the format or pattern that the input must follow. It is often used for validating inputs like phone numbers, postal codes, or passwords.
- **min, max, maxlength, minlength** : Used to restrict the range or length of input values. For example, min and max are commonly used with numeric fields, while maxlength and minlength are used for text inputs.
- **type** : Automatically validates data formats based on the input type. For example, type="email" checks for a valid email format, type="url" checks for a proper web address, and type="number" ensures only numeric values are entered.

A. Form Accessibility and Usability

Making forms easy to use and accessible helps all users fill them correctly. Using <label> tags linked to input fields tells the user what each field is for, which is very helpful for people using screen readers. Adding placeholders or small instructions inside the fields shows what type of information is expected. Grouping related fields together using <fieldset> makes the form easier to read and understand. Keeping a logical tab order lets users move through the form smoothly using the keyboard. These steps make forms clear, simple, and easy for everyone to use.

B. Form Submission and Handling

When a user submits a form, the browser collects all the information entered in the form and sends it to the server specified in the action attribute. If the form uses method="get", the data is added to the URL, which can be seen in the browser address bar. If the form uses method="post", the data is sent in the request body, which is more secure and better for sending sensitive information like passwords. Form submission allows the server to process the data, such as saving it to a database or sending a response back to the user.

Exercise Questions

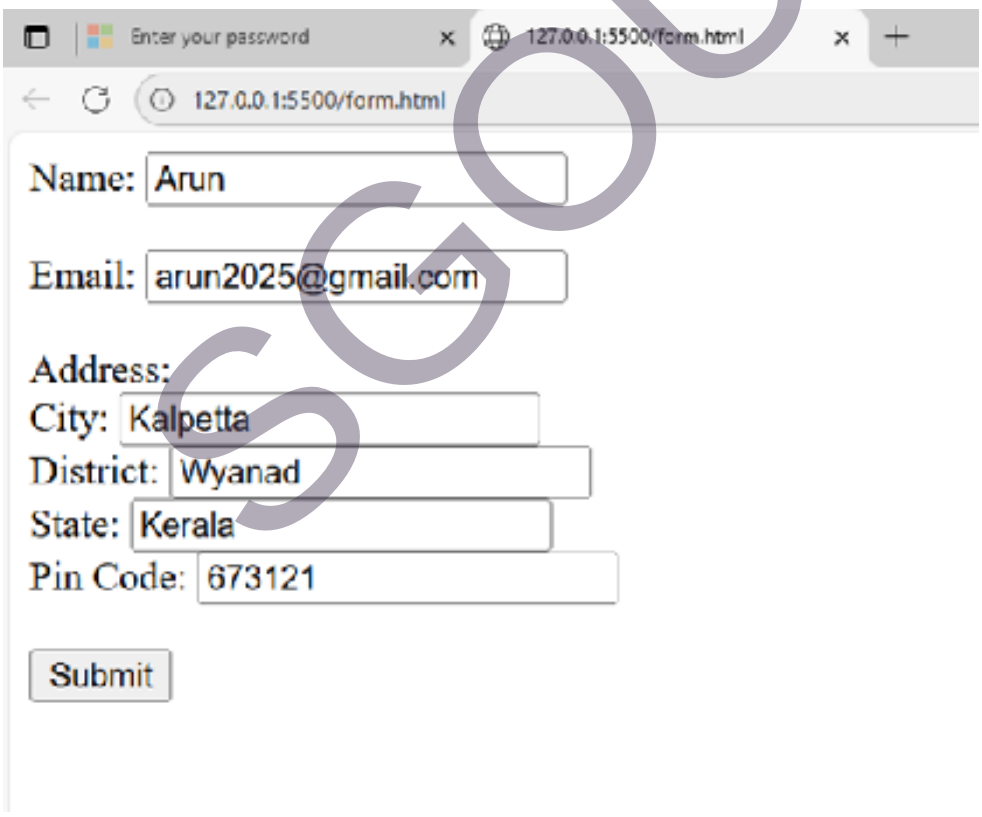
Question 1:

Create an HTML form to collect a user's details including name, email address, and complete address (city, district, state, and zip code). Include a Submit button.

Program:

```
<form>
  Name: <input type="text" name="username" required><br><br>
  Email: <input type="email" name="useremail" required><br><br>
   Address:<br>
  City: <input type="text" name="city" required><br>
  District: <input type="text" name="district" required><br>
  State: <input type="text" name="state" required><br>
  Pin Code: <input type="text" name="pincode" required><br><br>
   <input type="submit" value="Submit">
</form>
```

Output



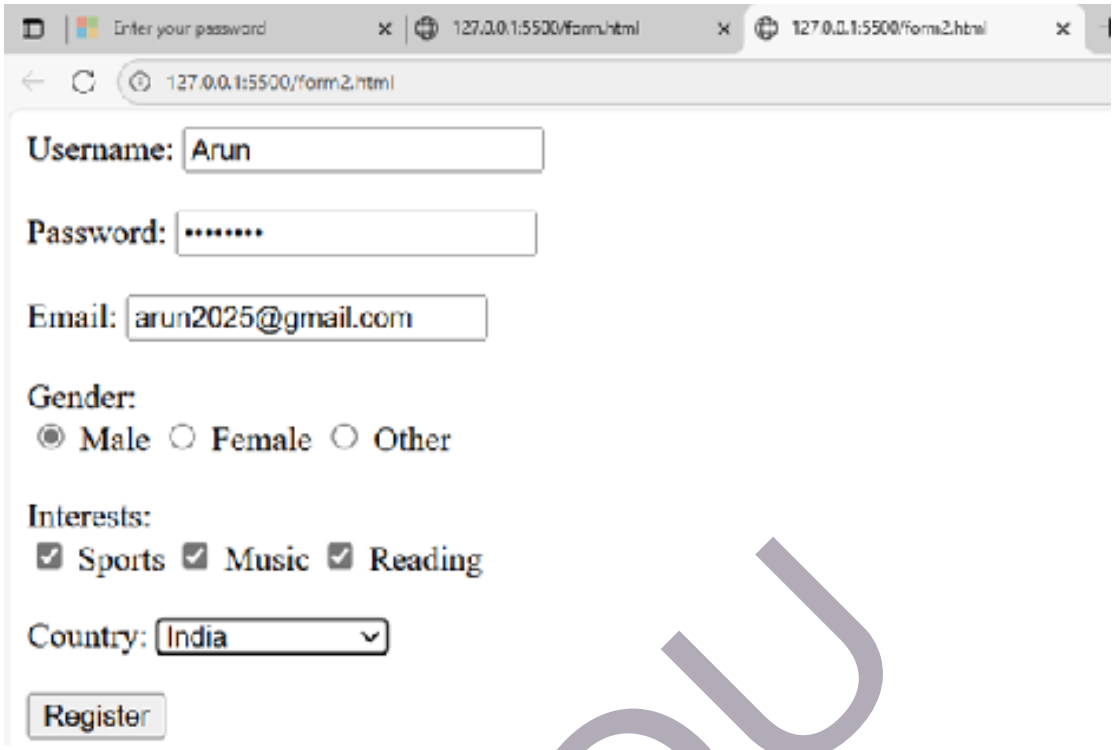
Question 2:

Design an HTML registration form that uses various input types including text, password, email, radio buttons, checkboxes, and a dropdown menu to collect a user's details such as username, password, email, gender, interests, and country. Include a Submit button.

