



DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

**Study Material for Academic Year 2024-25
(ODD Semester: Sep - Dec 2024)**

COURSE CODE : 22ISE342

COURSE NAME : WEB DESIGN TECHNOLOGIES

SEMESTER :III

**Name of the Faculty: Mrs. Priya N
Mrs. Shubhi Srivastava
Mrs. Krishnaveni**

Course Code: 22ISE342

Credits:03

L:T:P:S :2:0:1:0

CIE Marks:50

Exam Hours:3

SEE Marks:50

WEB DESIGN TECHNOLOGIES														
Course Code	22ISE342								CIE Marks		50			
L:T:P:S	2:0:1:0								SEE Marks		50			
Hrs / Week	2+2								Total Marks		100			
Credits	03								Exam Hours		03			
Course outcomes: At the end of the course, the student will be able to:														
22ISE342.1	Design webpages using XHTML and HTML5.													
22ISE342.2	Design webpages using Cascading Style Sheets.													
22ISE342.3	Develop JavaScript programs to validate dynamic Webpages.													
22ISE342.4	Develop Javascript and DHTML programs.													
22ISE342.5	Describe the methods to handle XML and PHP programming..													
22ISE342.6	Inspect the management of state in web applications and JavaScript frameworks which facilitates developer to focus on core features.													
Mapping of Course Outcomes to Program Outcomes and Program Specific Outcomes:														
	P01	P02	P03	P04	P05	P06	P07	P08	P09	P010	P011	P012	PS01	PS02
22ISE342.1	3	2	3	-	3	3	-	-	1	-	1	3	3	3
22ISE342.2	3	2	3	-	3	3	-	-	1	-	1	3	3	3
22ISE342.3	3	2	3	-	3	3	-	-	1	-	1	3	3	3
22ISE342.4	3	2	3	-	3	3	-	-	1	-	1	3	3	3
22ISE342.5	3	2	3	-	3	3	-	-	1	-	1	3	3	3
22ISE342.6	3	2	3	-	3	3	-	-	1	-	1	3	3	3
MODULE-1	XHTML								22ISE342.1, 22ISE342.2				6 Hours	
XHTML: Basic syntax, Standard XHTML document structure; Basic text markup, Images; Hypertext Links, Lists, Tables, Forms, Syntactic differences between HTML and XHTML Cascading Style Sheets: Introduction, Levels of style sheets, Style specification formats, Selector forms, The Box model, Background images, The and <div> tags														
Laboratory Component: 1. Design a personal web page using HTML5 which should include: a.) A brief description about yourself. b.)Your photo as the profile picture using canvas c.)An index which should be a list of different headings/sections present in a document in the form of link which when clicked takes you to that heading/section The different sections: <ul style="list-style-type: none">Your educational details(Has to be displayed using a table)Your Achievements. 2. Apply styles to the web page using CSS													3 Hours	
Text Book			Text Book 1: Ch2, Ch3											
MODULE-2	HTML 5								22ISE342.2				6 Hours	
Detecting HTML 5 features – Advanced CSS: Layout, Normal Flow, Positioning Elements, Floating Elements. Canvas, video, local storage, web workers, offline applications, geo-location, input types. Let’s call it drawing surface – Simple shapes, canvas, Paths ,texts, gradients and images.														
Laboratory Component: 1. Design a webpage form using the textbox, checkbox, radio buttons, submit andreset buttons 2. Write a HTML Program to design a simple calculator.													3 Hours	
Text Book		Text Book 1: Ch 4												
MODULE-3	Javascript								22ISE342.3				6 Hours	

Overview of JavaScript, General syntactic characteristics, Screen output and keyboard input, Control statements, Object creation and modification, Arrays, Functions, Constructor, Pattern matching using regular expressions.				
Laboratory Component: 1. Write a Program to display current date and time using HTML5 SemanticElements. 2. Write a JavaScript Program for the following problem: a. Input: A number n obtained using prompt Output: The first n Fibonacci numbers b. input : A number output : factorial of the number.				3 Hours
Text Book	Text Book 4 : Chapter 5			
MODULE-4	Javascript and HDML Documents	22ISE342.4	6 Hours	
JavaScript and DHTML Documents: The Document Object Model, Element access in JavaScript , Events and event handling. Moving elements, Element visibility, Dynamic content, Slow movement of elements.				
Laboratory Component: 1. Design and develop a XHTML document that includes JavaScript script to create stack of images such that images appear one top on another with images slightly visible. Whenever cursor is placed on an image that image should be completely visible and on moving cursor out image should go back to original position 2. Develop and demonstrate, using Javascript, a XHTML document that collects the USN (the valid format is: A digit from 1 to 4 followed by two upper-case characters followed by two digits followed by two upper-case characters followed by three digits; no embedded spaces allowed) and semester (valid format digit from 1 to 8) of the user. Event handler must be included for the form element that collects this information to validate the input. Messages in the alert windows must be produced when errors are detected.				3 Hours
Self-study / Case Study / Applications	Download any business data set [House price, car resale value etc] and perform cleaning operation of the data, followed by that use the knowledge you can acquired to find the key insight about the data and summarize the same.			
Text Book	Text Book 4 : Chapter 6,7			
MODULE-5	Basics of PHP and XML	22ISE342.5, 22ISE342.5,	6 Hours	
PHP: Origins and uses of PHP, Overview of PHP, General syntactic characteristics, Output, Control statements, Arrays, Functions, Pattern matching, Form handling, Files, Cookies XML: Introduction to XML, The Syntax of XML, Document structure, Document Type Definition (DTD).				
Laboratory Component: 1. Design a web page using XHTML and PHP to store current date-time in a COOKIE and display the 'Last visited on' date-time on the web page upon reopening of the same page.				3 Hours
Text Book	Text Book 3 : Chapter 4			
CIE Assessment Pattern (50 Marks – Theory and Lab)				
RBT Levels		Marks Distribution		
		Test (s)	Qualitative Assessment	Lab
		25	05	20
L1	Remember	5	-	-
L2	Understand	5	-	5
L3	Apply	5	5	10
L4	Analyze	10	-	5
L5	Evaluate	-	-	-
L6	Create	-	-	-

SEE Assessment Pattern (50 Marks – Theory)

RBT Levels		Exam Marks Distribution (50)
L1	Remember	10
L2	Understand	10
L3	Apply	10
L4	Analyze	20
L5	Evaluate	--
L6	Create	--

Suggested Learning Resources:

Text Books:

- 1) Robert W. Sebesta, "Programming the World Wide Web", 8th Edition, Pearson Education, 2015.
- 2) Randy Connolly, Ricardo Hoar, "Fundamentals of Web Development", 4th Edition, Pearson Education India, 2016
- 3) Mark Pilgrim, "HTML5: Up and Running: Dive into HTML5", 1st Edition, O'Reilly, Google Press Publishers & Distributors Pvt Ltd, 2010

Reference Books:

- 1) Paul Deitel, Harvey Deitel, Abbey Deitel, "Internet & World Wide Web How to program", 5th Edition, Pearson Education/PHI, 2012.
- 2) Robin Nixon, "Learning PHP, My SQL & Java Script with jQuery, CSS and HTML5", 5th Edition, O'Reilly Publications, 2018.

Web links and Video Lectures (e-Resources):

- https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction
- <https://www.browserstack.com/guide/top-html5-features>
- https://www.w3schools.com/php/php_intro.asp
- https://www.w3schools.com/js/js_operators.asp https://onlinecourses.swayam2.ac.in/aic20_sp11/preview

Activity-Based Learning (Suggested Activities in Class)/ Practical Based learning

- Demonstration mini projects.
- Contents related activities (Activity-based discussions)
- Organizing Group wise discussions
- Seminars

MODULE 1

XHTML

HTML Documents

All HTML documents must start with a type declaration: `<!DOCTYPE html>`.

The HTML document itself begins with `<html>` and ends with `</html>`.

The visible part of the HTML document is between `<body>` and `</body>`.

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags:

Example

```
<h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

Example

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

HTML Links

HTML links are defined with the `<a>` tag:

Example

```
<a href="https://newhorizoncollegeofengineering.in/">This is a link</a>
```

The link's destination is specified in the **href attribute**.

Attributes are used to provide additional information about HTML elements.

HTML Images

HTML images are defined with the **** tag.

The source file (**src**), alternative text (**alt**), and size (**width** and **height**) are provided as **attributes**:

Example

```

```

HTML Basics

HTML (HyperText Markup Language) is a markup language used to structure and organize the content on a web page. It uses various tags to define the different elements on a page, such as headings, paragraphs, and links.

HTML Hierarchy

HTML elements are hierarchical, which means that they can be nested inside each other to create a tree-like structure of the content on the web page.

This hierarchical structure is called the DOM (Document Object Model), and it is used by the web browser to render the web page. For example,

```
<!DOCTYPE html>
<html>
  <head>
    <title>My web page</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>This is my first web page.</p>
    <p>It contains a
      <strong>main heading</strong> and <em> paragraph </em>.
    </p>
  </body>
</html>
```

Browser Output

Hello, world!

This is my first web page.

It contains a **main heading** and *paragraph*.

In this example, the `html` element is the root element of the hierarchy and contains two child elements: `head` and `body`. The `head` element, in turn, contains a child element called the title, and the `body` element contains child elements: `h1` and `p`.

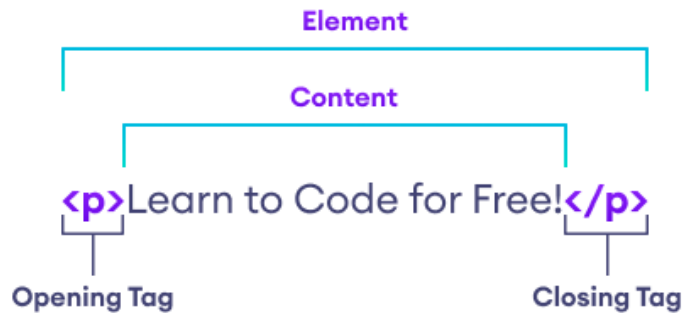
Let's see the meaning of the various elements used in the above example.

- `<html>`: the root element of the DOM, and it contains all of the other elements in the code
- `<head>`: contains metadata about the web page, such as the title and any linked CSS or JavaScript files
- `<title>`: contains the title of the web page, which will be displayed in the web browser's title bar or tab
- `<body>`: contains the main content of the web page, which will be displayed in the web browser's window
- `<p>`: contains the paragraphs of text on the web page
- ``, ``: child elements of the `<p>` elements, they are used to mark text as important and emphasized respectively

Note: Only the elements inside the `<body>` tag renders in the web browser.

What are HTML elements?

HTML elements consist of several parts, including the opening and closing tags, the content, and the attributes. Here is an explanation of each of these parts:



Here,

- **The opening tag:** This consists of the element name, wrapped in angle brackets. It indicates the start of the element and the point at which the element's effects begin.
- **The closing tag:** This is the same as the opening tag, but with a forward slash before the element name. It indicates the end of the element and the point at which the element's effects stop.
- **The content:** This is the content of the element, which can be text, other elements, or a combination of both.
- **The element:** The opening tag, the closing tag, and the content together make up the element.

What are HTML Attributes?

HTML elements can have attributes, which provide additional information about the element. They are specified in the opening tag of the element and take the form of name-value pairs. Let's see an example:

```
<a href="http://example.com"> Example </a>
```

The href is an attribute. It provides the link information about the <a> tag. In the above example,

- href - the name of attribute
- https://www.programiz.com - the value of attribute

Note: HTML attributes are mostly optional.

HTML Syntax

We need to follow a strict syntax guidelines to write valid HTML code. This includes the use of tags, elements, and attributes, as well as the correct use of indentation and white space. Here are some key points about HTML syntax:

1. HTML tags consist of the element name, wrapped in angle brackets. For example, <h1>, <p>, are some HTML tags.

2. HTML elements are created by enclosing the content of the element inside the opening and closing tags of the element. For example,

```
<span>Some text.</span>
```

is an HTML element.

3. HTML attributes are used to provide additional information about HTML elements and are specified in the opening tag of the element. For example,

```
<a target="www.google.com">Click Here</a>
```

Here, target is an attribute.

4. HTML code should be well-formed and properly indented, with each element on its own line and each level of hierarchy indented by one level. This makes the code easier to read and understand, and can help to avoid errors. For example,

```
<html>
  <head>
    <title>My First HTML Page</title>
  </head>
  <body>
    <h1>My First HTML Page</h1>
    <p> Hello World!</p>
  </body>
</html>
```

HTML Paragraphs

The HTML <p> tag is used to create paragraphs. For example,

```
<p>HTML is fun to learn.</p>
```

Browser Output



HTML is fun to learn.

As we can see, a paragraph starts with the `<p>` and ends with the `</p>` tag.

HTML paragraphs always start on a new line. To demonstrate this, let's create a webpage with two paragraphs.

```
<p>Paragraph 1: Short Paragraph</p>
```

```
<p>Paragraph 2: Long Paragraph, this is a long paragraph with more text to fill an entire line in the website.</p>
```

Browser Output



Paragraph 1: Short Paragraph

Paragraph 2: Long Paragraph, this is a long paragraph with more text to fill an entire line in the website.

The above HTML code contains two paragraphs. And each paragraph starts on a new line even though there is sufficient space after the first paragraph.

Note: By default, browsers automatically add whitespace (margin) above and below paragraphs. It is possible to change whitespace and other design aspects using CSS.

HTML Paragraphs and Spaces

Paragraphs automatically remove extra spaces and lines from our text. For example,

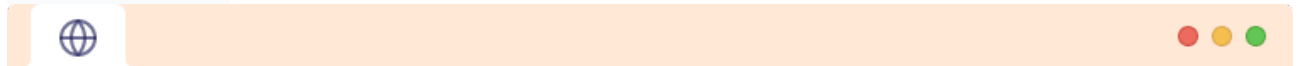
```
<p>
```

The paragraph tag removes all extra spaces.

The paragraph tag also removes all extra lines.

```
</p>
```

Browser Output



The paragraph tag removes all extra spaces. The paragraph tag also removes all extra lines.

Here, the output

- remove all the extra spaces between words
- remove extra lines between sentences

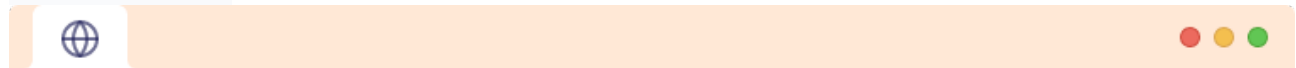
Note: It's possible to add extra spaces and lines in HTML, which we will discuss later in this tutorial.

Adding Line Breaks in Paragraphs

We can use the *HTML line break tag*, `
`, to add line breaks within our paragraphs. For example,

```
<p>We can use the <br> HTML br tag <br> to add a line break.</p>
```

Browser Output



We can use the
HTML `br` tag
to add a line break without creating a new paragraph

Note: The `
` tag does not need a closing tag like the `<p>` tag.

Pre-formatted Text in HTML

As we have discussed, paragraphs cannot preserve extra lines and space. If we need to create content that uses multiple spaces and lines, we can use the *HTML* `<pre>` tag.

The `<pre>` tag creates preformatted text. Preformatted texts are displayed as written in the HTML file. For example,

```
<pre>
This text
will be
displayed
```

```
in the same manner
as it is written
```

```
</pre>
```

Browser Output



This text
will be
displayed

in the same manner
as it is written

Other Elements Inside Paragraphs

It's possible to include one HTML element inside another. This is also true for `<p>` tags. For example,

```
<p>
```

We can use other tags like ``the strong tag to emphasize text``

```
</p>
```

Browser Output



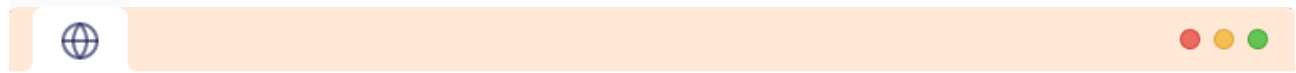
We can use other tags like **the strong tag to emphasize text**

In the above example, we have used the `` tag inside the `<p>` tag.

Browsers automatically make the contents inside `` tags bold.

Paragraph is Block-level

The `<p>` tag is a block-level element. It starts on a new line and takes up the full width (of its parent element).



The `<p>` element is a block-level element

Note: Keep note of which elements are inline-level and which are block-level. It will be important later when we learn CSS.

Add Extra Space Inside Paragraphs

As discussed earlier, we cannot normally add extra empty spaces inside paragraphs. However, we can use a certain HTML entity called **non-breaking space** to add extra spaces. For example,