Program:

```
<form>
  Username: <input type="text" name="username" required><br><br>
  Password: <input type="password" name="password" required><br><br>
  Email: <input type="email" name="email" required><br><br>
   Gender:<br>
  <input type="radio" name="gender" value="male" required> Male
  <input type="radio" name="gender" value="female" required> Female
 <input type="radio" name="gender" value="other" required> Other<br><br>
   Interests:<br>
  <input type="checkbox" name="interest" value="sports"> Sports
  <input type="checkbox" name="interest" value="music"> Music
 <input type="checkbox" name="interest" value="reading">Reading<br><br>
  Country:
  <select name="country" required>
    <option value="">Select Country</option>
    <option value="india">India</option>
    <option value="usa">USA</option>
    <option value="uk">UK</option>
    <option value="australia">Australia</option>
  </select><br><br>
   <input type="submit" value="Register">
</form>
```

Output:



Question 3:

Create an HTML registration form that demonstrates the use of <fieldset> and <legend> to group related form elements. The form should include:

- Personal Details: Name, Email, Date of Birth, Gender (radio buttons)
- Account Details: Username, Password
- Hobbies: Multiple selections using checkboxes
- Location: Country selection using a dropdown menu
- A Submit button

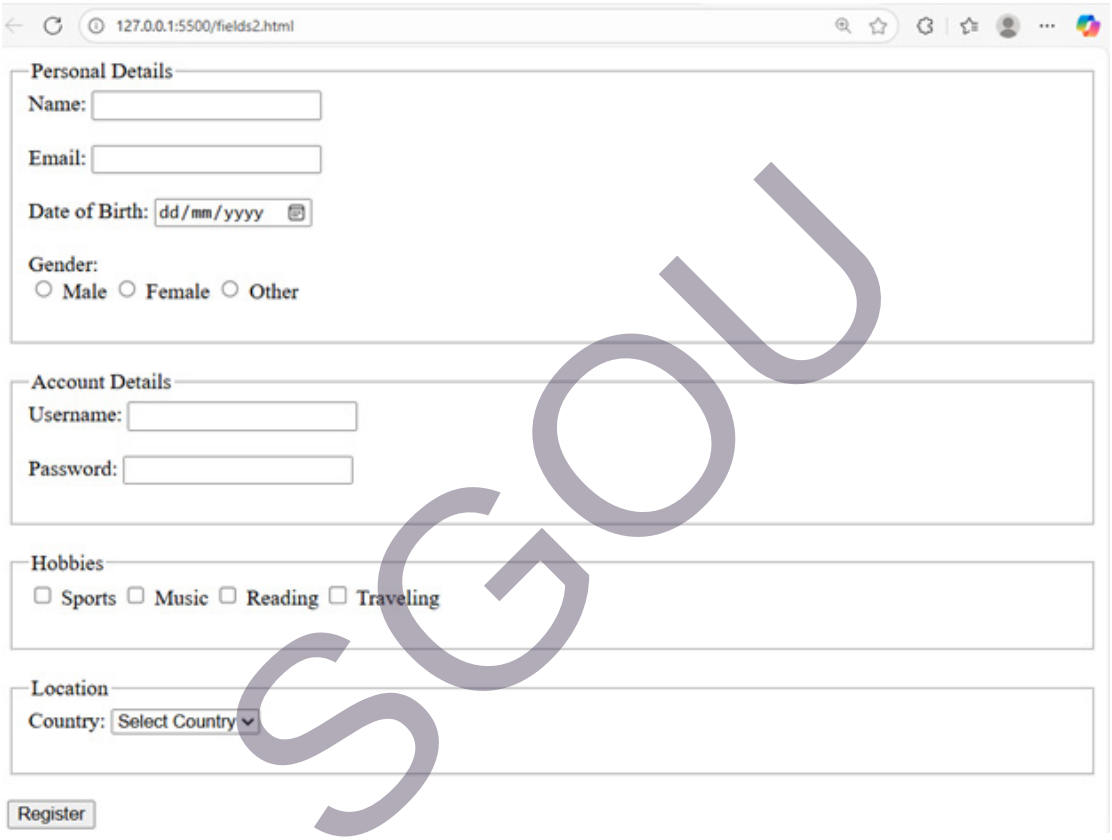
Program:

```
<form>
  <fieldset>
    <legend>Personal Details</legend>
    Name: <input type="text" name="name" required><br><br>
    Email: <input type="email" name="email" required><br><br>
    Date of Birth: <input type="date" name="dob" required><br><br>
    Gender:<br>
```

```
  <input type="radio" name="gender" value="male" required> Male
  <input type="radio" name="gender" value="female" required> Female
<input type="radio" name="gender" value="other" required> Other<br><br>
</fieldset>
<br>
 <fieldset>
  <legend>Account Details</legend>
  Username: <input type="text" name="username" required><br><br>
  Password: <input type="password" name="password" required><br><br>
</fieldset>
<br>
   <fieldset>
  <legend>Hobbies</legend>
  <input type="checkbox" name="hobby" value="sports"> Sports
  <input type="checkbox" name="hobby" value="music"> Music
  <input type="checkbox" name="hobby" value="reading"> Reading
            <input type="checkbox" name="hobby" value="traveling">
Traveling<br><br>
 </fieldset>
 <br>
  <fieldset>
  <legend>Location</legend>
  Country:
  <select name="country" required>
    <option value="">Select Country</option>
    <option value="india">India</option>
    <option value="usa">USA</option>
    <option value="uk">UK</option>
    <option value="australia">Australia</option>
```

```
    </select><br><br>
  </fieldset>
  <br>
   <input type="submit" value="Register">
</form>
```