```
<p>Extra space &nbsp;&nbsp;&nbsp; inside a paragraph</p>
```

Browser Output



Extra space inside a paragraph

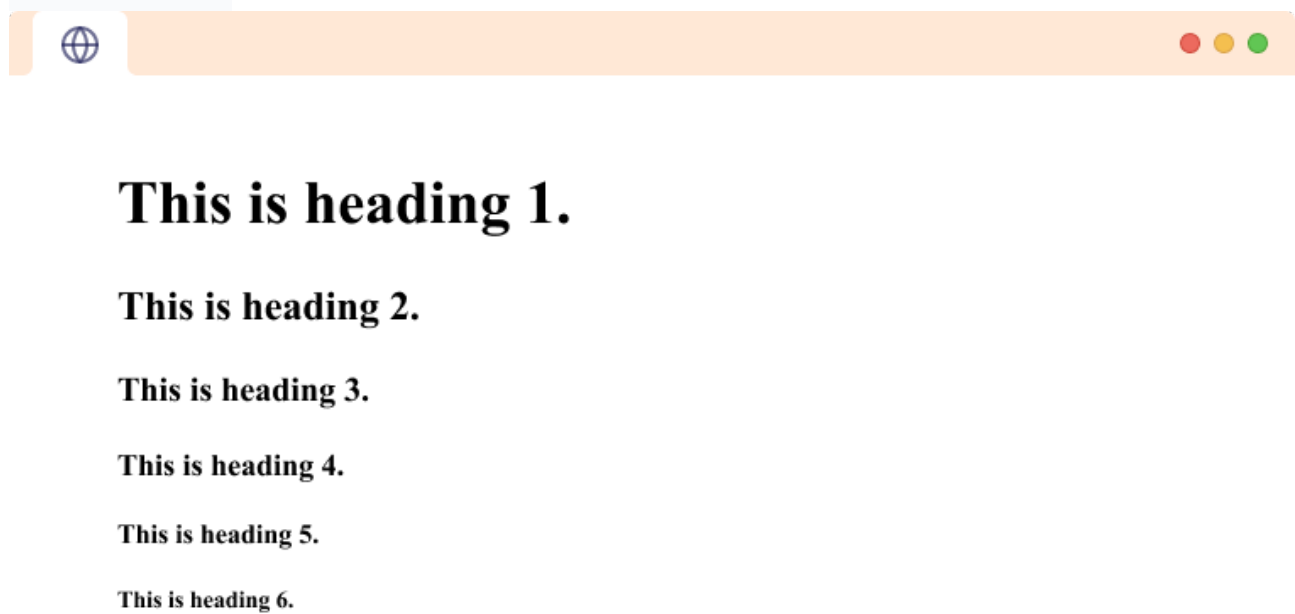
Here, ` ` is an *HTML entity*, which is interpreted as a space by browsers. This allows us to create multiple spaces inside paragraphs and other HTML tags.

HTML Headings

The HTML heading tags (`<h1>` to `<h6>`) are used to add headings to a webpage. For example,

```
<h1>This is heading 1.</h1>
<h2>This is heading 2.</h2>
<h3>This is heading 3.</h3>
<h4>This is heading 4.</h4>
<h5>This is heading 5.</h5>
<h6>This is heading 6.</h6>
```

Browser Output



In the example, we have used tags `<h1>` to `<h6>` to create headings of varying sizes and importance.

The `<h1>` tag denotes the most important heading on a webpage. Similarly, `<h6>` denotes the least important heading.

The difference in sizes of heading tags comes from the browser's default styling. And, you can always change the styling of heading tags, including font size, using CSS.

Note: Do not use heading tags to create large texts. It's because search engines like Google use heading tags to understand what a page is about.

h1 Tag is Important

The HTML `<h1>` tag defines the most important heading in a webpage.

Although it's possible to include multiple `<h1>` tags in a webpage, the general practice is to use a single `<h1>` tag (usually at the beginning of the page).

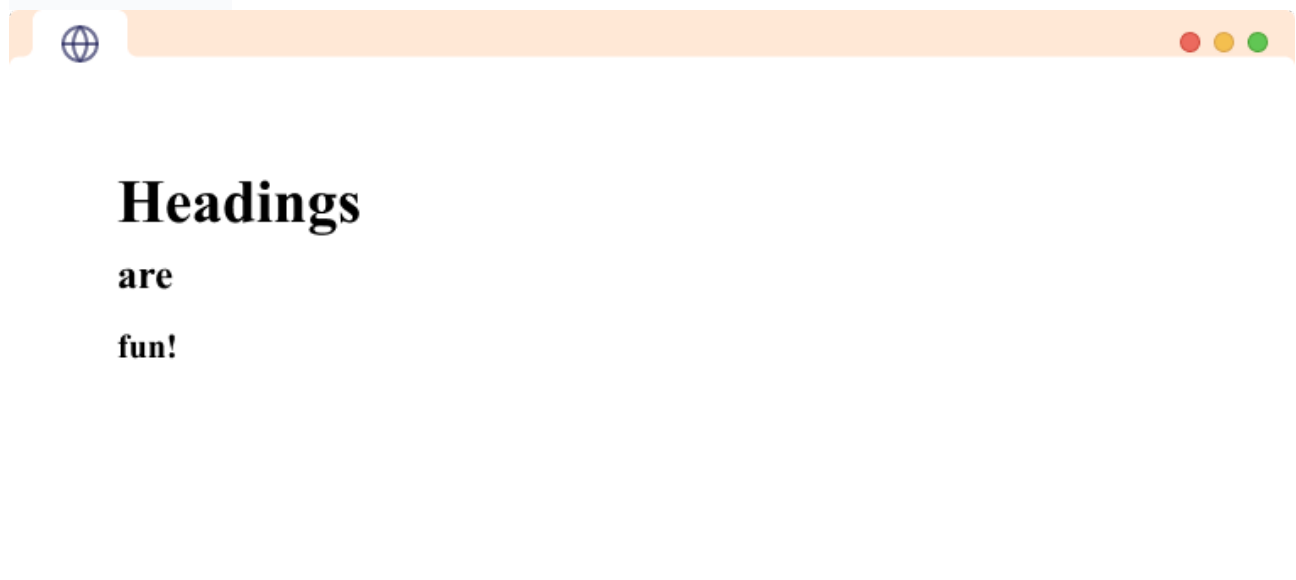
The `<h1>` tag is also important for SEO. Search engines (such as Google) treat content inside `<h1>` with greater importance compared to other tags.

Headings are block-level elements

The Headings `<h1>` to `<h6>` tags are block-level elements. They start on a new line and take up the full width of their parent element. For example,

```
<h1>Headings</h1> <h2>are</h2> <h3>fun!</h3>
```

Browser Output



HTML Comments

HTML comments are used to insert notes to a web page. For example,

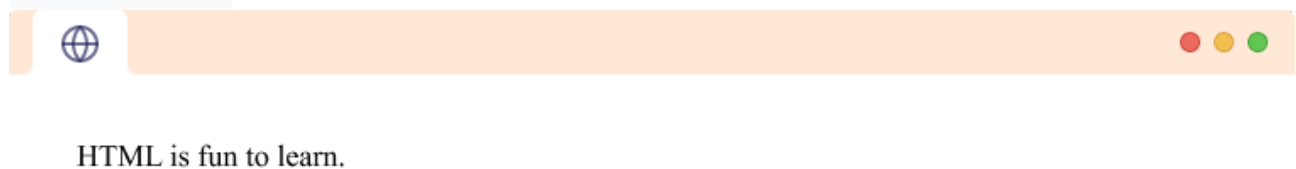
```
<!-- heading 2 -->  
<h2>Comments in HTML</h2>
```

Here, <!-- heading 2 --> is a comment. In HTML, comments start with <!-- and ends with -->

HTML comments are not displayed by browsers. They are used to add notes for documentation purposes within code. For example,

```
<!-- an HTML div -->  
<div>  
  <p>HTML is fun to learn.</p>  
</div>
```

Browser Output



Here,

```
<!-- an HTML div -->
```

is a comment. Hence, it is not displayed in the browser output.

Why use HTML Comments?

Comments are mainly used to make our code more readable. They are completely hidden from the webpage and only show up in the code.

Commenting on your code is a good practice as it helps us to express what the code is doing. It can act as an anchor for you if you want to change your code in the future.

In a collaborative environment, code comments are helpful for other developers as well.

Note: Even though browsers don't display comments, it's still possible to view comments using the browser's **View Source** feature. That's why we must not add sensitive information inside comments.

Single-Line and Multi-line Comments

In HTML, we use the same syntax to create single-line and multi-line comments.

Single Line Comment

```
<!-- Write a heading -->  
<h1>Important Heading</h1>
```

Multi-line Comments

```
<!-- Multiple Line comments  
can include line breaks  
and also extra   spaces -->
```

```
<p>this will display in the webpage</p>
```

HTML Tags Inside Comments

If we put HTML elements inside comments, they will be ignored. For example,

```
<p>this will be displayed</p>
```

```
<!-- <p>this will not be displayed</p> -->
```

```
<p>this will also be displayed</p>
```

Browser Output



this will be displayed

this will also be displayed

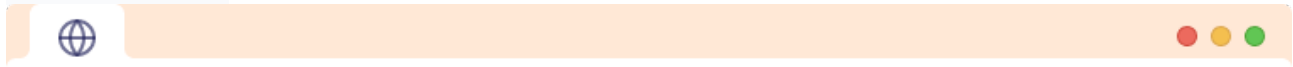
HTML Table

The HTML table tag (<table>) is used to represent data in a structured way by creating a table. For example,

```
<table border="1" >
<tr>
<th>Name</th>
<th>Age</th>
<th>Country</th>
</tr>
<tr>
<td>Harry Depp</td>
<td>28</td>
<td>Britain</td>
</tr>
<tr>
<td>John Smith</td>
<td>35</td>
<td>USA</td>
</tr>
<tr>
<td>Ram Krishna</td>
<td>19</td>
<td>Nepal</td>
```

```
</tr>
</table>
```

Browser Output



Name	Age	Country
Harry Depp	28	Britain
John Smith	35	USA
Ram Krishna	19	Nepal

In the above example, you can see we have used multiple tags to create a table in HTML.

- `<table>`
- `<tr>`
- `<td>`
- `<th>`

Table tag `<table>` in HTML

The `<table>` tag is used to define a table. For example,

```
<table>
...
</table>
```

Table Row `<tr>` in HTML

The `<tr>` tag is used to define a row in a table. For example,

```
<table>
<tr>
...
</tr>
</table>
```

The table row can include either table heading, `<th>` or table data, `<td>`.

```

<tr>
  <th>Name</th>
  <th>Country</th>
</tr>
<tr>
  <td>Prasanna</td>
  <td>Nepal</td>
</tr>
<tr>
  <td>Simon</td>
  <td>USA</td>
</tr>

```

In a table, there can be any number of rows.

Table Heading, <th> in HTML

The <th> tag is used to define a table header. It is generally the top row of the table. For example,

```

<table>
  <tr>
    <th>Item</th>
    <th>Count</th>
  </tr>
  <tr>
    <td>Mango</td>
    <td>125</td>
  </tr>
  <tr>
    <td>Orange</td>
    <td>75</td>
  </tr>
</table>

```

Browser Output



Item	Count
Mango	125
Orange	75

In the above example, Item and Count are table headers and they are used to represent the category of data in a particular row.

Here, the styling of the table headers is bold and center-aligned. This is because the `<th>` tag has some default styling.

Table Cell `<td>` in HTML

The `<td>` tag is used to define table cells (data). The table cells store data to be displayed in the table. For example,

```
<tr>
  <td>Apple</td>
  <td>Mango</td>
  <td>Orange</td>
</tr>
```

In the above example, `<td>Apple</td>`, `<td>Mango</td>` and `<td>Orange</td>` are table cells.

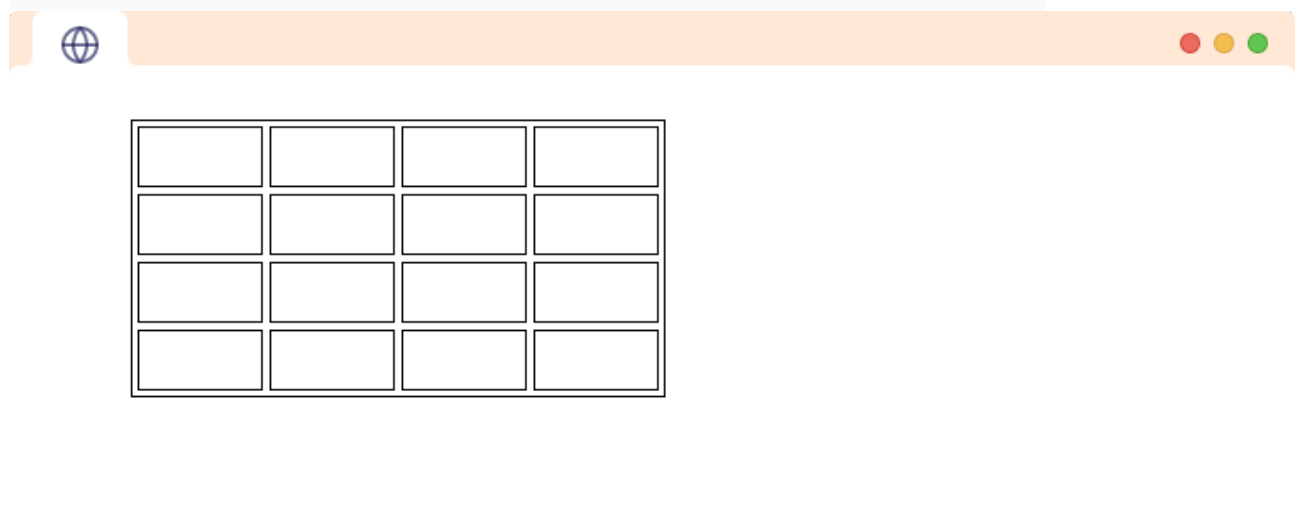
Table cells are generally inside the table row or table headers.

Table Border

Remember we have used the border attribute in our first example.

```
<table border="1">
  ...
</table>
```

In HTML, the **border** attribute is used to add a border to a table and all the cells.



Note: We can have borders of various styles in tables, however for more specific borders, we need to use CSS.

To prevent double borders like the one in the example above, we can set the border-collapse property of the table. For example,

```
<table border="1" style="border-collapse: collapse;">
...
</table>
```

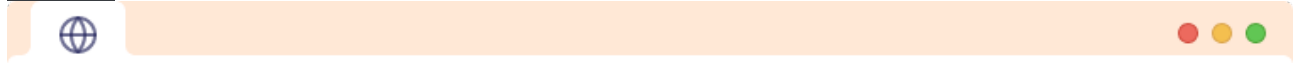


Table Head, Body, and Footer

The HTML table can be divided into three parts: a header, a body, and a footer.

1. Table Header

We use the <thead> tag to add a table head. The <thead> tag must come before any other tags inside a table. For example,

```
<table>
<thead>
  <tr>
    <th>Head1</th>
    <th>Head2</th>
  </tr>
</thead>
```

```
...
...
</table>
```

The content of <thead> is placed on the top part of the table and we usually place the rows with table headers inside the <thead> tag.

2. Table Body

We use the `<tbody>` tag to add a table body. The `<tbody>` tag must come after `<thead>` and before any other tags inside a table. For example,

```
<table>
<thead>
...
</thead>
<tbody>
<tr>
  <td>Cell 1</td>
  <td>Cell 2</td>
</tr>
</tbody>
```

```
...
...
</table>
```

The content of `<tbody>` is placed on the center part of the table and we usually place the rows with the content we want to represent in the `<tbody>`.

3. Table Footer

We use the `<tfoot>` tag to add a table footer. The `<tfoot>` tag must come after `<tbody>` and before any other tags inside a table. For example,

```
<table>
<thead>
...
</thead>
<tbody>
...
</tbody>
<tfoot>
<tr>
  <td>foot 1</td>
  <td>foot 2</td>
</tr>
</tfoot>
</table>
```

The content of `<tbody>` is placed on the bottom part of the table and we usually place the rows with the footer in the `<tfoot>`.

All these tags must be placed inside a `<table>` tag and must contain at least one `<tr>`. For example,

Example: HTML Table Head, Body, and Footer

```
<table>
<thead>
  <tr>
    <th>S.N</th>
    <th>Item</th>
    <th>Quantity</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>1</td>
    <td>Apple</td>
    <td>2</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Mango</td>
    <td>2</td>
  </tr>
  <tr>
    <td>3</td>
    <td>Orange</td>
    <td>1</td>
  </tr>
</tbody>
<tfoot>
  <tr>
    <td></td>
    <td>Total</td>
    <td>5</td>
  </tr>
</tfoot>
</table>
```

Browser Output



S.N	Item	Quantity
1	Apple	2
2	Mango	2
3	Orange	1
	Total	5

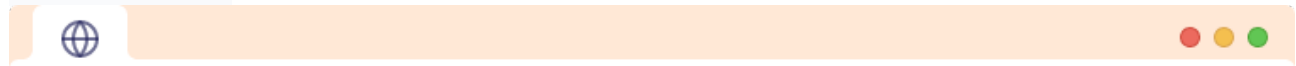
Colspan and Rowspan

Colspan

The colspan attribute merges cells across multiple columns. For example,

```
<table>
<tr>
  <th>S.N</th>
  <th>Item</th>
  <th>Quantity</th>
</tr>
<tr>
  <td>1</td>
  <td>Apple</td>
  <td>2</td>
</tr>
<tr>
  <td>2</td>
  <td>Mango</td>
  <td>2</td>
</tr>
<tr>
  <td>3</td>
  <td>Orange</td>
  <td>1</td>
</tr>
<tr>
  <td colspan="2">Total</td>
  <td>5</td>
</tr>
</table>
```

Browser Output



S.N	Item	Quantity
1	Apple	2
2	Mango	2
3	Orange	1
Total		5

In the above example, you can see that the last row only has 2 cells with one cell occupying 2 columns.

The value of the colspan attribute determines how many columns the cell occupies.

Rowspan

The rowspan attribute merges cells across multiple rows. For example,