Output:



Question 4:

Create an HTML registration form that demonstrates client-side validation using the required attribute and pattern checks. The form should include:

- Personal details: Name, Email, Phone Number, Date of Birth
- Account details: Username, Password (with pattern for strong password)
- Gender selection using radio buttons
- Interests selection using checkboxes
- Country selection using a dropdown menu
- A Submit button

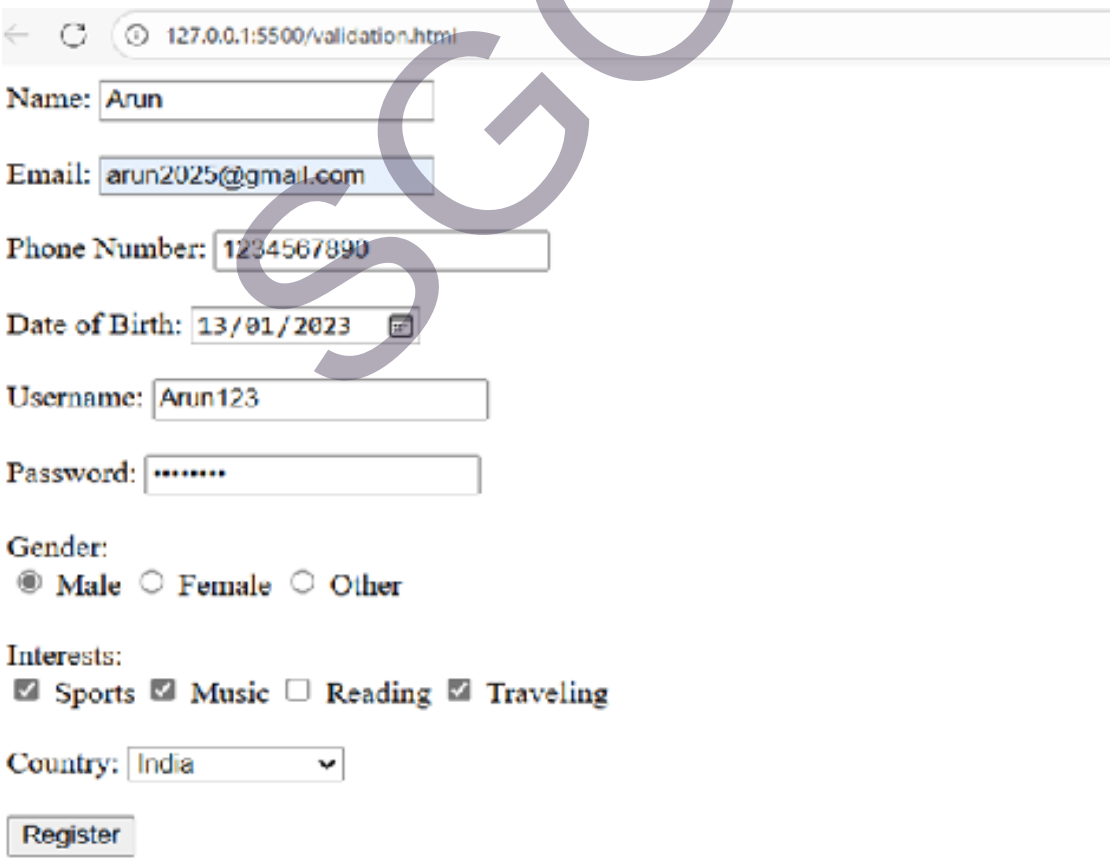
Ensure that all fields are validated on the client-side before submission, using required fields, input types, and pattern attributes.

Program:

```
<form>

  Name: <input type="text" name="name" required placeholder="Enter your full name"><br><br>

   Email: <input type="email" name="email" required placeholder="Enter your email"><br><br>

  Phone Number:

 <input type="tel" name="phone" required pattern="[0-9]{10}" title="Enter a 10-digit phone number" placeholder="1234567890"><br><br>

  Date of Birth: <input type="date" name="dob" required><br><br>

 Username:<input type="text" name="username" required placeholder="Choose a username"><br><br>

  Password:

 <input type="password" name="password" required

       pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"

        title="Must contain at least one number, one uppercase, one lowercase letter, and minimum 6 characters"

       placeholder="Enter a strong password"><br><br>

  Gender:<br>

 <input type="radio" name="gender" value="male" required> Male

 <input type="radio" name="gender" value="female" required> Female

<input type="radio" name="gender" value="other" required> Other<br><br>

   Interests:<br>

 <input type="checkbox" name="interest" value="sports"> Sports

 <input type="checkbox" name="interest" value="music"> Music
```

```
  <input type="checkbox" name="interest" value="reading"> Reading

        <input type="checkbox" name="interest" value="traveling">
Traveling<br><br>

  Country:

 <select name="country" required>

   <option value="">Select Country</option>

   <option value="india">India</option>

   <option value="usa">USA</option>

   <option value="uk">UK</option>

   <option value="australia">Australia</option>

 </select><br><br>

   <input type="submit" value="Register">

</form>
```

Output:



Question 5:

Create an HTML registration form to collect the following details:

- Name
- Email
- Password
- Gender
- Country

The form should submit the data to welcome.html using the POST method. Include a Submit button.

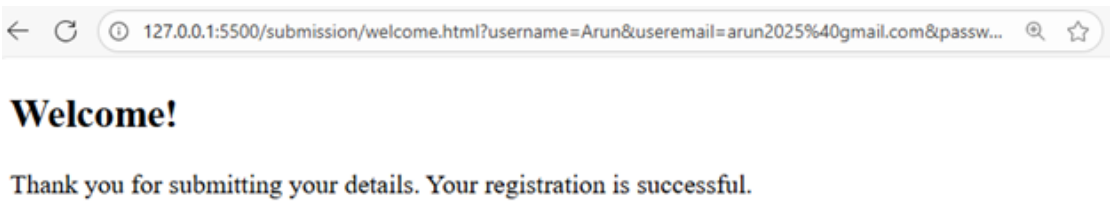
Program:

```
<form action="welcome.html" method="post">
  Name: <input type="text" name="username" required><br><br>
  Email: <input type="email" name="useremail" required><br><br>
  Password: <input type="password" name="password" required><br><br>
   Gender:<br>
  <input type="radio" name="gender" value="male" required> Male
  <input type="radio" name="gender" value="female" required> Female
 <input type="radio" name="gender" value="other" required> Other<br><br>
   Country:
  <select name="country" required>
    <option value="">Select Country</option>
    <option value="india">India</option>
    <option value="usa">USA</option>
    <option value="uk">UK</option>
    <option value="australia">Australia</option>
  </select><br><br>
   <input type="submit" value="Register">
</form>
```