```
<table>
<tr>
  <th>Name</th>
  <th>Subject</th>
  <th>Marks</th>
</tr>
<tr>
  <td rowspan="3">67<Mark Smith</td>
  <td>English</td>
</tr>
<tr>
  <td>Maths</td>
  <td>82</td>
</tr>
<tr>
  <td>Science</td>
  <td>91</td>
</tr>
</table>
```

Browser Output



Name	Subject	Marks
Mark Smith	English	67
	Maths	82
	Science	91

In the above example, you can see that the first column only has 2 cells with one cell occupying 2 rows.

The value of the rowspan attribute determines how many rows the cell occupies.

HTML Lists

HTML lists are used to display related information in an easy-to-read and concise way as lists.



- | | | |
|-------------|-------------|-------------------------|
| • List item | 1. 1st item | List item title |
| • List item | 2. 2nd item | - list item description |
| • List item | 3. 3rd item | List item title |
| • List item | 4. 4th item | - list item description |
| | | List item title |
| | | - list item description |

We can use three types of lists to represent different types of data in HTML:

1. Unordered List
2. Ordered List
3. Description List <dl>

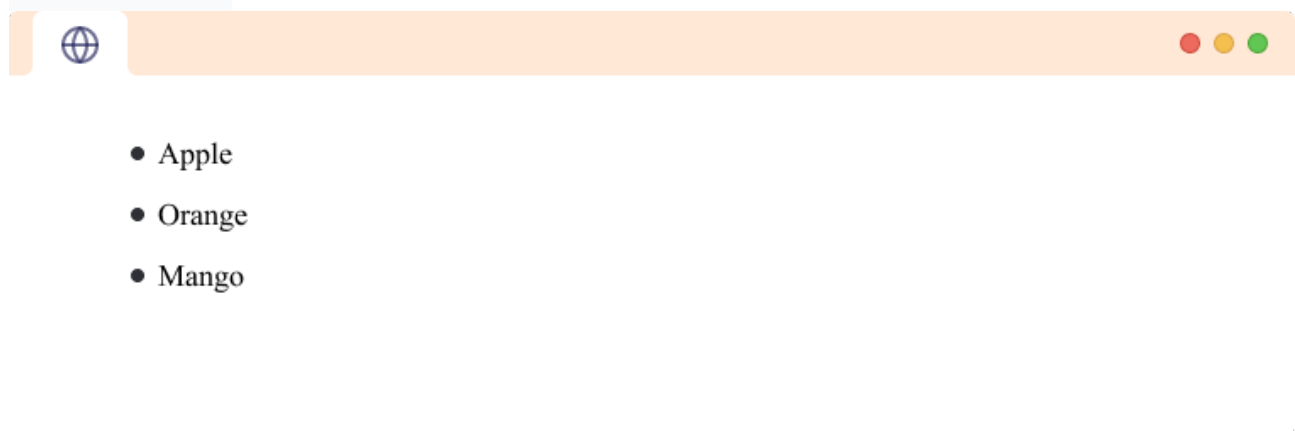
Unordered List

The unordered list is used to represent data in a list for which the order of items does not matter.

In HTML, we use the `` tag to create unordered lists. Each item of the list must be a `` tag which represents list items. For example,

```
<ul>
<li>Apple</li>
<li>Orange</li>
<li>Mango</li>
</ul>
```

Browser Output



Here, `Apple`, `Orange`, and `Mango` are the list items.

To learn more about unordered lists, visit [HTML Unordered Lists](#).

Ordered List

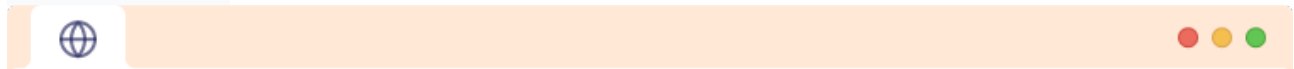
The ordered list is used to represent data in a list for which the order of items has significance.

The `` tag is used to create ordered lists. Similar to unordered lists, each item in the ordered list must be a `` tag. For example,

```
<ol>
<li>Ready</li>
<li>Set</li>
<li>Go</li>
```

```
</ol>
```

Browser Output



1. Ready
2. Set
3. Go

Here, you can see list items are represented with numbers to represent a certain order.

To learn more about ordered lists, visit [HTML Ordered Lists](#).

Description List

The HTML description list is used to represent data in the name-value form. We use the `<dl>` tag to create a definition list and each item of the description list has two elements:

- **term/title** - represented by the `<dt>` tag
- **description of the term** - represented by the `<dd>` tag

Let's see an example,

```
<dl>
<dt>HTML</dt>
<dd>Hyper-Text Markup Language</dd>
<dt>CSS</dt>
<dd>Cascading StyleSheets</dd>
<dt>JS</dt>
<dd>Javascript</dd>
</dl>
```

Browser Output



HTML

Hyper-Text Markup Language

CSS

Cascading StyleSheets

JS

Javascript

HTML Unordered List

We use the HTML unordered list to define a list where the sequence or order of the list items doesn't matter. We can use an unordered list for keeping track of groceries, supplies and random objects.

In HTML, we use the `` tag to create an unordered list. For example,

```
<ul>
<li>Apple</li>
<li>Mango</li>
<li>Orange</li>
<li>Banana</li>
</ul>
```

Browser Output





- Apple
- Mango
- Orange
- Banana

Each item of the list is enclosed inside the `` tag and they are represented by the dot bullet point symbol.

By default, the symbol to represent an unordered list in HTML is a dot bullet point, however, we can change them as per our choice.

Below, we can see examples for all the marker types.

				
Dot	Circle	Square	None	
● Apple	○ Apple	■ Apple	Apple	
● Mango	○ Mango	■ Mango	Mango	
● Orange	○ Orange	■ Orange	Orange	
● Banana	○ Banana	■ Banana	Banana	

Nesting Lists

In HTML, we can create a nested list by adding one list inside another. For example,

```
<ul>
  <li>
    Coffee
    <ul>
      <li>Cappuccino</li>
      <li>Americano</li>
      <li>Espresso</li>
    </ul>
  </li>
  <li>
    Tea
    <ul>
      <li>Milk Tea</li>
      <li>Black Tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Browser Output



- Coffee
 - Cappuccino
 - Americano
 - Espresso
- Tea
 - Milk Tea
 - Black Tea
- Milk

In the above example, you can see we have added unordered lists inside another unordered list.

In this case, the first and second list items of the outer unordered list include unordered lists.

Ordered List inside Unordered List

Similarly, we can also mix list types while nesting and add ordered lists inside the unordered list. For example,

```
<ul>
  <li>
    Coffee
    <ol>
      <li>Cappuccino</li>
      <li>Americano</li>
      <li>Espresso</li>
    </ol>
  </li>
  <li>
    Tea
    <ol>
      <li>Milk Tea</li>
      <li>Black Tea</li>
    </ol>
  </li>
  <li>Milk</li>
</ul>
```

Browser Output



- Coffee
 - 1. Cappuccino
 - 2. Americano
 - 3. Espresso
- Tea
 - 1. Milk Tea
 - 2. Black Tea
- Milk

Multi-level Nesting of Unordered List

In our examples, we are nesting the list up to a single level, however, we can also nest lists up to multiple levels. This might be important while creating lists like Table of Content. For example,



I. Background Skills

A. Unix Commands

B. Vim Text Editors

II. HTML

A. Minimal Page

B. Headings

C. Elements

D. Lists

i. Minimal Page

ii. Headings

iii. Elements

iv. Lists

E. Links

i. Absolute

ii. Relative

F. Elements

III. CSS

A. Anatomy

B. Basic Selection

i. Element

ii. Class

iii. ID

iv. Group

C. The DOM

D. Advanced Selectors

E. Box Model

IV. Programming

A. Python

B. JavaScript

V. Database

A. Flat File

B. Relational

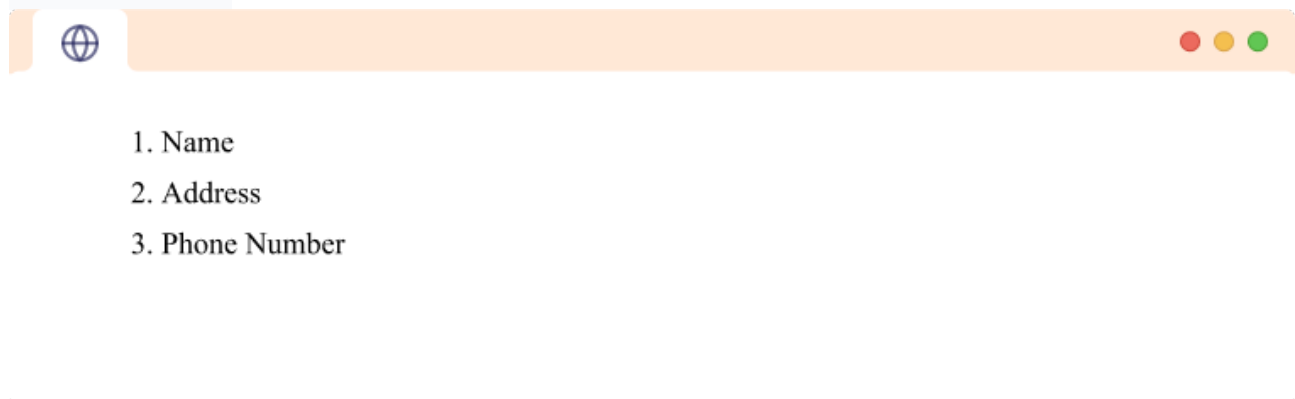
HTML Ordered List

We use the HTML ordered list to define a list where the sequence or order of the list items is important. We can use the HTML ordered list for recipes, algorithms, top ten lists, and so on.

We use the `` tag to create an unordered list. For example,

```
<ol>  
<li>Name</li>  
<li>Address</li>  
<li>Phone Number</li>  
</ol>
```

Browser Output



Each item of the list is enclosed inside the `` tag and they are numbered by decimal numbers.

By default, ordered lists are ordered by numbers, however, we can change them as per our choice.

Ordered Lists Type

We use the **type** attribute to change the marker for the list. There are five types of numbering in the ordered list. They are

Below, we can see examples for all the number types.



type = "1"

1. Name
2. Address
3. Phone Number

type = "a"

- a. Name
- b. Address
- c. Phone Number

type = "A"

- A. Name
- B. Address
- C. Phone Number

type = "i"

- i. Name
- ii. Address
- iii. Phone Number

type = "I"

- I. Name
- II. Address
- III. Phone Number

start Attribute

We use the start attribute to change the starting point for the numbering of the list. For example,

```
<ol start='5'>
  <li>Harry</li>
  <li>Ron</li>
  <li>Sam</li>
</ol>
```

Browser Output



5. Harry
6. Ron
7. Sam

Here, we change the starting value of the list to 5.

This attribute also works with other types. For example,

```
<ol type="a" start='5'>
  <li>Harry</li>
  <li>Ron</li>
  <li>Sam</li>
```

```
</ol>
```

Browser Output



- e. Harry
- f. Ron
- g. Sam

Similarly, we can use the start attribute along with all other types.

reversed Attribute

We can use the reversed attribute on the ordered list to reverse the numbering on the list. For example,

```
<ol reversed>  
<li>Cat</li>  
<li>Dog</li>  
<li>Elephant</li>  
<li>Fish</li>  
</ol>
```

Browser Output



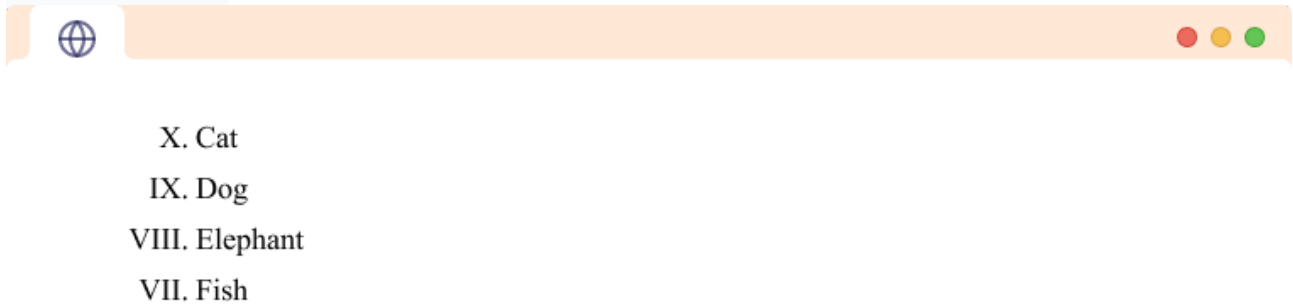
4. Cat
3. Dog
2. Elephant
1. Fish

Here, we can see the order of the list is reversed, the first list item is numbered 4 and the last is numbered 1.

Similarly, the reversed attribute can also be used with other types and in conjunction with the start attribute. For example,

```
<ol reversed type="I" start="10">
  <li>Cat</li>
  <li>Dog</li>
  <li>Elephant</li>
  <li>Fish</li>
</ol>
```

Browser Output



In the above example, we use the upper-case roman numeral type and start at 10 and reverse the order of the numbers.

Nesting Lists

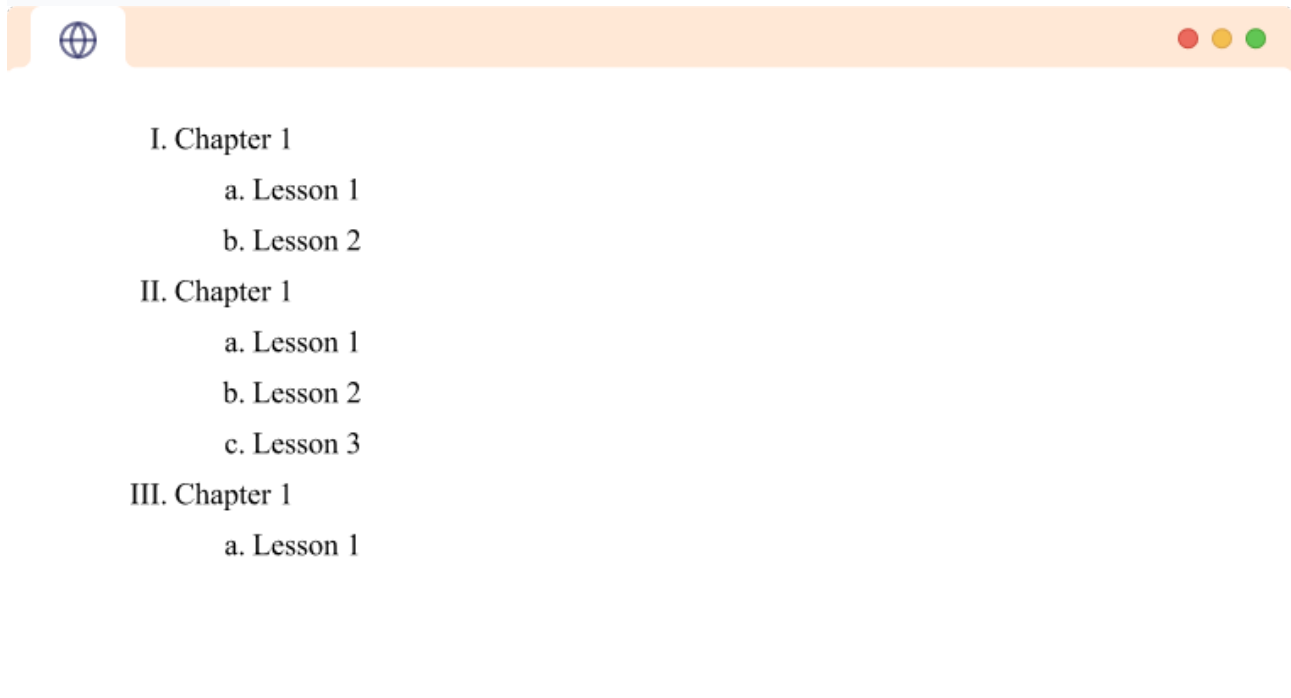
In HTML, we can create a nested list by adding one list inside another. For example,

```
<ol type="I">
  <li>
    Chapter 1
    <ol type="a">
      <li>Lesson 1</li>
      <li>Lesson 2</li>
    </ol>
  </li>
  <li>
    Chapter 2
    <ol type="a">
      <li>Lesson 1</li>
      <li>Lesson 2</li>
      <li>Lesson 3</li>
    </ol>
  </li>
  <li>
    Chapter 3
```



```
<ol type="a">  
  <li>Lesson 1</li>  
</ol>  
</li>  
</ol>
```

Browser Output



In the above example, you can see we have added an ordered list inside another ordered list.

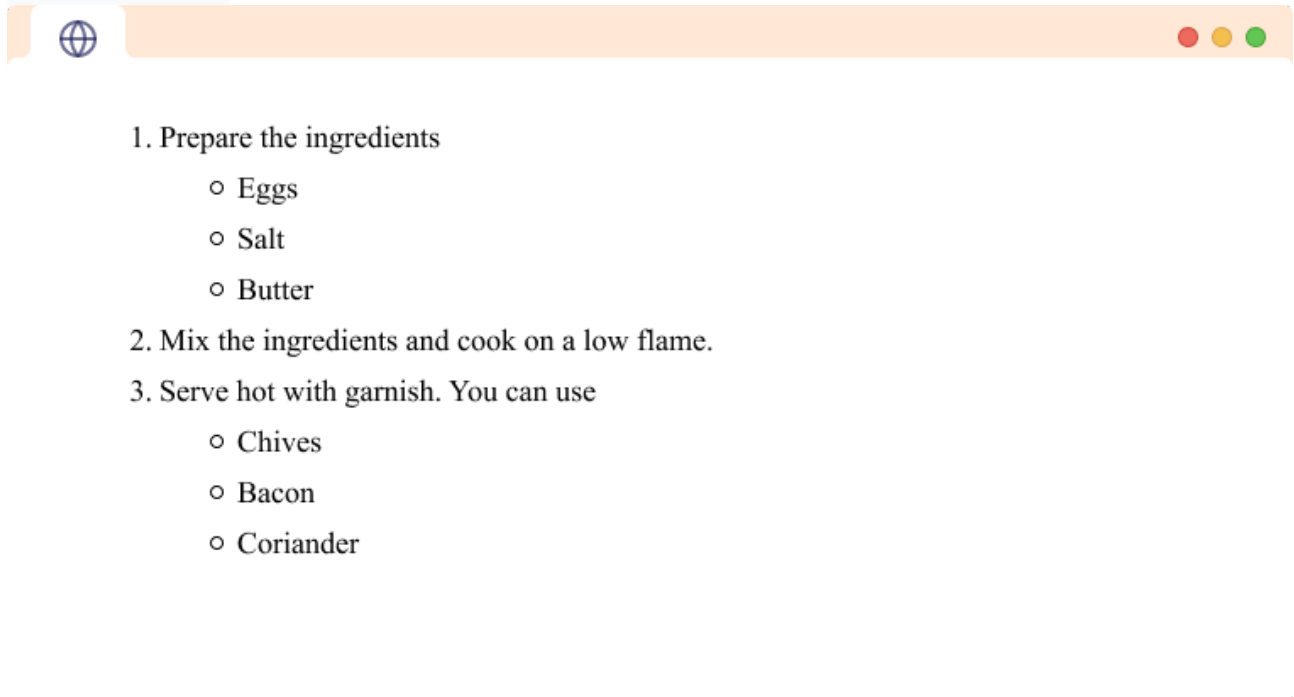
In this case, the list item of the outer ordered list also includes an ordered list.

Similarly, we can also mix list types while nesting and add an unordered list inside the ordered list. For example,

```
<ol>  
  <li>  
    Prepare the ingredients.  
    <ul>  
      <li>Eggs</li>  
      <li>Salt</li>  
      <li>Butter</li>  
    </ul>  
  </li>  
  <li>  
    Mix the ingredients and cook on a low flame.  
  </li>  
  <li>  
    Serve hot with garnish. You can use
```

```
<ul>
  <li>Chives</li>
  <li>Bacon</li>
  <li>Coriander</li>
</ul>
</li>
</ol>
```

Browser Output



HTML Description List

We use the HTML description list to create a list where list items include terms and descriptions of the term.

In HTML, we use the `<dl>` tag to create a description list. For example,

```
<dl>
  <dt>HTML</dt>
  <dd>HyperText Markup Language</dd>
  <dt>CSS</dt>
  <dd>Cascading Style Sheets</dd>
</dl>
```

Browser Output



HTML

HyperText Markup Language

CSS

Cascading Style Sheets

1. Single Term and Description

```
<dl>  
<dt>HTML</dt>  
<dd>HyperText Markup Language</dd>  
<dt>CSS</dt>  
<dd>Cascading Style Sheets</dd>  
</dl>
```

Browser Output



HTML

HyperText Markup Language

CSS

Cascading Style Sheets

Here, we can see a single term is followed by a single description.

2. Single Term and Multiple Description

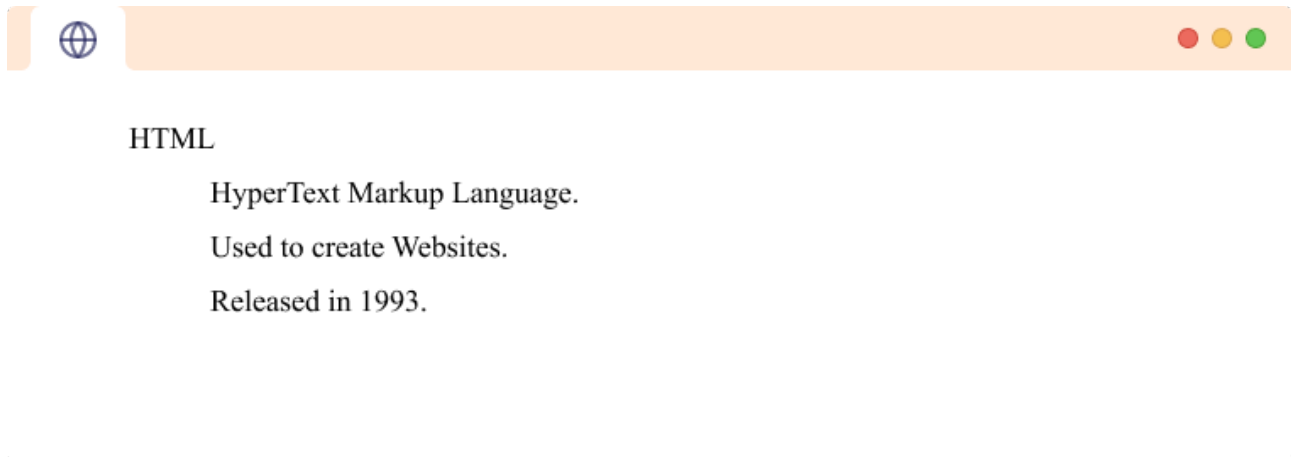
Sometimes, we can come across data where there are multiple descriptors that fit the same term, like a product and its features list.

In such a case, rather than listing the product name multiple times before each feature, we can follow the single term `<dt>` with multiple feature/description `<dd>` to present it better. For example,

```
<dl>
```

```
<dt>HTML</dt>
<dd>HyperText Markup Language.</dd>
<dd>Used to create Websites.</dd>
<dd>Released in 1993.</dd>
</dl>
```

Browser Output



Here, we can see that the single term `<dt>HTML</dt>` is described by multiple definitions `<dd>`

3. Multiple Term and Single Description

Sometimes, we come across data where multiple keys may have similar values. Like multiple programming languages can feature the same feature sets.

In such cases, rather than repeating the key/value pair, we can group several keys `<dt>` followed by a single description `<dd>` such that the single description describes many terms,

```
<dl>
<dt>HTML</dt>
<dd>is markup language</dd>
<dt>Python</dt>
<dt>Java</dt>
<dd>are programming languages.</dd>
</dl>
```

Browser Output



HTML

is markup language

Python

Java

are programming languages.

Here, we can see that multiple terms `<dt>Python</dt>` and `<dt>Java</dt>` share the same description `<dd>are programming languages.</dd>`.

HTML Line break

Line-Break vs Multiple Paragraphs

As we know, an *HTML paragraph* always starts on a new line, instead of a line break we can also use multiple paragraphs. For example,

```
<p>Use the</p>  
<p> br tag</p>  
<p> to create line breaks in text.</p>
```

Browser Output



Use the

br tag

to create line breaks in text.

We can see we have successfully applied line breaks using multiple paragraphs. However, there are several drawbacks of this approach:

- Screen readers treat each paragraph as separate sentences. That is, Use the, br tag, and to create line breaks in text. will be read as different sentences which will create confusion for users.
- Each paragraph has its own default styling, hence with multiple paragraphs, there will be unnecessary margin and padding between the lines making them not seem as continuous. You can see that in the above output as well.

Example: Line breaks using HTML `
` and `<p>`

```
<p>Roses are red</p>
<p>Violets are blue</p>
<p>HTML is Fun</p>
<p>and so are You!</p>
```

```
<p>
Roses are red <br>
Violets are blue <br>
HTML is fun <br>
and so are You!
</p>
```

Browser Output



Roses are red

Violets are blue

HTML is Fun

and so are You!

Roses are red

Violets are blue

HTML is Fun

and so are You!

HTML Horizontal Line

The HTML Horizontal rule tag, `<hr>`, is used to insert a horizontal line between our paragraphs. For example,

```
<p>First Paragraph</p>  
<hr>  
<p>Second Paragraph</p>
```

Browser Output



First Paragraph



Second Paragraph

The `<hr>` tag is an empty tag, i.e. it doesn't require a closing tag.

HTML horizontal line to separate Page sections

The <hr> tag is used to apply a thematic break in HTML. i.e It is used when there is a change of topic or you need to specify separation between content. Commonly, a <hr> tag is used between sections in an article. For example,

```
<h2>Physics</h2>
```

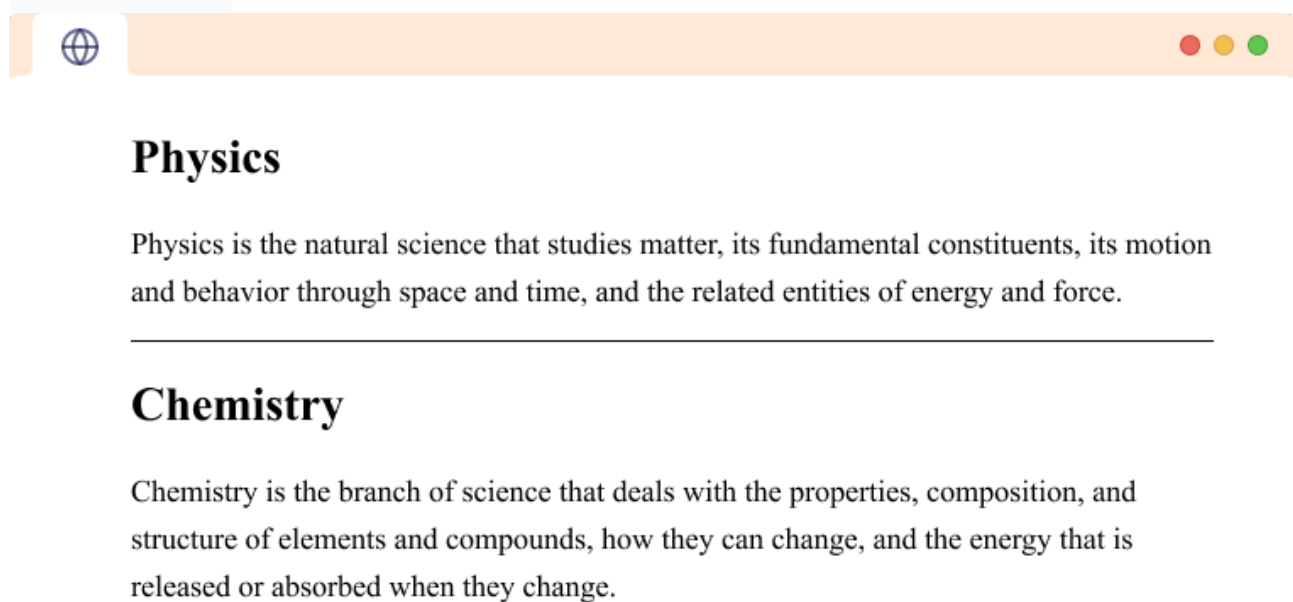
```
<p>Physics is the natural science that studies matter, its fundamental constituents, its motion and behavior through space and time, and the related entities of energy and force. </p>
```

```
<hr>
```

```
<h2>Chemistry</h2>
```

```
<p>Chemistry is the branch of science that deals with the properties, composition, and structure of elements and compounds, how they can change, and the energy that is released or absorbed when they change.</p>
```

Browser Output



HTML Form

An HTML Form is a section of the document that collects input from the user. The input from the user is generally sent to a server (Web servers, Mail clients, etc). We use the HTML <form> element to create forms in HTML.

Email

example@gmail.com

Password

Enter Your Password

☐

Remember Me

[Forgot Password?](#)

Sign In

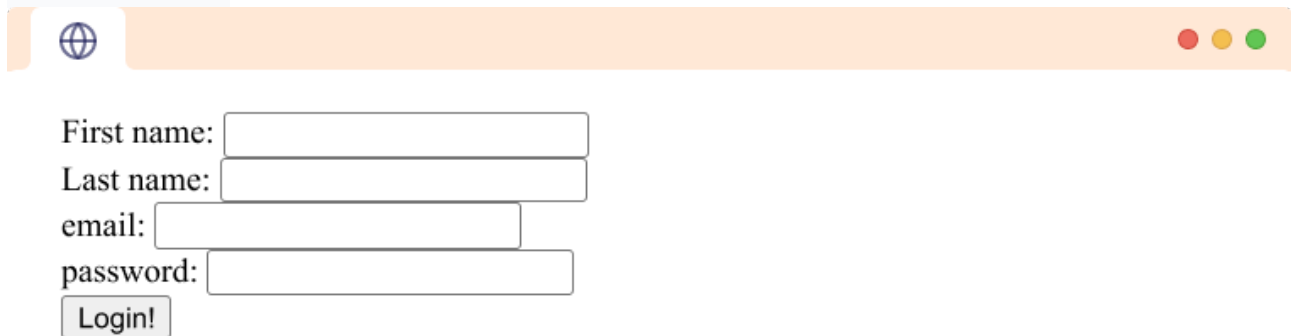
Example: HTML Form

The HTML `<form>` element is used to create HTML forms. For example,

```
<form>
  <label for="firstname">First name: </label>
  <input type="text" name="firstname" required>
  <br>
  <label for="lastname">Last name: </label>
  <input type="text" name="lastname" required>
  <br>
  <label for="email">email: </label>
  <input type="email" name="email" required>
  <br>
  <label for="password">password: </label>
  <input type="password" name="password" required>
  <br>
```

```
<input type="submit" value="Login!">
</form>
```

Browser Output

A screenshot of a web browser window. The browser has a light orange header bar with a globe icon on the left and three colored window control buttons (red, yellow, green) on the right. The main content area is white and contains a login form. The form consists of four text input fields stacked vertically, each preceded by a label: 'First name:', 'Last name:', 'email:', and 'password:'. Below the 'password:' field is a 'Login!' button with a grey background and black text.

HTML Form Elements

A form contains special interactive elements that users use to send the input. They are text inputs, textarea fields, checkboxes, dropdowns, and much more. For example,

```
<form>
  <label for="name">Name:</label>
  <input type="text" name="name"><br><br>
  <label for="gender">Gender:</label>
  <input type="radio" name="gender" id="male" value="male">
  <label for="male">Male</label>
  <input type="radio" name="gender" id="female" value="female">
  <label for="female">Female</label> <br><br>
  <label for="country">Country: </label>
  <select name="country" id="country">
    <option>Select an option</option>
    <option value="nepal">Nepal</option>
    <option value="usa">USA</option>
    <option value="australia">Australia</option>
  </select><br><br>
  <label for="message">Message:</label><br>
  <textarea name="message" id="message" cols="30" rows="4"></textarea><br><br>
  <input type="checkbox" name="newsletter" id="newsletter">
  <label for="newsletter">Subscribe?</label><br><br>
  <input type="submit" value="Submit">
</form>
```

Google Chrome File Edit View History Bookmarks Profiles Tab Window Help

Welcome to Form Section x +

File | /Users/shailendrakhare/Desktop/3rd%20semester/form.html?name=shubhi&gender=female&country=usa&message=i+am+a+girl

Name:

Gender: ☐ Male ☐ Female

Country:

Message:

☐ Subscribe?

Contact Information

HTML (Hypertext Markup Language) provides various form elements that allow you to create interactive forms on web pages. Forms are essential for gathering user input, such as text, checkboxes, radio buttons, and more. Here are some common HTML form elements:

1. ``<form>``: The ``<form>`` element is used to create a container for all the form elements. It specifies where the data entered into the form should be sent for processing and defines the method for submitting the form (usually "GET" or "POST").

```
```html
<form action="process.php" method="post">
 <!-- Form elements go here -->
</form>
```
```

2. ``<input>``: The ``<input>`` element is used to create various types of input fields, including text fields, password fields, radio buttons, checkboxes, and more.

```
```html
<input type="text" name="username">
<input type="password" name="password">
<input type="radio" name="gender" value="male"> Male
<input type="radio" name="gender" value="female"> Female
<input type="checkbox" name="subscribe" value="yes"> Subscribe to newslett```
```

3. ``<textarea>``: The ``<textarea>`` element is used to create a multi-line text input field, suitable for longer text entries.

```
```html
<textarea name="comments" rows="4" cols="50"></textarea>
```
```

4. ``<select>``: The ``<select>`` element is used to create a drop-down list of options. You can use ``<option>`` elements to define the available choices.

```
```html
<select name="country">
  <option value="usa">United States</option>
  <option value="canada">Canada</option>
  <option value="uk">United Kingdom</option>
</select>
```
```

5. ``<button>``: The ``<button>`` element is used to create a clickable button within a form.

```
```html
<button type="submit">Submit</button>
```
```

6. ``<label>``: The ``<label>`` element is used to provide a label for a form element. It helps improve accessibility and user experience by associating the label with its corresponding input element.

```
```html
<label for="username">Username:</label>
<input type="text" id="username" name="username">
```
```

7. ``<fieldset>`` and ``<legend>``: These elements are used to group related form controls and provide a title or description for the group. This helps organize and structure complex forms.

```
```html
<fieldset>
  <legend>Contact Information</legend>
  <!-- Form elements related to contact information -->
</fieldset>
```
```

8. ``<input type="submit">`` and ``<input type="reset">``: These input elements are used to create a "Submit" button and a "Reset" button, respectively.

```
```html
<input type="submit" value="Submit">
<input type="reset" value="Reset">
```
```

These are some of the most commonly used HTML form elements. By combining these elements and attributes, you can create a wide range of interactive forms for collecting user data on your web pages.

## Form Attributes

HTML form elements can have several attributes that allow you to control their behavior, appearance, and interaction with the user. Here are some common attributes used with HTML form elements:

1. ``action``: This attribute specifies the URL to which the form data will be submitted when the user submits the form.

```
```html<form action="process.php" method="post">

  <!-- Form elements go here -->

</form>

```
```

2. ``method``: The ``method`` attribute determines how the form data is sent to the server. It can be set to either "GET" or "POST."

- ``GET``: Appends form data to the URL as query parameters. This method is suitable for non-sensitive data.

- ``POST``: Sends form data in the request body. This method is more secure and is typically used for sensitive information.