welcome.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome Page</title>
</head>
<body>
  <h2>Welcome!</h2>
  <p>Thank you for submitting your details. Your registration is successful.</p>
</body>
</html>
```

Output:



Lab Practice Questions

1. Create a feedback form that includes text fields for name and email, a text area for feedback, and a submit button.

2. Design a login form with username and password fields, including "Remember Me" checkbox and a "Login" button.

3. Create a course registration form that allows the user to select a course from a drop-down list and choose multiple preferred timings using checkboxes.

4. Design a contact form with fields for full name, phone number, email, and message. Validate the phone number to accept only 10 digits using the pattern attribute.

5. Create a student information form grouped into two sections using <fieldset> and <legend> , one for personal details and another for academic details.

Experiment No: 8

BASIC CONSTRUCTS OF JAVASCRIPT

Objectives

- to familiarize with JavaScript syntax and its basic constructs
- to use variables and data types effectively
- to implement arithmetic and logical operations
- to apply control structures and loops in programs
- to define and use functions for performing specific tasks

Theory

JavaScript is a powerful and widely used scripting language that adds interactivity and dynamic behavior to web pages. It works alongside HTML and CSS to create responsive and user-friendly web applications. The basic constructs of JavaScript form the foundation of programming in this language. These include variables, data types, operators, control structures, loops, and functions. Understanding these constructs is essential for writing efficient programs that can perform calculations, make decisions, repeat tasks, and respond to user actions. Mastering these core elements enables developers to build logical, structured, and interactive web content.

2.1.1 Fundamentals of JAVASCRIPT

JavaScript is a client-side scripting language used to make web pages dynamic and interactive. It can be embedded directly into HTML documents and executed by the web browser.

2.1.1.1 Variables and Data Types

A variable in JavaScript is a container used to store data values. It allows programmers to store, modify, and retrieve information during program execution.

Syntax:

```
var variableName = value;

let variableName = value;

const variableName = value;
```

Here,

- **var** : Declares a variable (function-scoped).

- **let** :Declares a block-scoped variable (introduced in ES6).
- **const** : Declares a constant whose value cannot be changed.

Example:

- var name = "John";
- let age = 25;
- const pi = 3.14;

Data types define the type of data that a variable can hold, such as numbers, text, or Boolean values. JavaScript is a loosely typed or dynamic language, meaning you don't need to declare the type of variable before using it, the type is determined automatically at runtime. Common data types of Javascript are described in Table 2.1.1.

Table 2.1.1 Common Data Types in JavaScript

Data Type	Description	Example
String	Represents text values enclosed in quotes.	"Hello", 'World'
Number	Represents numeric values (integer or floating point).	10, 3.14
Boolean	Represents logical values.	true, false
Undefined	A variable declared but not assigned a value.	let x;
Null	Represents an intentional absence of any value.	let x = null;
Object	Collection of key–value pairs.	{name: "John", age: 25}
Array	Ordered list of values.	[10, 20, 30]

Exercise Questions

Exercise 1: Write a JavaScript program to implement the concept of variables.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Variables Example</title>
</head>
<body>
   <script>
      var name = "John";      // String variable
```

```
        let age = 25;           // Number variable
        const pi = 3.14;        // Constant value
        document.write("Name: " + name + "<br>");
        document.write("Age: " + age + "<br>");
        document.write("Value of PI: " + pi);
    </script>
</body>
</html>
```

Output:

Name: John

Age: 25

Value of PI: 3.14

Exercise 2: Write a JavaScript program to d**isplaying different data types.**

Source Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Data Types Example</title>
</head>
<body>
    <script>
        let name = "Alice";                          // String
        let age = 22;                                // Number
        let isStudent = true;                        // Boolean
        let marks = [85, 90, 78];                    // Array
        let person = {name: "Alice", city: "Chennai"};   // Object
        document.write("Name: " + name + "<br>");
        document.write("Age: " + age + "<br>");
        document.write("Is Student: " + isStudent + "<br>");
        document.write("Marks: " + marks + "<br>");
```

```
        document.write("Person Details: " + person.name + ", " + person.city);
    </script>
</body>
</html>
```

Output:

Name: Alice

Age: 22

Is Student: true

Marks: 85,90,78

Person Details: Alice, Chennai

2.1.1.2 Operators

Operators in JavaScript are special symbols used to perform operations on operands (variables and values). They allow programmers to perform tasks such as arithmetic calculations, comparisons, and logical decisions. (Table 2.1.2).

For example, let sum = 10 + 5;

Here, + is an operator, and 10 and 5 are operands.

Syntax: result = operand1 operator operand2;

Example: let total = a + b;

Table 2.1.2 Types of Operators in JavaScript

Type	Description	Example
1. Arithmetic Operators	Used for mathematical operations.	+, -, *, /, %, ++, --
2. Assignment Operators	Used to assign values to variables.	=, +=, -=, *=, /=
3. Comparison Operators	Used to compare two values.	==, ===, !=, >, <, >=, <=
4. Logical Operators	Used to combine conditions.	&&, ||, !
5. String Operators	Used to combine strings.	+, +=
6. Ternary Operator	Acts as a shorthand for if... else.	condition ? value1 : value2

Exercise Questions

Exercise 3: Write a JavaScript program to implement the concept of arithmetic operators.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Arithmetic Operators</title>
</head>
<body>
   <script>
      let a = 10, b = 5;
      document.write("Addition: " + (a + b) + "<br>");
      document.write("Subtraction: " + (a - b) + "<br>");
      document.write("Multiplication: " + (a * b) + "<br>");
      document.write("Division: " + (a / b) + "<br>");
      document.write("Remainder: " + (a % b));
   </script>
</body>
</html>
```

Output:

Addition: 15

Subtraction: 5

Multiplication: 50

Division: 2

Remainder: 0

Exercise 4: Write a JavaScript program to implement the concept of comparison and logical operators.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Comparison and Logical Operators</title>
</head>
<body>
   <script>
      let x = 20, y = 15;
      document.write("x > y : " + (x > y) + "<br>");
      document.write("x == y : " + (x == y) + "<br>");
      document.write("(x > 10 && y < 30): " + (x > 10 && y < 30));
   </script>
</body>
</html>
```

Output:

x > y : true

x == y : false

(x > 10 && y < 30): true

Exercise 5: Write a JavaScript program to implement the concept of ternary operator.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Ternary Operator</title>
</head>
<body>
   <script>
      let age = 18;
      let result = (age >= 18) ? "Eligible to vote" : "Not eligible";
      document.write(result);
   </script>
</body>
</html>
```

Output:

Eligible to vote

2.1.1.3 Conditional Statements

Conditional statements in JavaScript are used to make decisions in a program. They allow the program to execute certain blocks of code based on whether a specified condition is true or false. Types of conditional statements are described in Table 2.1.3.