```
```html
```

```
<form action="process.php" method="post">
```

```
<!-- Form elements go here -->
```

```
</form>
```

```
...
```

3. ``name``: The ``name`` attribute is used to give a name to the form element. It's crucial when processing the form data on the server.

```
```html
```

```
<input type="text" name="username">
```

```
...
```

4. ``id``: The ``id`` attribute provides a unique identifier for the form element, which can be useful for styling with CSS and associating labels with form controls.

```
```html
```

```
<input type="text" id="username" name="username">
```

```
<label for="username">Username:</label>
```

```
...
```

5. ``value``: The ``value`` attribute sets the initial or default value for input fields, checkboxes, and radio buttons.

```
```html
```

```
<input type="text" name="username" value="JohnDoe">
```

```
<input type="checkbox" name="subscribe" value="yes" checked>
```

```
...
```

6. `placeholder`: The `placeholder` attribute provides a hint or example of what should be entered into the input field. It disappears when the user starts typing.

```
```html
```

```
<input type="text" name="username" placeholder="Enter your username">
```

```
...
```

7. `disabled`: The `disabled` attribute disables a form element, preventing the user from interacting with it or submitting its value.

```
```html
```

```
<input type="text" name="disabledField" value="I am disabled" disabled>
```

```
...
```

8. `required`: The `required` attribute enforces that a user must fill out a form field before they can submit the form.

```
```html
```

```
<input type="text" name="requiredField" required>
```

```
...
```

9. `readonly`: The `readonly` attribute makes an input field or textarea uneditable, but the user can still see the content.

```
```html
```

```
<input type="text" name="readonlyField" value="I am readonly" readonly>
```

```
```
```

10. `min` and `max`: These attributes are used with number and date input types to define the minimum and maximum allowed values.

```
```html
```

```
<input type="number" name="age" min="0" max="100">
```

```
<input type="date" name="birthdate" min="2000-01-01" max="2022-12-31">
```

```
```
```

These are some of the common attributes used with HTML form elements. Each attribute serves a specific purpose and allows you to control the behavior and appearance of form elements according to your requirements.

HTML Input Tag

The HTML `<input>` tag defines the field where the user can enter data. The type attribute determines what type of user input is taken.

```
<input type="text" name="firstname">
```

Browser Output



Here,

- **type** - determines the type of input the `<input>` tag takes
- **name** - specifies the name of the input which is what the data is named as when submitting the data to a server.

Different Input Types

The various types of input tags available in HTML5 are:

1. **text** - creates a single-line text fields (default)
2. **button** - creates a button with no default functionality
3. **checkbox** - creates a checkbox
4. **color** - creates a color picker
5. **date** - creates a date picker
6. **datetime-local** - creates a date and time picker
7. **email** - creates an input field that allows the user to input a valid email address
8. **file** - creates an input field that lets the user upload a file or multiple files
9. **hidden** - creates an invisible input field
10. **image** - creates a button using an image
11. **month** - creates an input field that lets the user enter month and year
12. **password** - creates an input field that lets the user enter information securely
13. **radio** - creates a radio button
14. **range** - creates a range picker from which the user can select the value
15. **reset** - creates the button which clears all the form values to their default value
16. **search** - allows user to enter their search queries in the text fields
17. **submit** - allows user to submit form to the server

- 18. `tel` - defines the field to enter a telephone number
- 19. `time` - creates an input field that accepts time value
- 20. `url` - lets the user enter and edit a URL
- 21. `week` - lets the user pick a week and a year from a calendar

1. Input Type text

The input type `text` is used to create single-line text fields. It is the default input type.

```
<label for="name">Search: </label>  
<input type="text" id="name">
```

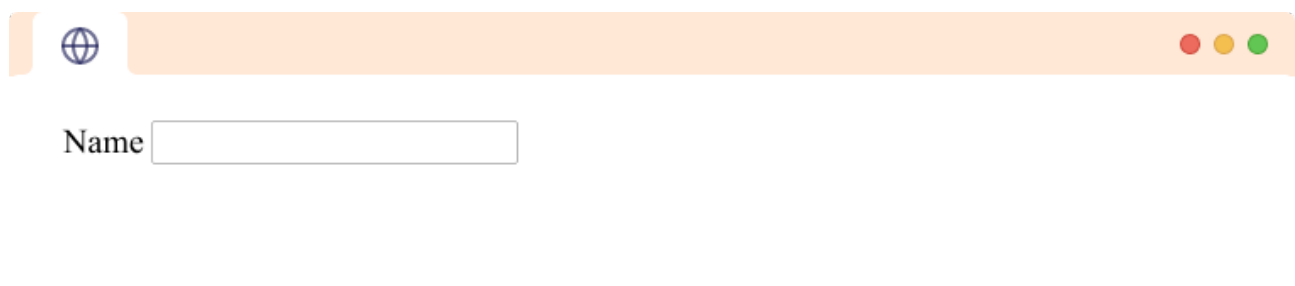
Browser Output

A screenshot of a web browser window with an orange header bar. The address bar contains a globe icon and three colored window control buttons (red, yellow, green). Below the header, the text "Search:" is followed by a single-line text input field.

The input type `text` can also contain `minlength`, `maxlength`, and `size` attributes. For example,

```
<label for="name">Name</label>  
<input type="text" id="name" minlength="4" maxlength="8">
```

Browser Output

A screenshot of a web browser window with an orange header bar. The address bar contains a globe icon and three colored window control buttons (red, yellow, green). Below the header, the text "Name" is followed by a single-line text input field.

In the above example, values are only allowed between the length of **4** to **8** characters.

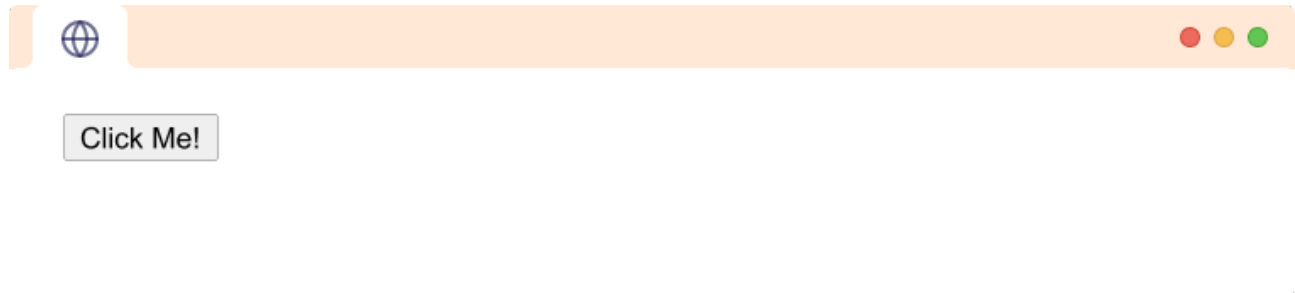
Note: If the `type` attribute is not provided, the tag becomes the type of text.

2. Input Type button

The input type `button` is used to create a button with no default functionality. For example,

```
<input type="button" value="Click Me!">
```

Browser Output



The text inside the value attribute is displayed in the button.

Note: Generally, javascript is used to add functionality to such buttons.

3. Input Type checkbox

The input type checkbox is used to create a checkbox input. For example,

```
<input type="checkbox" id="subscribe" value="subscribe">  
<label for="subscribe">Subscribe to newsletter!</label><br>
```

Browser Output (checkbox unselected)



The checkbox can be toggled between selected and not selected.

Browser Output (checkbox selected)



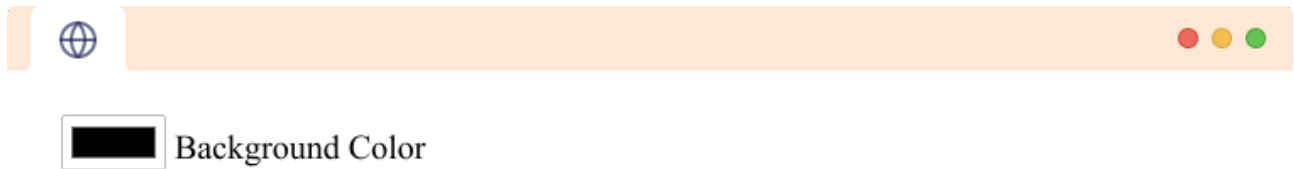
The value of the checkbox is included in the form data only if the checkbox is selected and is omitted if the checkbox is not selected.

4. Input Type color

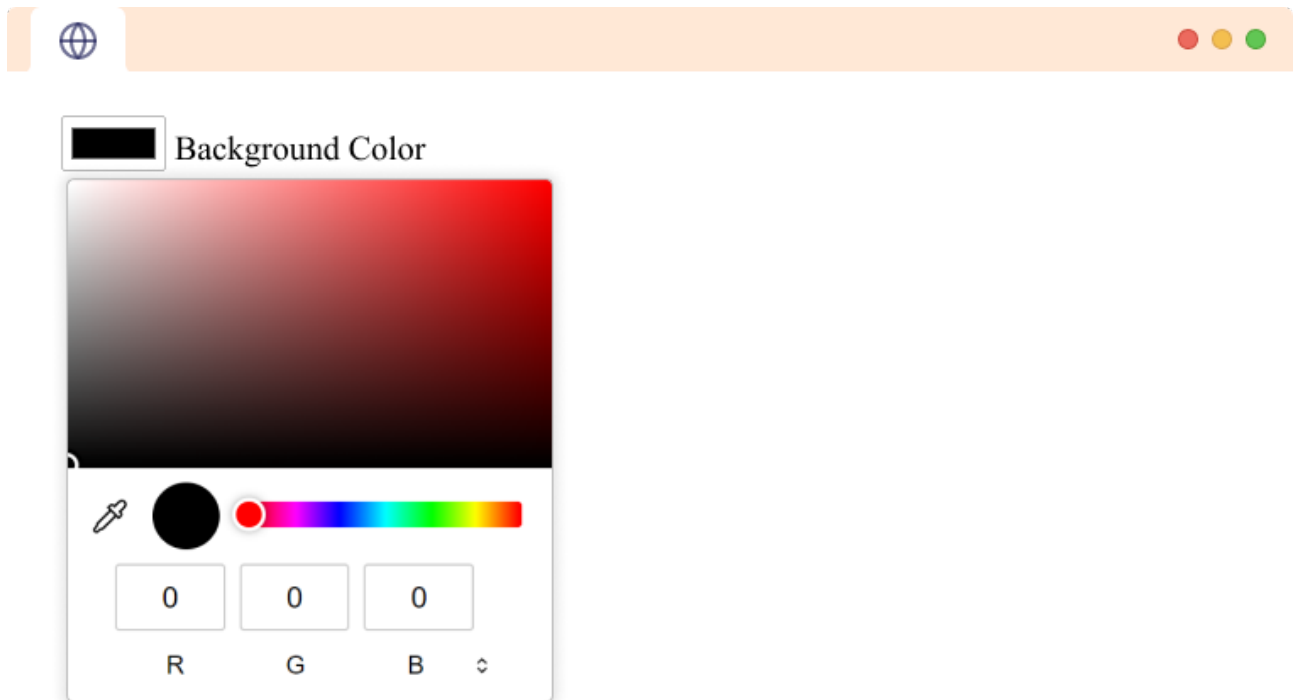
The input type `color` is used to create an input field that lets the user pick a color. For example,

```
<input type="color" id="background">  
<label for="background">Background Color</label>
```

Browser Output (before expanding)



Browser Output (after expanding)




The color picker is inbuilt into the browser. Users can also manually enter the hex code of the color instead. The UI for the color picker differs from browser to browser.

5. Input Type date

The input type `date` is used to create an input field that lets the user input a date using the date picker interface from the browser. For example,

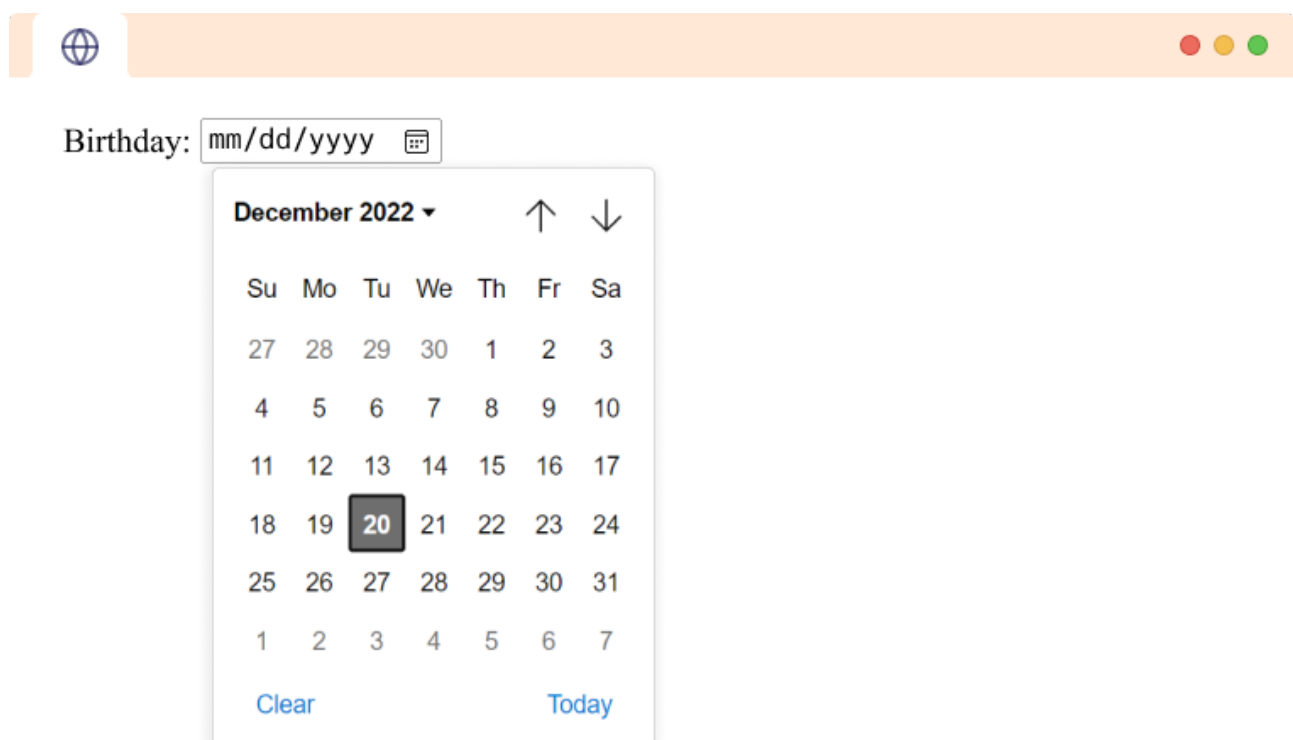
```
<label for="birthday">Birthday:</label>
<input type="date" id="birthday">
```

Browser Output (before expanding)



Birthday:

Browser Output (after expanding)



Birthday:

December 2022 ▾ ↑ ↓

Su	Mo	Tu	We	Th	Fr	Sa
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Clear Today

6. Input Type datetime-local

The input type `datetime-local` is used to create an input field that lets the user pick a date and time using a date-time-picker from the browser. The time selected from the input does not include information about the timezone. For example,

```
<label for="meeting-time">Choose a time for your appointment:</label>
<input type="datetime-local" id="meeting-time" >
```

Browser Output (before expanding)



Choose a time for your appointment:

Browser Output (after expanding)



Choose a time for your appointment:

December 2022 ▾

↑ ↓

05

39

PM

Su	Mo	Tu	We	Th	Fr	Sa		
							06	40 AM
27	28	29	30	1	2	3	07	41
4	5	6	7	8	9	10	08	42
11	12	13	14	15	16	17	09	43
18	19	20	21	22	23	24	10	44
25	26	27	28	29	30	31	11	45
1	2	3	4	5	6	7		

Clear

Today

7. Input Type email

The input type `email` is used to create an input field that allows the user to input a valid email address.

```
<label for="email">Enter your email:</label>
<input type="email" id="email">
```

Browser Output



A browser window mockup with an orange header bar containing a globe icon on the left and three colored window control buttons (red, yellow, green) on the right. Below the header, the text "Enter your email:" is followed by an empty text input field.

This input field throws an error if the value provided is not a valid email. For example,

Browser Output



A browser window mockup showing the email input field with the text "testemail" entered. Below the input field, a small error message box is displayed with the text: "Please include an '@' in the email address. 'testemail' is missing an '@'."

8. Input Type file

The input type `file` is used to create an input field that lets the user upload a file or multiple files from their device. For example,

```
<input type="file" name="file">
```

Browser Output



A browser window mockup showing a file input field. The field consists of a button labeled "Choose File" and the text "No file chosen" to its right.

9. Input Type hidden

The input type `hidden` is used to create an invisible input field. This is used to supply additional value to the server from the form that cannot be seen or modified by the user. For example,

```
<input type="hidden" name="id" value="123" >
```

10. Input Type image

The input type `image` is used to create a button using an image.

```
<input type="image" src="/submit.png" alt="submit" >
```

Browser Output



Let's see an example of how we can use it in a form.

```
<form>
  <label for="firstname">First name: </label>
  <input type="text" id="firstname" name="firstname"><br><br>
  <label for="lastname">Last name: </label>
  <input type="text" id="lastname" name="lastname"><br><br>
  <input type="image" src="/submit.png" alt="submit" >
</form>
```

Browser Output

A screenshot of a web browser window showing a form. The form consists of two text input fields. The first field is preceded by the label "First name:" and the second by "Last name:". Below the input fields is a blue button with the word "SUBMIT" in white capital letters.