For example, "If the student's mark is above 50, display 'Pass'; otherwise, display 'Fail'."

Syntax:

```
if (condition) {
   // Code to execute if condition is true
}
else {
   // Code to execute if condition is false
}
```

Table 2.1.3 Types of Conditional Statements

Statement	Description	Example
if	Executes code block if condition is true.	if (x > 10)
if...else	Executes one block if condition is true, another if false.	if (x > 10) else
if...else if...else	Used to test multiple conditions.	if (x > 0) else if (x == 0)
switch	Used when multiple conditions depend on one expression.	switch(choice)

Exercise Questions

Exercise 6: Write a JavaScript program to implement the concept of if Statement.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>if Statement Example</title>
</head>
<body>
```

```
    <script>
        let age = 20;
        if (age >= 18) {
            document.write("You are an adult.");
        }
    </script>
</body>
</html>
```

Output:

You are an adult.

Exercise 7: Write a JavaScript program to implement the concept of if...else statement.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>if else Example</title>
</head>
<body>
    <script>
        let num = 7;
        if (num % 2 == 0) {
            document.write(num + " is Even");
        } else {
            document.write(num + " is Odd");
        }
    </script>
</body>
</html>
```

Output:

7 is Odd

Exercise 8: Write a JavaScript program to implement the concept of if...else if...else statement.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>if else if Example</title>
</head>
<body>
   <script>
      let marks = 85;
      if (marks >= 90) {
         document.write("Grade: A+");
      } else if (marks >= 75) {
         document.write("Grade: A");
      } else if (marks >= 60) {
         document.write("Grade: B");
      } else {
         document.write("Grade: C");
      }
   </script>
</body>
</html>
```

Output:

Grade: A

Exercise 9: Write a JavaScript program to implement the concept of switch statement.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Switch Example</title>
</head>
<body>
```

```
    <script>
        let day = 3;
        switch(day) {
            case 1: document.write("Monday"); break;
            case 2: document.write("Tuesday"); break;
            case 3: document.write("Wednesday"); break;
            case 4: document.write("Thursday"); break;
            case 5: document.write("Friday"); break;
            default: document.write("Weekend");
        }
    </script>
</body>
</html>
```

Output:

Wednesday

2.1.1.4 Loops

A **loop** in JavaScript is used to execute a block of code repeatedly until a specified condition becomes false. Loops help reduce repetition and make code shorter, cleaner, and more efficient. Types of Loops in JavaScript explained in Table 2.1.4.

For example, Instead of writing `document.write(1);`

`document.write(2); document.write(3);,`

we can use a loop to print all numbers automatically.

Syntax:

```
for (initialization; condition; increment/decrement) {
   // Code to execute repeatedly
}
```

Table 2.1.4 Types of Loops in JavaScript

Loop Type	Description	Syntax Example
For	Runs a block of code a specific number of times.	for(let i=1; i<=5; i++)
while	Runs while a condition is true.	while(i<=5)
do...while	Runs at least once, even if the condition is false.	do { } while(i<=5)

for...in	Loops through properties of an object.	for (let key in obj)
for...of	Loops through elements of an iterable (like arrays).	for (let value of arr)

Exercise Questions

Exercise 10: Write a JavaScript program to implement the concept of *for* loop.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>For Loop Example</title>
</head>
<body>
  <script>
    for (let i = 1; i <= 5; i++) {
      document.write("Number: " + i + "<br>");
    }
  </script>
</body>
</html>
```

Output:

Number: 1

Number: 2

Number: 3

Number: 4

Number: 5

Exercise 11: Write a JavaScript program to implement the concept of *while* loop.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>While Loop Example</title>
```

```
</head>
<body>
   <script>
      let i = 1;
      while (i <= 5) {
         document.write("Count: " + i + "<br>");
         i++;
      }
   </script>
</body>
</html>
```

Output:

Count: 1

Count: 2

Count: 3

Count: 4

Count: 5

Exercise 12: Write a JavaScript program to implement the concept of *do… while* **loop.**

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Do While Loop Example</title>
</head>
<body>
   <script>
      let i = 1;
      do {
         document.write("Value: " + i + "<br>");
         i++;
      } while (i <= 3);
   </script>
</body>
</html>
```

Output:

Value: 1

Value: 2

Value: 3

Exercise 13: Write a JavaScript program to implement the concept of *for….in (object)* **loop.**

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>for...in Loop Example</title>
</head>
<body>
   <script>
      let person = {name: "Ravi", age: 25, city: "Chennai"};
      for (let key in person) {
         document.write(key + ": " + person[key] + "<br>");
      }
   </script>
</body>
</html>
```

Output:

name: Ravi

age: 25

city: Chennai

Exercise 14: Write a JavaScript program to implement the concept of *for…. of (Array)* **loop.**

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>for...of Loop Example</title>
</head>
```

```
<body>
  <script>
    let fruits = ["Apple", "Banana", "Mango"];
    for (let fruit of fruits) {
      document.write(fruit + "<br>");
    }
  </script>
</body>
</html>
```

Output:

Apple

Banana

Mango

2.1.1.5 Functions

A function in JavaScript is a block of reusable code designed to perform a specific task. Functions make programs modular, reduce repetition, and improve readability. They can take inputs (called parameters) and return an output (called return value). Types of functions in JavaScript shown in Table 2.1.5.

For example, instead of writing the same code multiple times, you can write a function once and call it whenever needed.

Syntax:

```
function functionName(parameters) {
  // code to be executed
  return value;   // optional
}
```

Calling a Function: functionName(arguments);

Table 2.1.5 Types of Functions in JavaScript

Type	Description	Example
User-defined Function	Created by the user for specific tasks.	function add(a,b){...}
Built-in Function	Provided by JavaScript itself.	alert(), parseInt(), document.write()

Function with Parameters	Takes input values while calling.	function greet(name)
Function with Return Value	Returns a result after execution.	return sum;
Anonymous Function	Function without a name, often used in events.	function() { ... }
Arrow Function (ES6)	Shorter syntax for functions.	let add = (a,b) => a+b;

Exercise Questions

Exercise 15: Write a JavaScript program to implement the concept of simple function.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Simple Function</title>
</head>
<body>
   <script>
      function greet() {
         document.write("Welcome to JavaScript Functions!");
      }
      greet(); // function call
   </script>
</body>
</html>
```


Output:

Welcome to JavaScript Functions!

Exercise 16: Write a JavaScript program to implement the concept of function with parameters.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Function with Parameters</title>
</head>
<body>
   <script>
      function displayInfo(name, age) {
         document.write("Name: " + name + "<br>");
         document.write("Age: " + age);
      }
      displayInfo("Rahul", 20);
   </script>
</body>
</html>
```

Output:

Name: Rahul

Age: 20

Exercise 17: Write a JavaScript program to implement the concept of function with return value.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Function with Return Value</title>
</head>
<body>
   <script>
      function add(a, b) {
         return a + b;
      }
      let result = add(10, 5);
```

```
        document.write("Sum = " + result);
    </script>
</body>
</html>
```

Output:

Sum = 15

Exercise 18: Write a JavaScript program to check whether a number is even or odd.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Even or Odd</title>
</head>
<body>
    <h3>Even or Odd Number</h3>
    <script>
        let number = 7;
        if (number % 2 == 0) {
            document.write(number + " is even");
        } else {
            document.write(number + " is odd");
        }
    </script>
</body>
</html>
```


Output:

7 is odd

Exercise 19: Write a JavaScript function to find the sum of two numbers.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Function Example</title>
</head>
<body>
   <h3>Sum of Two Numbers using Function</h3>
   <script>
      function addNumbers(a, b) {
         return a + b;
      }
      let result = addNumbers(12, 18);
      document.write("Sum = " + result);
   </script>
</body>
</html>
```

Output:

Sum = 30

2.1.2 Procedure

General steps to run a JavaScript program as follows:

1. **Open a Text Editor:** Use any text editor such as Notepad, Visual Studio Code, or Sublime Text to write your JavaScript code.

2. **Write the JavaScript Code:** You can write JavaScript in two ways.
   - **Internal JavaScript:** Inside an HTML file using the <script> tag.
   - **External JavaScript:** In a separate .js file linked to an HTML file.

3. **Save the File:**
   - If using **internal JavaScript**, save the file with the .html extension (e.g., program.html).

- If using **external JavaScript**, save the JavaScript file as .js (e.g., script.js) and link it in your HTML file.

4. **Open the HTML File in a Browser:**
   - Double-click the HTML file or
   - Right-click and select **"Open with → Browser"** (e.g., Chrome, Edge, Firefox).

5. **View the Output:**
   - The output will appear on the **webpage** or in the **browser console** (open console using Ctrl + Shift + I → **Console tab**).

6. **Modify and Re-run (if needed):** Make changes in the code, save the file again, and refresh the browser to see updated results.

Lab Practice Questions

1. Write a JavaScript program to display your name, course, and college name on a webpage. (Hint: Use document.write() statement.)

2. Write a JavaScript program to find whether a number entered by the user is positive, negative, or zero using `if…else if…else`.

3. Write a JavaScript program to print the first 10 natural numbers using a `for` loop.

4. Write a JavaScript program that defines a function `square()` which accepts a number and returns its square.

Experiment No: 9

Title: Form Validation using Javascript

Objectives

- Apply JavaScript functions to validate user input before submitting an HTML form
- Apply conditional logic to check whether required form fields are filled correctly
- Apply validation techniques to ensure numeric input falls within a specific range

Theory

Form validation is one of the most important aspects of web development. It ensures that users enter the correct and complete information before a form is submitted to the server. Using JavaScript, validation can be performed on the client side meaning the browser checks the data before sending it to the server. This reduces unnecessary server load and enhances user experience by providing instant feedback.

The process of validation usually involves checking whether:

- Required fields are filled.
- The data entered is of the correct type (text, number, email, etc.).
- The input value falls within an acceptable range.

JavaScript functions are typically called when a form is submitted. If a validation condition fails, the function displays an error message and prevents the form from being submitted by returning false.

2.9.1 Validate Empty Fields Using JavaScript

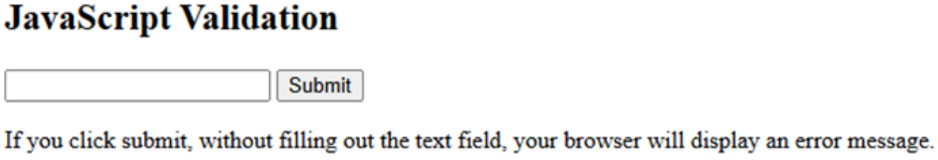
This example checks if the user has entered a value in the text field before submitting the form. If the field is empty, an alert message is shown, and the form submission is stopped.

Sample Program

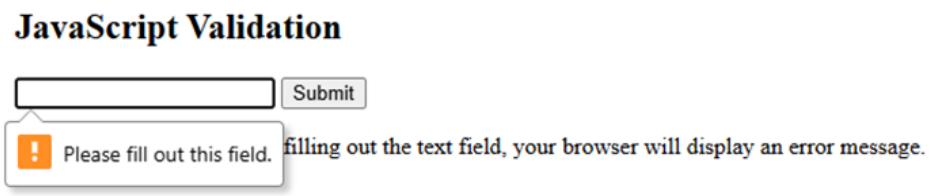
```
<!DOCTYPE html>

<html>

<head>
```

```
<title>Form Validation Example</title>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
<h2>Form Validation Using JavaScript</h2>
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Expected Output



If the user clicks the Submit button without entering a name, an alert box displays "Name must be filled out," and the form will not be submitted.



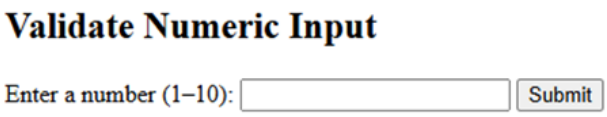
2.9.2 Validate Numeric Input Using JavaScript

JavaScript can also be used to ensure that users enter numbers within a specified range. In the following example, the user is prompted to enter a number between 1 and 10.

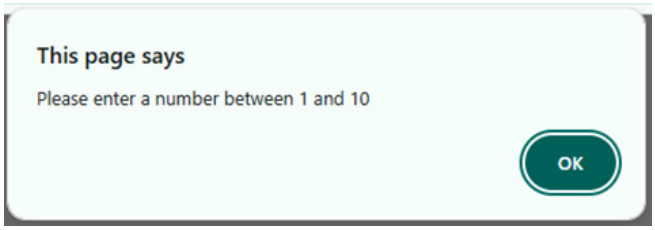
Sample Program

```
<!DOCTYPE html>
<html>
<head>
<title>Numeric Validation</title>
<script>
function validateNumber() {
  let x = document.getElementById("num").value;
  if (isNaN(x) || x < 1 || x > 10) {
    alert("Please enter a number between 1 and 10");
    return false;
  }
}
</script>
</head>
<body>
<h2>Validate Numeric Input</h2>
<form onsubmit="return validateNumber()">
  Enter a number (1–10):
  <input type="text" id="num">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Expected Output



If the entered value is not a number or is outside the range 1–10, an alert box appears, and form submission is prevented.