11. Input Type month

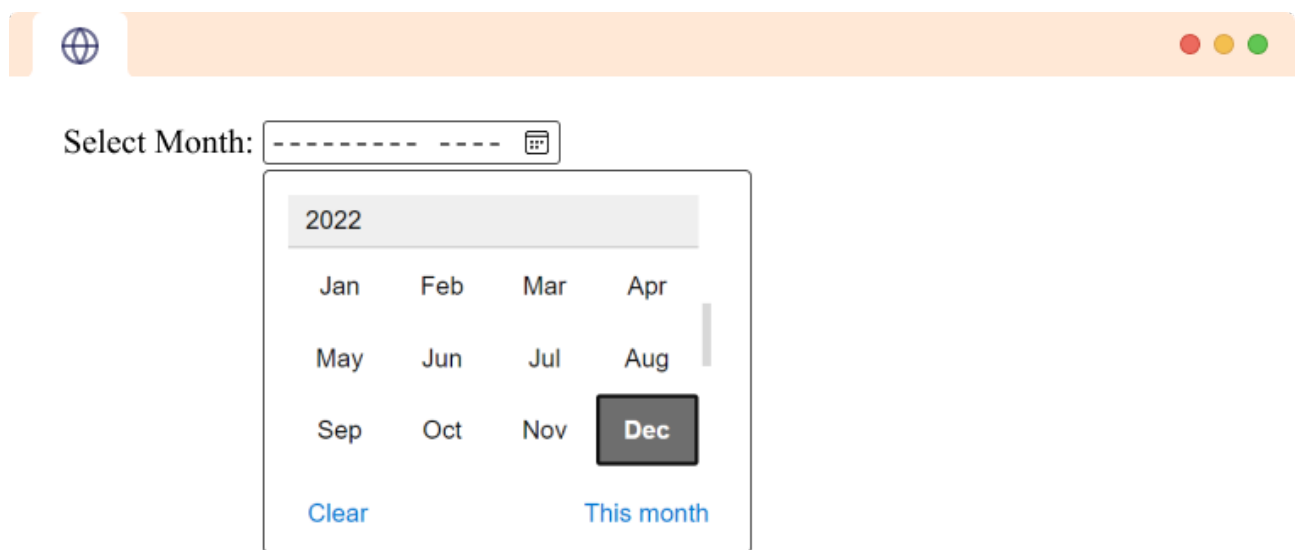
The input type `month` is used to create an input field that lets the user enter month and year using a visual interface from the browser. For example,


```
<label for="start">Select Month:</label>
<input type="month" id="start" >
```

Browser Output (before expanding)

A screenshot of a web browser window with an orange header bar. The browser's address bar shows a globe icon and three colored window control buttons (red, yellow, green) on the right. Below the header, the text 'Select Month:' is followed by a small, collapsed month selection widget. The widget consists of a text box containing dashes '-----' and a small calendar icon on the right.

Browser Output (after expanding)

A screenshot of a web browser window, similar to the one above, showing the 'Select Month:' label followed by an expanded month selection widget. The widget is a light gray box containing a year selector at the top set to '2022'. Below the year is a grid of month abbreviations: Jan, Feb, Mar, Apr in the first row; May, Jun, Jul, Aug in the second row; Sep, Oct, Nov, Dec in the third row. The 'Dec' button is highlighted with a dark gray background. At the bottom of the widget, there are two blue links: 'Clear' on the left and 'This month' on the right.

12. Input Type password

The input type `password` is used to create an input field that lets the user enter information securely. For example,

```
<label for="password">Password:</label>
<input type="password" id="password">
```

Browser Output



The browser displays all the characters the user types using an asterisk (*).

13. Input Type radio

The input type `radio` is used to define a radio button. Radio buttons are defined in a group. Radio buttons let users pick one option out of a group of options.

```
<form>
  <input type="radio" id="cat" name="animal" value="cat">
  <label for="cat">cat</label>
  <input type="radio" id="dog" name="animal" value="dog">
  <label for="dog">dog</label>
</form>
```

Browser Output



From the above example, we can see that all the radio buttons share the same `name` attribute. It allows the user to select exactly one option from the group of radio buttons.

When submitting the form data, the key for the input will be the `name` attribute, and the value will be the radio button selected.

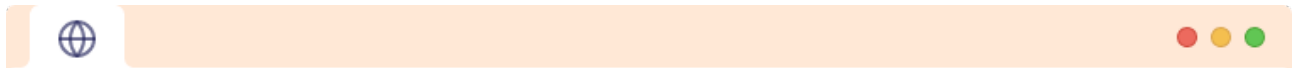
Note: The `name` attribute is used as the key for the data when submitting the form.

14. Input Type range

The input type `range` is used to create a range picker from which the user can select the value. User can select a value from the range given. It has a default range from **0** to **100**. For example,

```
<label for="range">Select value: </label>
<input type="range" id="range" value="90">
```

Browser Output



15. Input Type reset

The input type `reset` defines the button which clears all the form values to their default value. For example,

```
<form>
  <label for="name">Name:</label>
  <input id="name" type="text" /><br />
  <input type="reset" value="Reset" />
</form>
```

Browser Output



Browser Output (after reset)

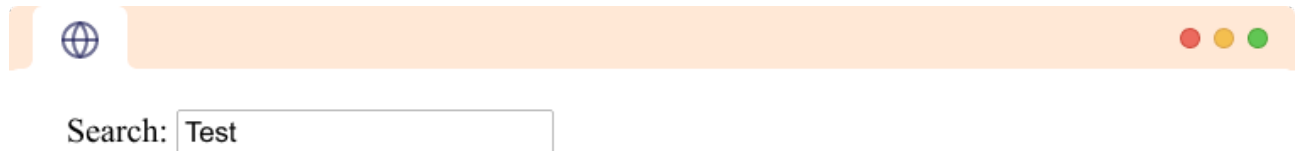


16. Input Type search

The input type `search` allows user to enter their search queries in the text fields. It is similar to input type `text`. For example,

```
<label for="search">Search: </label><input type="search" id="search" >
```

Browser Output



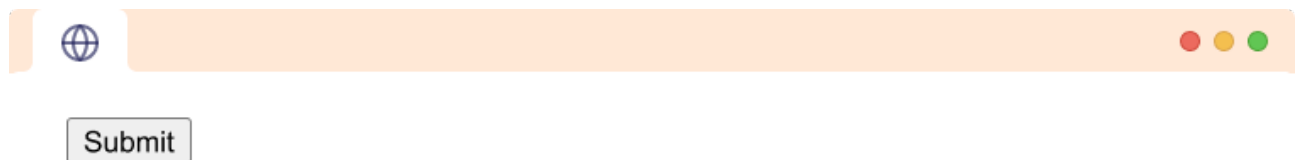
Note: The search input does not work as a search unless we use some JavaScript to do the search calculation.

17. Input Type submit

The input type `submit` is used when the user submits the form to the server. It is rendered as a button. For example,

```
<input type="submit" value="submit">
```

Browser Output



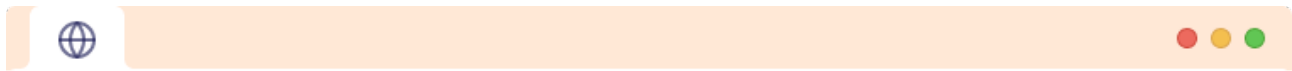
Here, The text provided in the `value` attribute of the input is shown in the button.

18. Input Type tel

The input type `tel` is used to define the field to enter a telephone number. The telephone number is not automatically validated to a particular format as telephone formats differ across the world. It has an attribute named `pattern` used to validate the entered value. For example,

```
<label for="phone">Phone Number:</label>
<input type="tel" id="phone" pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}">
```

Browser Output



Phone Number:

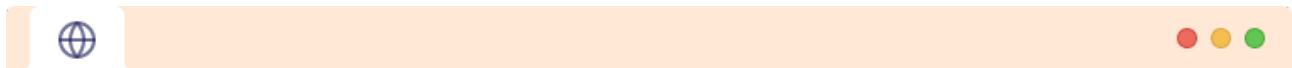
The pattern in the above example allows numbers in the format XXX-XX-XXX where X is a number between 0 and 9.

19. Input Type time

The input type `time` attribute creates an input field that accepts time value. It allows users to add time in hours, minutes, and seconds easily. For example,

```
<label for="time">Select Time:</label>
<input type="time" id="time">
```

Browser Output (before expanding)



Select Time:

Browser Output (after expanding)



Select Time: 

04	00	PM
05	01	AM
06	02	
07	03	
08	04	
09	05	
10	06	

20. Input Type url

The input type `url` is used to let the user enter and edit a URL. For example,

```
<label for="url">URL:</label>  
<input type="url" id="url" placeholder="https://example.com"  
pattern="https://.*">
```

Browser Output



URL:

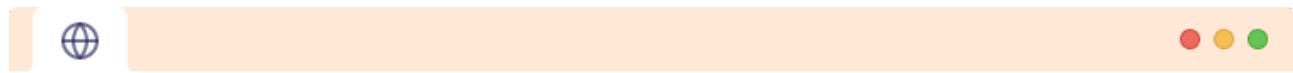
Here, `placeholder` is a hint that specifies the expected value of an input field, and the `pattern` defines what type of value is accepted. The above pattern means that only text beginning with `https://` will be valid.

21. Input Type week

The input type `week` lets the user pick a week and a year from a calendar. For example,

```
<label for="week">Week</label>
<input type="week" id="week" >
```

Browser Output



Week: 

December 2022 ▾

↑ ↓

Week	Su	Mo	Tu	We	Th	Fr	Sa
48	27	28	29	30	1	2	3
49	4	5	6	7	8	9	10
50	11	12	13	14	15	16	17
51	18	19	20	21	22	23	24
52	25	26	27	28	29	30	31
1	1	2	3	4	5	6	7

Clear

This week

Introduction to CSS

Cascading Style Sheets, commonly known as CSS, is a stylesheet language used in web development to control the presentation and styling of HTML and XML documents. CSS allows web designers and developers to define the look and layout of web pages, separating the content (HTML) from the presentation (CSS). Here's an introduction to Cascading Style Sheets:

1. **Separation of Concerns**: CSS enables the separation of content and presentation. HTML is used to structure the content of a web page, defining headings, paragraphs, images, links, and more, while CSS is responsible for controlling the visual aspects like colors, fonts, spacing, and positioning.
2. **Selectors and Rules**: CSS uses selectors to target HTML elements, and each selector is associated with a set of rules that specify how those elements should be styled. For example, you can select all `<p>` (paragraph) elements and set their text color to blue.

```
```css
```

```
p {
```

```
 color: blue;
```

```
}
```

```
```
```

3. **Properties and Values**: CSS rules consist of properties and values. Properties define what aspect of the element you want to style, such as `color` for text color or `font-size` for the size of text. Values set the specific style, like "red" for color or "16px" for font size.

```
```css
```

```
p {
```

```
 color: red;
```



```
font-size: 16px;
```

```
}
```

```
...
```

4. **Cascading**: The "C" in CSS stands for "Cascading," which refers to the system that determines which styles should be applied when there are conflicting rules. CSS rules can be defined in different ways, such as inline styles, internal styles in the `<style>` element within an HTML document, or external stylesheets in separate .css files. The rules cascade in a specific order of importance, with the more specific rules taking precedence over more general ones.

5. **Inheritance**: CSS allows properties to be inherited by child elements from their parent elements. For example, if you set a font size on a parent container, the child elements within it will inherit that font size unless otherwise specified.

6. **Responsive Design**: CSS plays a crucial role in creating responsive web designs. With CSS, you can use media queries to adapt the layout and styling of your web page based on the device's screen size and orientation. This is essential for ensuring that web pages look good on various devices, from desktop monitors to smartphones and tablets.

7. **Cross-Browser Compatibility**: CSS helps ensure that web pages look consistent across different web browsers. However, achieving cross-browser compatibility can sometimes be a challenge due to variations in how browsers interpret and render CSS rules. Web developers often use vendor prefixes and other techniques to address these differences.

8. **External Stylesheets**: It is common practice to create external CSS files, separate from the HTML documents. This approach makes it easier to maintain and update styles across multiple web pages. You link these external stylesheets to your HTML documents using the `<link>` element within the `<head>` section.

```
```html
```

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

...

CSS is a powerful tool for web designers and developers to create visually appealing and responsive web pages. It allows for fine-grained control over the appearance of web content and helps maintain consistency in the design of a website.

Types of CSS (Cascading Style Sheet)

Cascading Style Sheet (CSS) is used to set the style in web pages that contain HTML elements. It sets the background color, font-size, font-family, color, ... etc. properties of elements on a web page.

There are three types of CSS which are given below:

- Inline CSS
- Internal or Embedded CSS
- External CSS

Inline CSS: Inline CSS contains the CSS property in the body section attached to the element is known as inline CSS. This kind of style is specified within an HTML tag using the style attribute.

Example: This example shows the application of inline-css.

```
<!DOCTYPE html>

<html>

<head>

    <title>Inline CSS</title>

</head>

<body>

    <p style="color:#009900; font-size:50px;
            font-style:italic; text-align:center;">    New
            Horizon College of Engineering

    </p>

</body>
```

</html>

Internal or Embedded CSS: This can be used when a single HTML document must be styled uniquely. The CSS rule set should be within the HTML file in the head section i.e. the CSS is e<!DOCTYPE html>embedded within the <style> tag inside the head section of the HTML file.

Example: This example shows the application of internal-css.

<html>

<head>

<title>Internal CSS</title>

<style>

```
.main {  
    text-align: center;  
}
```

```
.NHCE {  
    color:    #009900;  
    font-size: 50px;  
    font-weight: bold;  
}
```

```
.ISE {  
    font-style: bold;  
    font-size: 20px;
```

}

</style>

</head>

<body>

<div class="main">

```
<div class="NHCE">New Horizon College of Engineering</div>
```

```
<div class="ISE">
```

```
    A computer science dept of NHCE
```

```
</div>
```

```
</div>
```

```
</body>
```

```
</html>
```

External CSS: External CSS contains separate CSS files that contain only style properties with the help of tag attributes (For example class, id, heading, ... etc). CSS property is written in a separate file with a .css extension and should be linked to the HTML document using a **link** tag. It means that, for each element, style can be set only once and will be applied across web pages.

Example: The file given below contains CSS property. This file saves with .css extension.
For Ex: **sample.css**

```
body {  
    background-color:powderblue;  
}  
.main {  
    text-align:center;  
}NHCE {  
#ISE {  
    font-style:bold;    font-  
    size:20px;  
}
```

Below is the HTML file that is making use of the created external style sheet.

- **link** tag is used to link the external style sheet with the html webpage.
- **href** attribute is used to specify the location of the external style sheet file.

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <link rel="stylesheet" href="sample.css" />
```

```
</head>
```

```
<body>
```

```
    <div class="main">
```

```
        <div class="NHCE">New Horizon College of Engineering</div>
```

```
        <div id="ISE">
```

```
            A computer science dept of NHCE
```

```
        </div>
```

```
    </div>
```

```
</body>
```

</html>

CSS Selectors

The following is a list of the most common and well-supported CSS selectors. There are many, many more, but these are the ones you should know well.

- [Element Type Selectors](#)
- [Descendant Selectors](#)
- [Class selectors](#)
- [Id Selectors](#)
- [Child Selectors](#)
- [Adjacent sibling selectors](#)
- [Pseudo Selectors](#)
- [Universal Selectors](#)

Element Type Selectors

The most basic CSS selectors are Element Type Selectors. That's a fancy name for simply **using an HTML tag**, without the angle braces.

We've used this selector extensively already.

For example, if we wanted to make **all paragraphs** have green text, we would use the following CSS rule:

```
p { color: green; }
```

•

Descendant Selectors

Match an element that is a descendant of another element. This uses two separate selectors, separated by a space.

For example, if we wanted **all emphasized text in our paragraphs** to be green text, we would use the following CSS rule:

```
p em { color: green; }
```

Class Selectors

Match an element that has the specified class.

To match a specific class attribute, we always start the selector with a period, to signify that we are looking for a class value. The period is followed by the class attribute value we want to match.

For example, if we wanted **all elements with a class of "highlight"** to have a different background color, we would use the following CSS rule:

```
.highlight { background-color: #ffcccc; }
```

Child selectors

Match an element that is an immediate child of another element.

For example, if we wanted **all emphasized text in our paragraphs's** to have green text, **but not emphasized text in other elements**, we would use the following CSS rule:

```
p > em { color: green; }
```

Pseudo Selectors

An Aside About Link States

Anchor elements are special. You can style the `<a>` element with an Element Type Selector, but it might not do exactly what you expect. This is because links have different states, that relate to how they are interacted with. The four primary

states of a link are: link, visited, hover, active. By far the most common pseudo selectors are used to style our links. There are four different pseudo selectors to be used in conjunction with links:

:link

A link that has not been previously visited (visited is defined by the browser history)

:visited

A link that has been visited

:hover

A link that the mouse cursor is "hovering" over

:active

A link that is currently being clicked

```

a:link          { color: red }          /* unvisited links
*/ a:visited
    */ a:hover  { color: blue }          /* visited links
*/
                { color: green }        /* user hovers
a:active        { color: lime }          /* active links*/

```

For reasons of browser compatibility, you should always specify the pseudo selectors in this order.

Universal Selector

Matches every element on the page.

For example, if we wanted **every element** to have a solid 1px wide border, we would use the following CSS rule:

```
* { border: 1px solid blue;}
```

For reasons that are likely obvious after the previous example, you should be careful with universal selectors. When might you want to use them?

The answer is, not often. But an example would be to **set the margins and padding for all elements** on the page to zero. We'll learn a better way to do this shortly.