2.9.3 Combine Multiple Validations

You can combine several validation checks in a single form — for example, checking if the name field is filled and if the age field contains a valid number.

Sample Program

```
<!DOCTYPE html>
<html>
<head>
<title>Combined Validation</title>
<script>
function validateForm() {
  let name = document.forms["userForm"]["uname"].value;
  let age = document.forms["userForm"]["uage"].value;
  if (name == "") {
    alert("Name must be filled out");
    return false;
  }
  if (isNaN(age) || age < 1 || age > 120) {
    alert("Please enter a valid age between 1 and 120");
    return false;
  }
```

```
    }
    </script>
    </head>
    <body>
    <h2>Combined Form Validation</h2>
    <form name="userForm" onsubmit="return validateForm()">
      Name: <input type="text" name="uname"><br><br>
      Age: <input type="text" name="uage"><br><br>
      <input type="submit" value="Submit">
    </form>
    </body>
    </html>
```

Expected Output

- If the name field is empty, an alert asks the user to fill it out.
- If the age is not a number or not between 1 and 120, another alert appears.
- The form will submit only when all fields are valid.

2.9.4 Lab Practice questions

1. Create a web form with fields for Name, Email, and Phone Number. Validate that none of these fields are left blank before submission.
2. Design a form where the user must enter their age between 18 and 60. Display an alert message if the input is invalid.
3. Develop a registration form that validates both name (non-empty) and numeric ID (must be a number).
4. Create a form with two fields: Password and Confirm Password. Validate that both entries match before submission.
5. Build a simple feedback form that checks whether the email field contains an "@" symbol and is not left empty.
6. Extend the combined validation program to include a dropdown list for gender selection. Ensure that a gender is selected before submission.
7. Experiment with different alert messages to display customized feedback for various validation failures.

Experiment No: 10

Creating Animated Gifs and Lists

Objectives

- Create a simple animated GIF from multiple image frames (using a free tool)
- Embed and use the animated GIF in a webpage
- Create HTML lists (ordered, unordered, nested) and apply CSS styling
- Add simple animations to list items (hover effects and keyframe-based animated lists) to demonstrate motion in UI

Theory

2.10.1 Materials / Software

- Computer with internet access (for downloading tools or sample images).
- GIMP (or alternative) installed, or access to an online GIF maker.
- Sample PNG/JPG images (3–8 frames) — you can create frames by exporting different stages of a drawing or use photos.
- Text editor and browser.

2.10.2 Create an animated GIF (using GIMP)

Steps

1. Open GIMP.
2. Create or import the frames: File → Open as Layers... and select all frame images in the order you want them to appear. Each layer becomes one GIF frame.
3. (Optional) Resize images to the same dimensions: Image → Scale Image....
4. Set the frame delay by renaming layers or using Filters: in the Layers panel, rename a layer like frame1 (200ms) to set delay. Default is usually 100ms.
5. Export as GIF: File → Export As... → choose filename.gif.
6. In the export dialog choose As animation, set looping (e.g., forever), and the default frame delay if needed. Click Export.
7. Open the GIF in your browser to verify animation.

Deliverable

- my_animation.gif saved locally.

2.10.3 Embed and display animated GIF in HTML

Create a simple HTML file to display the GIF.

```
<!-- save as display_gif.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Animated GIF Demo</title>
  <style>
    body { font-family: Arial, sans-serif; padding: 20px; background:#f7f7f7; }
        .preview { max-width: 320px; border: 2px solid #ccc; padding:8px; background:#fff; }
    .caption { margin-top:8px; font-size:14px; color:#333; }
  </style>
</head>
<body>
  <h2>Animated GIF Preview</h2>
  <div class="preview">
    <img src="my_animation.gif" alt="Example animation" style="width:100%; display:block;" />
    <div class="caption">Filename: my_animation.gif — loops forever</div>
  </div>
</body>
</html>
```

Open display_gif.html in your browser to view.

2.10.4 HTML Lists: creation and styling

1. Basic HTML lists

```
<!-- save as lists.html -->
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8" />
 <title>Lists and Animations</title>
 <style>
  body { font-family: Arial, sans-serif; padding:20px; }
  .styled-ul {
   list-style-type: none; /* remove bullets */
   padding: 0;
   width: 320px;
  }
  .styled-ul li {
   background: #fff;
   margin: 8px 0;
   padding: 12px 14px;
   border-radius: 8px;
   box-shadow: 0 1px 3px rgba(0,0,0,0.08);
   transition: transform 0.2s ease, box-shadow 0.2s ease;
  }
  .styled-ul li:hover {
   transform: translateY(-4px);
   box-shadow: 0 8px 20px rgba(0,0,0,0.12);
  }
  /* nested list styling */
  .nested { margin-left: 18px; font-size: 0.95em; color:#444; }
  /* small icon bullet using pseudo-element */
  .icon-bullet {
   position: relative;
   padding-left: 28px;
```

```
  }
   .icon-bullet::before {
    content: "□";
    position: absolute;
    left: 8px;
    top: 12px;
    font-size: 12px;
    color:#ff6600;
   }
  </style>
</head>
<body>
  <h3>Unordered List (styled)</h3>
  <ul class="styled-ul">
    <li class="icon-bullet">HTML Basics
      <ul class="nested">
        <li>Tags</li>
        <li>Attributes</li>
      </ul>
    </li>
    <li class="icon-bullet">CSS Fundamentals</li>
    <li class="icon-bullet">JavaScript Intro</li>
  </ul>
  <h3>Ordered List</h3>
  <ol>
    <li>Install editor</li>
    <li>Create files</li>
    <li>Open in browser</li>
  </ol>
</body>
</html>
```

2. Animated list using CSS keyframes

Add an auto-entrance animation for items.

```
/* add inside <style> of lists.html */
@keyframes slideIn {
  from { transform: translateX(-30px); opacity: 0; }
  to { transform: translateX(0); opacity: 1; }
}
.styled-ul li {
  animation: slideIn 500ms ease forwards;
}
.styled-ul li:nth-child(1) { animation-delay: 0ms; }
.styled-ul li:nth-child(2) { animation-delay: 120ms; }
.styled-ul li:nth-child(3) { animation-delay: 240ms; }
```

This creates a staggered entrance effect similar to an animated list.