```
* {
    margin: 0;
    padding: 0;
}
```

In CSS, class selectors and ID selectors are both used to target and apply styles to HTML elements, but they have some key differences in terms of usage and specificity.

Class Selector:

- **Syntax:** Class selectors are prefixed with a dot (.) followed by the class name.

```
/* CSS class selector */
.myClass { color:
  blue;
  font-weight: bold;
}
```

HTML Usage: Apply the class to an HTML element using the class attribute.

```
<!-- HTML with class -->
<p class="myClass">This is a paragraph with the class style.</p>
<h1 class="myClass">NHCE</h1>
```

- **Usage Notes:**
- Classes are reusable, meaning you can apply the same class to multiple elements.
- Multiple classes can be applied to a single element.
- Class selectors have lower specificity compared to ID selectors.

ID Selector:

- **Syntax:** ID selectors are prefixed with a hash (#) followed by the ID name.

```
/* CSS ID selector */ #myId {
  color: red;
  font-style: italic;
}
```

HTML Usage: Apply the ID to an HTML element using the id attribute. Note that the ID must be unique within the HTML document.

```
<!-- HTML with ID -->
```


<p id="myId">This is a paragraph with the ID style.</p>

- **Usage Notes:**
- IDs should be unique within a page. Unlike classes, the same ID should not be applied to multiple elements.
- IDs have higher specificity compared to class selectors.
- While classes encourage a modular and reusable approach, IDs are generally used for unique and specific elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    /* Class selector */
    .myClass { color:
      blue;
      font-weight: bold;
    }

    /* ID selector */ #myId {
      color: red;
      font-style: italic;
    }
  </style>
  <title>Class vs ID Example</title>
</head>
<body>
  <p class="myClass">This is a paragraph with the class style.</p>
  <p id="myId">This is a paragraph with the ID style.</p>
</body>
</html>
```

The Box model

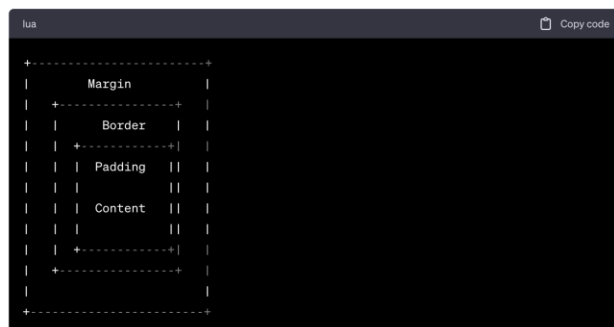
The box model is a fundamental concept in CSS (Cascading Style Sheets) that describes the layout of elements on a web page. Each HTML element is considered a box, and the box model defines the

properties of these boxes, including content, padding, border, and margin. Understanding the box model is crucial for styling and layout design in web development.

The box model consists of four main components:

- **Content:**
 - This is the actual content of the box, such as text, images, or other HTML elements.
 - It is defined by the width and height properties.
- **Padding:**
 - Padding is the space between the content and the border.
 - It can be set using the padding property and can have values for top, right, bottom, and left sides.
- **Border:**
 - The border surrounds the padding and content.
 - It is defined by the border property and can have properties like border-width, border-style, and border-color.
-

Here's a visual representation of the box model:



Mr

?

Print Line

- The margin is the space outside the border.
- the element's border and surrounding elements.
- Margin is set using the margin property and can have values for top, right, bottom, and left sides.

```
.box {  
  
    width: 200px; height:  
  
    100px;      padding:  
  
    20px;  
  
    border: 2px solid #333; margin: 10px;  
  
}
```

And corresponding HTML:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial- scale=1.0">  
  <title>Box Model Example</title>  
  <link rel="stylesheet" href="styles.css">  
</head>  
<body>  
  <div class="box">This is a box with content, padding, border, and margin.</div>  
</body>  
</html>
```

In this example:

- The width and height properties define the content area.
- The padding creates space inside the content area.
- The border surrounds the padding.
- The margin provides spacing outside the border.

Remember that the actual space occupied by an element on the web page includes the content area, padding, border, and margin. The total width of the element is calculated as follows: $\text{width} + 2 * \text{padding} + 2 * \text{border} + 2 * \text{margin}$. Understanding the box model is essential for precise layout control in web development.

Background images

Background images in CSS allow you to set an image as the background of an HTML element. You can use background images to enhance the visual appeal of your website or to provide additional context to certain sections. Here's how you can use background images in CSS:

1. Background Image Property:

The background-image property is used to set the background image for an element.

Example:

```
body {  
    background-image: url('background.jpg');  
    background-size: cover; /* Adjust as needed, 'cover' is one option  
*/  
}
```

In this example, the background image is set for the entire body element. The url('background.jpg') specifies the path to the image file. You can replace 'background.jpg' with the actual path to your image.

2. Background Position:

The background-position property defines the starting position of the background image. You can specify values in pixels, percentages, or keywords like center or bottom.

Example:

```
body {  
    background-image: url('background.jpg');  
    background-position: center; /* Center the background image */  
}
```

Background Attachment:

The background-attachment property determines whether the background image scrolls with the content or remains fixed.

```
body {  
  
    background-image: url('background.jpg'); background-attachment: fixed; /*  
  
    Fixed background */  
}
```

```
body {  
  
    background-image: url('background.jpg'); background-size:  
  
    cover;  
  
    background-position: center; background-  
  
    attachment: fixed;  
  
}
```

The and <div> tags

The and <div> tags are both HTML elements that are used to group and apply styles to content, but they serve different purposes.

Tag:

The tag is an inline HTML element that is used to apply styles to a specific portion of text or inline elements within a larger block of content. It does not add any visual formatting on its own but is often used in conjunction with CSS to style a specific portion of text.

Example:

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

In this example, the tag is used to apply the color blue to the word "blue" within the paragraph.

<div>Tag:

The <div>tag is a block-level HTML element that is used to group and structure content. It is often used as a container for other HTML elements, allowing you to apply styles or manipulate the group as a whole. The <div>tag is a versatile container and is commonly used in layout design.

Example:

```
<div style="border: 1px solid black; padding: 10px;">
  <h2>Section Title</h2>
  <p>This is a paragraph of text within a <code>&lt;div&gt;</code> element.</p>
```

</div>

In this example, the <div>tag is used to create a container that wraps around a heading (<h2>) and a paragraph (<p>), and styling is applied to the entire container.

Common Use Cases:

- **Tag:**
 - Applying inline styles to a specific portion of text.
 - Applying JavaScript functionality to a specific inline element.
- **<div>Tag:**
 - Grouping and structuring content for layout purposes.
 - Creating containers for styling or applying CSS styles to a group of elements.
 - Serving as a target for JavaScript manipulation.

Both and <div> are essential in web development, and their usage depends on the specific needs of the content and the design of the webpage. is typically used for inline styling or targeting specific text, while <div> is used for grouping and structuring content for layout and styling purposes.

Question Bank:

1. Describe Standard XHTML document structure with neat diagram.
2. Differentiate between ordered list, unordered list and description list with sample html code.
3. Design webpage which includes following data
 - a) Employee Introduction
 - b) Employee achievement
 - c) Employee work experience
4. Differentiate between HTML and XHTML.
5. Explain Cascading Style Sheets.
6. Explain Box model with diagram
7. Illustrate forms with HTML Code.
8. Illustrate span and div tag with HTML code.
9. Design webpage form which includes following data
 - a) Username or email id
 - b) Password
 - c) Gender(radio button to be used)
 - d) Language preferences
 - e) Submit and reset button
10. Show how to include background image in web page.

MODULE 2

HTML 5

Detecting HTML 5 features – Advanced CSS: Layout, Normal Flow, Positioning Elements, Floating Element

CSS provides several advanced layout techniques that allow developers to create complex and responsive designs. Here are some key aspects of advanced CSS layout:

1. Flexbox:

Flexbox (Flexible Box Layout) is a one-dimensional layout method for laying out items in rows or columns. It provides a more efficient and predictable way to distribute space and align items within a container.

.container

`display: flex;`

`justify-content: space-between;`

`align-items`

}

.item

`flex: 1;`

}

CSS provides several advanced layout techniques that allow developers to create complex and responsive designs. Here are some key aspects of advanced CSS layout:

1. Flexbox:

Flexbox (Flexible Box Layout) is a one-dimensional layout method for laying out items in rows or columns. It provides a more efficient and predictable way to distribute space and align items within a container.

`.container { display: flex; justify-content: space-between; align-items: center; } .item { flex: 1; }`

2. CSS Grid:

CSS Grid Layout is a two-dimensional layout system that allows you to create complex grid-based layouts with rows and columns. It is particularly useful for designing responsive web applications.

.container

`display: grid;`

`grid-template-columns 1 2 1`

`grid-template-rows`

```
    gap: 10px;
}
.item
    grid-column: 2;
}
```

3. Grid Layout with Flexbox for Alignment:

Combining CSS Grid and Flexbox allows you to create powerful layouts. Use Grid for the overall structure and Flexbox for alignment and distribution within grid items.

```
.container
    display: grid;
    grid-template-columns 1 1
    gap: 20px;
}

.item
    display: flex;
    flex-direction
    justify-content: space-between;
    align-items
}
```

4. Multi-Column Layout:

The multicol property allows you to create multi-column layouts, where content flows into multiple columns within a container.

```
.container
    column-count: 3
    column-gap: 20px
```

5. Normal Flow:

In the normal flow, elements are laid out one after another in the order they appear in the HTML document. The flow can be either horizontal (left to right) or vertical (top to bottom), depending on the elements.

```
/* Elements will appear in the normal flow */
```

```
.normal-flow
width 200px
height 100px
border: 1px solid #ccc;
margin 10px
}
```

6. Positioning:

Positioning allows you to control the exact placement of an element relative to its containing element or the viewport. The position property is used for this purpose.

Static:

- Default value.
- Elements are positioned according to the normal flow of the document.
- The top, right, bottom, and left properties have no effect.

Relative:

- The element is positioned relative to its normal position in the document flow.
- The top, right, bottom, and left properties can be used to offset the element from its normal position.

Absolute:

- The element is removed from the normal document flow, and its position is relative to its nearest positioned ancestor.
- If there is no positioned ancestor, it is positioned relative to the initial containing block (usually the <html> element).
- The top, right, bottom, and left properties are used to position the element.

Fixed:

- The element is removed from the normal document flow and positioned relative to the viewport (the browser window).
- It remains fixed even when the page is scrolled.
- The top, right, bottom, and left properties are used to position the element

Sticky:

- The element is treated as relative positioned until it crosses a specified point, then it is treated as fixed positioned.
- The top, right, bottom, and left properties are used to specify the "sticking" point.

Example:

```
/* Positioning Example */  
.positioned-element  
    position: relative;  
    top: 20px;  
    left: 30px;  
    border: 1px solid #ccc;  
    padding 10px  
}
```

7. Floating Elements:

Floating elements allows text and inline elements to wrap around them. Floated elements are taken out of the normal flow and positioned to the left or right of their containing element.

Example:

```
/* Floating Example */  
.float-left  
    float: left;  
    width 150px  
    margin 10px  
}  
.float-right  
    float: right;  
    width 150px  
    margin 10px  
}
```

8. Canvas

In HTML, the <canvas> element is used to create graphics using JavaScript. It provides a drawing surface for creating dynamic and interactive content. Here's a basic example of using the <canvas> element along with JavaScript to draw a simple rectangle:

```
<!DOCTYPE html>  
<head>  
    <title>Canvas Example</title>  
    <style>
```

```

    canvas {
        border: 1px solid #000;
    }
</style>
</head>
<body>
<canvas id="myCanvas" width="400" height="200"></canvas>
<script>
    // Get the canvas element
    var canvas = document.getElementById('myCanvas');

    // Get the 2D rendering context
    var context = canvas.getContext('2d');

    // Draw a rectangle
    context.fillStyle = 'lightblue';
    context.fillRect(50, 50, 100, 50);

    // Draw a circle
    context.beginPath();
    context.arc(300, 100, 30, 0, 2 * Math.PI, false);
    context.fillStyle = 'lightgreen';
    context.fill();
    context.lineWidth = 2;
    context.strokeStyle = '#003300';
    context.stroke();
</script>
</body>
</html>

```

Here is context.arc() signature:

arc(x, y, radius, startAngle, endAngle, anticlockwise)

The x and y values define the center of our circle, and the radius will be the radius of the circle upon which our arc will be drawn.

startAngle and endAngle are in radians, not degrees.

anticlockwise is a true or false value that defines the direction of the arc.

In this example:

- The `<canvas>` element is created with the ID "myCanvas" and a width and height.
- The `<script>` section contains JavaScript code that gets the 2D rendering context of the canvas using `getContext('2d')`.
- Two shapes are drawn on the canvas: a filled rectangle and a filled circle.
- `fillStyle` property sets the fill color, and `strokeStyle` sets the stroke color.

9. Video

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Video Example</title>
</head>
<body>
  <h2>HTML Video Example</h2>
  <video width="640" height="360" controls>
    <source src="sample_video.mp4" type="video/mp4">
    Your browser does not support the video tag.
  </video>
</body>
</html>
```

- The `<video>` element is used to embed the video.
- The width and height attributes set the dimensions of the video player.
- The controls attribute adds video controls (play, pause, volume, etc.).
- The `<source>` element inside `<video>` specifies the video file (example.mp4) and its type (video/mp4). You can include multiple `<source>` elements for different video formats to ensure compatibility with various browsers.
- The text "Your browser does not support the video tag." will be displayed if the browser doesn't support the `<video>` tag.

Make sure to replace "example.mp4" with the actual path to your video file. Additionally, you may need to provide multiple sources for different video formats to support various browsers.

Here's a breakdown of the `<video>` attributes:

- src: Specifies the URL of the video file.
- type: Specifies the MIME type of the video file.
- controls: Enables video controls.
- width and height: Set the dimensions of the video player.

10. Local storage

Local Storage is a web storage feature introduced in HTML5 that allows web applications to store data persistently in a web browser. It provides a simple key-value store and is designed to be a more secure and efficient alternative to cookies for storing data on the client side. The data stored in Local Storage persists even when the browser is closed and reopened, making it suitable for long-term storage.

- **Simple Key-Value Store:** Local Storage stores data as key-value pairs. You can store data using a key and retrieve it later using the same key.
- **Capacity:** Local Storage has a larger storage capacity compared to cookies. While the exact storage limit can vary between browsers, it's typically around 5 MB per domain.
- **Data Type:** Data in Local Storage is stored as strings. If you want to store more complex data types (such as arrays or objects), you need to convert them to strings using methods like `JSON.stringify()` before storing and `JSON.parse()` when retrieving.
- **No Expiration:** The data stored in Local Storage does not have an expiration time, unlike cookies which can have an expiry date.
- **Security:** Local Storage is considered more secure than cookies because it is not sent with every HTTP request, reducing the risk of data interception. However, it's important to note that, like cookies, data stored in Local Storage is accessible by JavaScript on the same domain.

Local Storage is a valuable tool for creating more responsive and user-friendly web applications by allowing them to remember user preferences, settings, and other data on the client side.

11. Web Workers

Web Workers are a feature in HTML5 that allow you to run scripts in the background, separate from the main execution thread of a web page. They enable multitasking and parallel processing in web applications, improving overall performance and responsiveness.

Here are some key points about Web Workers:

- **Background Processing:** Web Workers run scripts in the background, independently of the main thread. This helps prevent the main thread from being blocked by time-consuming operations, enhancing the user experience.

- **Multithreading:** Web Workers provide a form of multithreading for web applications. They allow you to perform tasks concurrently, taking advantage of multi-core processors.
- **Communication:** Web Workers communicate with the main thread using a messaging system. They can exchange data by sending messages to each other. The data is serialized and deserialized, ensuring that it can be safely transferred between threads.
- **No DOM Access:** Web Workers do not have direct access to the DOM (Document Object Model) or the window object. This is because they run in a separate thread, and direct DOM access could lead to synchronization issues. However, they can communicate with the main thread through messages.

Types of Web Workers:

- **Dedicated Workers:** These workers are dedicated to a single script file. They have a one-to-one relationship with the script that created them.
- **Shared Workers:** Shared Workers can be accessed by multiple scripts running in different windows or tabs. They have a shared state and can communicate with any script that has a reference to them.

12. Offline applications

Creating offline web applications involves enabling the application to work even when the user is not connected to the internet. This can enhance the user experience by allowing them to access content and perform tasks offline. There are several techniques and technologies that enable the development of offline web applications. Here are some key concepts:

- **Service Workers:**
 - Service Workers are a crucial part of building offline-capable web applications. They are JavaScript files that run in the background, separate from the main browser thread.
 - Service Workers can intercept and handle network requests, allowing developers to implement caching strategies.
 - They enable features like background synchronization, push notifications, and offline support.

Caching Strategies:

- Caching is a key aspect of offline web applications. By caching resources, such as HTML, CSS, JavaScript, and images, you can ensure that these assets are available even when the user is offline.
 - Different caching strategies include:
 - **Cache First:** Serve the resource from the cache, and if it's not there, fetch it from the network.
 - **Network First:** Attempt to fetch the resource from the network, and if that fails, fall back to the cache.
 - **Network Only:** Always fetch the resource from the network and don't use the cache.
 - **Cache Only:** Always use the cache and don't make a network request.