2.10.5 Lab practice Questions

1. Create an animated GIF of at least 5 frames (size 300×300 px). Embed it in a webpage with a caption and download link.

2. Create three types of lists: default <ul>, <ol> with Roman numerals, and a nested list. Style them using CSS (change bullets, add icons, spacing).

3. Implement an animated list where items fade in one-by-one on page load (use @keyframes and nth-child delays).

4. Replace the GIF you created with a CSS-only animation that reproduces a similar motion compare pros/cons (file size, resolution, responsiveness).

MODEL QUESTION PAPER SETS

SREENARAYANAGURU OPEN UNIVERSITY

MODEL QUESTION PAPER - SET 1

QP CODE: ……

Reg. No:.....

Name: ……..

FIFTH SEMESTER EXAMINATION

DISCIPLINE CORE COURSE

BACHELOR OF COMPUTER APPLICATIONS

B21CA09DC - WEB TECHNOLOGY

Time: 3 Hours MaxMarks:70

Section A

Answer any 10 questions. Each carries one mark

(10x1=10)

1. Which HTML tag is used to display an image?
2. Which CSS property sets the vertical position of an element?
3. Which HTML element was used to divide a browser window into multiple sections?
4. Which control is used to set date in HTML forms?
5. Which HTML tag is used to include JavaScript in a web page?
6. What data type holds true or false?
7. What is the only button available in an alert box?
8. Which two methods are most commonly used to send data to the server?
9. What does CSS stand for?
10. Write about the speciality of group selector.
11. Which CSS property is used to change the text color?
12. Which CSS property is used to create rounded corners on an element?
13. What is the symbol of a comment in XML?
14. How many root elements can an XML document have?
15. What is the main purpose of using a DTD?

Section B

Answer any 5 questions. Each carries two marks

(5x2=10)

16. Write about any four HTML form attributes.

17. How is JavaScript embedded in an HTML document?

18. What are the properties of Document Objects?

19. Define Alert Box.

20. What are Fieldsets and Legends?

21. What are the three types of CSS?

22. Explain about CSS Pseudo-Elements and types.

23. What is the use of the CSS background-image property?

24. List any two values of the overflow property in CSS.

25. Explain XML vs HTML.

Section C

Answer any 5 questions. Each carries four marks

(5x4=20)

26. Compare XML and HTML.

27. What is XML Schema Definition (XSD)? Explain its key features

28. Write about any 4 common form controls?

29. Describe the HTML <audio> tag and its attributes.

30. Write a JavaScript program that declares variables, performs arithmetic operations, and displays the result using document.write().

31. Define Confirm Box.

32. What is the object handling method?

33. What is DOM (Document Object Model)? Explain its role in XML processing.

34. Write about CSS class selector and ID selector.

35. Describe any four CSS properties used for text and font styling.

Section D

Answer any 2 questions. Each carries fifteen mark

(2x15=30)

36. Define a form in HTML. Describe the various form controls. Write a HTML program to design a customer details input form.

37. Explain Line Breaks in Popup Boxes.

38. Discuss the purpose and advantages of group selectors in CSS. Illustrate with an example.

39. Compare XPath, XSLT, XQuery, and DOM in detail based on purpose, working, output, and applications.

SREENARAYANAGURU OPEN UNIVERSITY

MODEL QUESTION PAPER - SET 2

QP CODE: ……

Reg. No:.....

Name: ……..

FIFTH SEMESTER EXAMINATION

DISCIPLINE CORE COURSE

BACHELOR OF COMPUTER APPLICATIONS

B21CA09DC - WEB TECHNOLOGY

Time: 3 Hours MaxMarks:70

Section A

Answer any 10 questions. Each carries one mark

(10x1=10)

1. Which attribute specifies the destination of a link?
2. Which HTML tag displays an image on a webpage?
3. Which <audio> attribute displays play and pause controls?
4. Write about the Get method.
5. JavaScript primarily runs on which side in the client–server model?
6. What keyword refers to the current object?
7. Which character is used in JavaScript for line breaks?
8. Which technology helps in making real-time dynamic updates to XML documents?
9. Which HTML tag is used to link an external CSS file?
10. Write about the purpose of hover Pseudo Class.
11. Which CSS property is used to set the background color of an element?
12. What is the default value of the float property?
13. What character represents “&” in XML text?
14. What does XML stand for?
15. What is XML validation?

Section B

Answer any 5 questions. Each carries two marks

(5x2=10)

16. Write about enctype and target attributes of form tag.
17. Explain how JavaScript works in the client–server model.
18. What are data types in JavaScript?
19. Define prompt box.
20. Explain Radio Buttons Control with an example.
21. Name any two CSS properties used for text formatting.
22. Explain about attribute selectors.
23. What is the purpose of the CSS font-family property?
24. Differentiate between stretched and repeated border images in CSS
25. Define XML Syntax Rules.

Section C

Answer any 5 questions. Each carries four marks

(5x4=20)

26. Write an HTML code that uses the required attribute to make certain fields mandatory.
27. Explain the structure of an XML document with the help of a suitable example
28. Explain about Form Accessibility and Usability
29. Differentiate between <frame> and <iframe> in HTML.
30. What are the Non-Primitive Data Types in JavaScript ?
31. Explain different types of Pop up Boxes.
32. Write an HTML program to create a simple login form with username and password fields and a submit button.
33. Explain the role of XPath, XSLT, XQuery, and DOM in working with XML documents

34. Explain about universal selector and element selector.

35. Explain the CSS Box Model with a neat description.

Section D

Answer any 2 questions. Each carries fifteen mark

(2x15=30)

36. Explain frames in HTML, audio in HTML, and the <embed> tag with suitable syntax and examples.

37. Discuss about Objects and Operators in JavaScript

38. Detailed note on Text Color and Formatting in CSS.

39. Differentiate between nested elements and empty elements in XML. Provide suitable syntax examples to highlight their use and importance in structuring data.

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

DON'T LET IT BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

Web Technology

COURSE CODE: B21CA09DC

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841

ISBN 978-81-996171-5-5