Web Storage (localStorage, sessionStorage):

- Web Storage allows the storage of key/value pairs locally in the user's browser.
- While localStorage persists even when the browser is closed and reopened, sessionStorage is available only for the duration of the page session

13. Geo-location

Geolocation is a feature in web browsers that allows web applications to obtain the user's geographic location. This is achieved using the Geolocation API, which provides a simple way for developers to access location information from the device's hardware, such as GPS, Wi-Fi, or cellular networks.

Key components of the Geolocation API include:

- navigator.geolocation **Object:**
 - The navigator.geolocation object is the main entry point for the Geolocation API.
 - It provides methods for obtaining the device's current location and watching for changes in position.
- getCurrentPosition() **Method:**
 - The getCurrentPosition() method is used to asynchronously request the current location of the device.
 - It takes two callback functions as parameters: one for success (successCallback) and one for handling errors (errorCallback).

- **watchPosition() Method:**
 - The watchPosition() method is used to continuously monitor the device's location.
 - It also takes success and error callback functions, similar to getCurrentPosition().
 - The method returns a unique ID that can be used to later stop watching the position using clearWatch().
- Position Object:
- The position information is represented by a Position object, which contains details such as latitude, longitude, altitude, accuracy, and timestamp.

14. Path

In HTML, the <path> element is commonly used within the <svg> (Scalable Vector Graphics) element to define vector graphics. The <path> element contains a series of commands and parameters that describe the path of the shape. It's a powerful way to create various shapes and lines. Here's a basic example:

```
<svg width="100" height="100">
  <path d="M10 10 H90 V90 H10 Z" fill="blue" />
</svg>
```

In this example:

- M10 10: Move to the point (10, 10).
- H90: Draw a horizontal line to (90, 10).
- V90: Draw a vertical line to (90, 90).
- H10: Draw a horizontal line to (10, 90).
- Z: Close the path by drawing a straight line back to the starting point.

15. Texts

In HTML, you can use the <text> element to display text content. However, in most cases, the <text> element is not directly used for this purpose. Instead, the common way to display text in HTML is by using the <p>, <h1> through <h6>, , or other text-related elements.

16. Gradients and images

In JavaScript, you can work with gradients and images to create interesting visual effects on a webpage. Below, I'll provide a brief overview of how to create gradients and work with images using HTML and JavaScript.

Gradients:

You can create gradients using the CSS linear-gradient and radial-gradient properties. Here's a simple example using JavaScript to apply a linear gradient to the background of an HTML element:

```
<!DOCTYPE html>
<html lang="en">
<head>

<style>
  #gradientDiv
    width: 300px;
    height: 200px;
  }
</style>
<title>Gradient and Images in JavaScript</title>
</head>
<body>

  <div id="gradientDiv">

<script>
  // JavaScript to apply linear gradient dynamically
  var gradientDiv = document.getElementById('gradientDiv');
  gradientDiv.style.background = 'linear-gradient(to right, #ff7e5f, #feb47b)';
</script>

</body>
</html>
<html lang="en">
```

Images

You can also manipulate images using JavaScript. Here's a simple example of changing the source of an image dynamically:

```
<!DOCTYPE html>
<head>
  <title>Gradient and Images in JavaScript</title>
</head>
<body>


<script>
  // JavaScript to change image source dynamically
  var myImage = document.getElementById('myImage');
  myImage.src = 'new-image.jpg';
</script>
</body>
</html>
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
  #gradientDiv {
    width: 300px;
    height: 200px;
  }
</style>
<title>Gradient and Images in JavaScript</title>
</head>
<body>
<div id="gradientDiv"></div>
<script>
  // JavaScript to apply linear gradient dynamically
  var gradientDiv = document.getElementById('gradientDiv');
  gradientDiv.style.background = 'linear-gradient(to right, #ff7e5f, #feb47b)';
</script>
```

```
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    #gradientDiv {
      width: 300px;
      height: 200px;
    }
  </style>
  <title>Gradient and Images in JavaScript</title>
</head>
<body>

<div id="gradientDiv"></div>

<script>
  // JavaScript to apply linear gradient dynamically
  var gradientDiv = document.getElementById('gradientDiv');
  gradientDiv.style.background = 'linear-gradient(to right, #ff7e5f, #feb47b)';
</script>

</body>
</html>
```

Question Bank

- What is HTML5?What are the new features in HTML5?What is the purpose of semantic elements in HTML5?
- What is the difference between localStorage and sessionStorage in HTML5?
- What is the role of the <section> and <article> elements?
- What is the Geolocation API in HTML5?
- How do the <audio> and <video> elements work?
- What is the <canvas> element used for in HTML5?

MODULE 3

Javascript

The JavaScript language

JavaScript was initially created to “make web pages alive”.

The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads.

Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.

In this aspect, JavaScript is very different from another language called Java.

```
<!DOCTYPE html>
<html>
<body>
<h2>My First JavaScript</h2>
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>
<p id="demo"></p>
</body>
</html>
```

Why Study JavaScript?

JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages
4. JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example: The `<script>` Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
```

```
</script>
```

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

Javascript Syntax

Variables:

```
var x = 10; // Declaring a variable
```

```
let y = 20; // Block-scoped variable (introduced in ECMAScript 6)
```

```
const z = 30; // Constant (cannot be reassigned)
```

Data Types:

- Primitive types: string, number, boolean, null, undefined
- Complex types: object, array, function

Operators:

```
// Arithmetic operators
```

```
var sum = 5 + 3;
```

```
var difference = 10 - 5;
```

```
var product = 2 * 4;
```

```
var quotient = 8 / 2;
```

```
// Comparison operators
```

```
var isEqual = (a === b); // Strict equality
```

```
var isNotEqual = (x !== y);
```

```
// Logical operators
```

```
var andOperator = (a && b);
```

```
var orOperator = (x || y);
```

```
var notOperator = !isTrue;
```

Control Flow:

//Conditional Statements:

```
if (condition) {
```

```
    // code to be executed if the condition is true
```

```
} else if (anotherCondition) {
```

```
// code to be executed if anotherCondition is true
} else {
    // code to be executed if none of the conditions are true
}
```

Loops:

// For loop

```
for (var i = 0; i < 5; i++) {
    // code to be executed in each iteration
}
```

// While loop

```
while (condition) {
    // code to be executed while the condition is true
}
```

// Do-while loop

```
do {
    // code to be executed at least once, then checked for the condition
} while (condition);
```

Functions:

```
function add(a, b) {
    return a + b;
}
var result = add(3, 5);
```

Objects:

```
var person = {
    name: 'John',
    age: 30,
    isStudent: false,
    greet: function() {
        console.log('Hello!');
    }
};
console.log(person.name); // Accessing object property
person.greet(); // Calling object method
```

Arrays:

```
var fruits = ['apple', 'o', 'banana'];  
console.log(fruits[0]); // Accessing array element  
fruits.push('grape'); // Adding an element to the end of the array
```

Events:

```
// HTML: <button id="myButton">Click me</button>  
var button = document.getElementById('myButton');  
button.addEventListener('click', function() {  
    // code to be executed when the button is clicked  
});
```

General syntactic characteristics

JavaScript has several syntactic characteristics that define its structure and how code is written.

Here are some general syntactic characteristics of JavaScript:

A. Case Sensitivity: JavaScript is case-sensitive. For example, myVariable and myvariable are treated as two different variables.

B. Semicolons: Semicolons are used to terminate statements, but they are often optional. However, it's considered good practice to use them to avoid potential issues.

```
var x = 5; // Semicolon is used to terminate the statement
```

C. Comments: Comments can be added using // for single-line comments or /* */ for multi-line comments.

```
// This is a single-line comment
```

```
/*
```

```
 * This is a multi-line comment
```

```
 * spanning multiple lines
```

```
*/
```

D. Whitespace: JavaScript ignores whitespace (spaces, tabs, and line breaks) outside of string literals.

```
var   y   = 10; // Whitespace doesn't matter here
```

E. Blocks: Blocks of code are enclosed in curly braces {}.

```
if (condition) {  
    // Code inside the block
```

```
}
```

F. Variables: Variables are declared using var, let, or const. The modern practice is to use let and const over var for various reasons, including block scoping.

```
var variableName = "value";  
let anotherVariable = 42;  
const pi = 3.14;
```

G. Operators: JavaScript supports various operators for arithmetic, comparison, logical operations, etc.

```
var sum = 5 + 3;  
var isEqual = (x === y);  
var logicalAnd = (a && b);
```

H. Functions: Functions are declared using the function keyword.

```
function myFunction(parameter) {  
    // Function body  
}
```

I. Objects: Objects are created using curly braces {} and can contain key-value pairs.

```
var person = {  
    name: "John",  
    age: 30  
};
```

J. Arrays: Arrays are created using square brackets [].

```
var colors = ["red", "green", "blue"];
```

K. Strings: Strings are sequences of characters enclosed in single (') or double (") quotes.

```
var message = "Hello, World!";
```

Screen output and keyboard input in javascript

In JavaScript, if you're running your code in a web browser, the common way to perform screen output and handle keyboard input is through the browser's console object for output and event listeners for keyboard input.

Screen Output:

Using console.log():

```
// Output to the browser console  
console.log("Hello, World!");
```

When you run this code in a browser, you can view the output by opening the browser's developer tools

```
<html>  
<head>  
  <title>Output Example</title>  
</head>  
<body>  
  <div id="output"></div>  
  
  <script>  
    // Output to the HTML page  
    document.getElementById('output').innerText = 'Hello, World!';  
  </script>  
</body>  
</html>
```

In this example, the text "Hello, World!" is inserted into an HTML element with the id 'output'.

Keyboard Input:

Using Event Listeners:

```
<html>  
<head>  
  <title>Input Example</title>
```

```

</head>
<body>
  <input type="text" id="inputField" placeholder="Type something...">
  <div id="output"></div>

  <script>
    // Event listener for key press
    document.getElementById('inputField').addEventListener('keyup', function(event) {
      // Output the pressed key
      document.getElementById('output').innerText = 'You pressed: ' + event.key;
    });
  </script>
</body>
</html>

```

In this example, the script listens for keyup events on an input field and updates the content of the 'output' div with the pressed key.

Note:

1. The innerText property is used to get the current text content, and it's logged to the console.
2. The innerText property is then used to update the text content of the element.
3. innerText property in JavaScript, it's not a "keyword," but rather a property that allows you to get or set the text content of an element, excluding HTML tags.

NOTE: AddEventListener

The first argument to addEventListener specifies the type of event to listen for (e.g., 'click', 'mouseover', 'keydown', etc.), and the second argument is the function to be executed when the event occurs.

Control statements programs in javascript

Example 1: if Statement

```

// Example of an if statement
var x = 10;
if (x > 5) {
  console.log("x is greater than 5");
}

```

```
} else {  
    console.log("x is not greater than 5");  
}
```

Example 2: else if Statement

// Example of an else if statement

```
var grade = 75;
```

```
if (grade >= 90) {  
    console.log("A");  
} else if (grade >= 80) {  
    console.log("B");  
} else if (grade >= 70) {  
    console.log("C");  
} else {  
    console.log("F");  
}
```

// Example 3: switch statement

```
var day = "Monday";
```

```
switch (day) {  
    case "Monday":  
        console.log("It's the beginning of the week");  
        break;  
    case "Friday":  
        console.log("It's almost the weekend");  
        break;  
    default:  
        console.log("It's a regular day");  
}
```

// Example 4: for loop

```
for (var i = 0; i < 5; i++) {  
    console.log("Iteration " + (i + 1));  
}
```

Example 5: while Loop


```
// Example of a while loop
var count = 0;
while (count < 5) {
    console.log("Count: " + count);
    count++;
}
```

Example 6: do...while Loop

```
// Example of a do...while loop
var i = 0;
do {
    console.log("i: " + i);
    i++;
} while (i < 5);
```

Example 7: break Statement

```
// Example of a break statement
for (var i = 0; i < 10; i++) {
    if (i === 5) {
        console.log("Breaking the loop at i=5");
        break;
    }
    console.log("Iteration " + i);
}
```

Example 8: continue Statement

```
// Example of a continue statement
for (var i = 0; i < 5; i++) {
    if (i === 2) {
        console.log("Skipping iteration at i=2");
        continue;
    }
    console.log("Iteration " + i);
}
```

Object creation and modification

Object Literal Syntax (Creation and Modification):

In JavaScript, you can create and modify objects dynamically. Objects in JavaScript can be created using the object literal syntax, the Object constructor, or by defining a custom constructor function. Once created, you can modify the properties and methods of objects.

```
// Using object literal
```

```
let person = {  
  name: 'John',  
  age: 30,  
  City: 'Bangalore',  
  sayHello: function() {  
    }  
};
```

Modifying Objects:

1. Adding or Modifying Properties:

```
person.city = 'New York'; // Adding a new property  
person.age = 31; // Modifying an existing property  
person.address='NHCE';
```

2. Deleting Properties:

```
delete person.age; // Deleting a property
```

Arrays in Javascript

In javascript, arrays are used to store multiple values in a single variable. You can create an array using square brackets and separates the values with commas.

```
let myArray = [1, 2, 3, "four", true];
```

You can access elements in an array using their index starting from 0:

```
Let secondElement= myArray[1];
```

Arrays also have various built-in methods for manipulation, such as 'push' to add elements, 'pop' to remove the last element, 'length' to get the number of elements and more

```
myArray.push(5);
```

```
Let removeElement = myArray.pop();
```

```
Let arrayLength = myArray.length;
```

Functions in Javascript

In JavaScript, functions are blocks of reusable code that perform a specific task. You can define functions using the function keyword. Here's a basic example of a function:

```
// Function definition
```

```
function greet(name) {  
  console.log('Hello, ' + name + '!');  
}
```

```
// Function invocation
```

```
greet('John'); // Output: Hello, John!
```

Functions can also have a return value. Here's an example:

```
// Function definition with a return value
```

```
function addNumbers(a, b) {  
  return a + b;  
}
```

```
var result= addNumbers(1,2)
```

```
console.log(result);
```

```
// Function invocation and using the return value
```

```
let result = addNumbers(5, 3);
```

```
console.log(result); // Output: 8
```

Constructor in Javascript

a constructor is a special method used for creating and initializing objects created with a class.

Here's an example of a constructor in JavaScript:

```
// Defining a class with a constructor
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  // Other methods can be defined outside the constructor
  greet() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
  }
}

// Creating an instance of the class using the constructor
const person1 = new Person('John', 25);

// Accessing properties set by the constructor
console.log(person1.name); // Output: 'John'
console.log(person1.age); // Output: 25

// Calling a method defined in the class
person1.greet(); // Output: Hello, my name is John and I am 25 years old.

In this example:
```

- The Person class has a constructor that takes name and age parameters. The constructor is automatically called when you create a new instance of the class using the new keyword.
- Inside the constructor, this.name and this.age are used to set the properties of the object being created.
- After creating an instance of the Person class (person1), you can access its properties and call its methods.

Constructors provide a way to initialize the properties of an object when it is created, allowing you to set initial values based on the parameters passed to the constructor. They are commonly used in class-based object-oriented programming in JavaScript.

Pattern matching using regular expressions

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the `exec()` and `test()` methods of `RegExp`, and with the `match()`, `matchAll()`, `replace()`, `replaceAll()`, `search()`, and `split()` methods of `String`.

Creating a Regular Expression:

You can create a regular expression using the `RegExp` constructor or using a literal notation between slashes (`/.../`). For example:

```
// Using the RegExp constructor
```

```
let pattern1 = new RegExp('hello');
```

```
// Using a literal notation
```

```
let pattern2 = /world/;
```

Testing for a Match:

You can use the `test` method to check if a string matches a pattern:

```
let str = 'Hello, world!';
```

```
let pattern = /world/;
```

```
let result = pattern.test(str);
```

```
console.log(result); // Output: true
```

Matching and Extracting:

The `exec` method can be used to find the first match in a string and extract information:

```
let str = 'The cat and the mat.';
```

```
let pattern = /cat/;
```

```
let result = pattern.exec(str);
```

```
console.log(result[0]); // Output: 'cat'
```

Global Matching:

If you want to find all matches in a string, you can use the `g` flag:

g: Global flag, indicating that the pattern should be applied globally to the entire string, not just stopping after the first match.

```
let str = 'apple, banana, cherry';  
let pattern = /a/g;  
let matches = str.match(pattern);  
console.log(matches);
```

Replacement:

You can use the replace method to replace matched patterns with a specified string:

```
let str = 'I love JavaScript!';  
let pattern = /JavaScript/;  
let newStr = str.replace(pattern, 'coding');  
console.log(newStr);
```

search() Method:

Searches the string for a specified value and returns the position of the first match.

```
const pattern = /\d+/  
const text = "abc123def";  
const position = text.search(pattern); // Returns 3
```

split() Method:

Splits a string into an array of substrings based on a specified delimiter (in this case, a regular expression).

```
const pattern = /\s+/  
const text = "Hello World";  
const words = text.split(pattern); // Returns ["Hello", "World"]
```

Pattern matching in javascript

Regular Expressions:

Regular expressions can be used for pattern matching in strings.

```
const emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;  
const email = 'example@email.com';  
if (emailPattern.test(email)) {  
  console.log('Valid email address.');
```

```
} else {  
  console.log('Invalid email address.');
```

```
}
```

Array Methods:

Methods like filter, map, and find can be used to match patterns in arrays.

```
const numbers = [1, 2, 3, 4, 5];
```

```
const evenNumbers = numbers.filter(num => num % 2 === 0);
```

```
console.log(evenNumbers); // Output: [2, 4]
```

Question Bank

- What is JavaScript? What are the different data types in JavaScript?
- What is the difference between `var`, `let`, and `const`?
- What is `this` keyword in JavaScript?
- What is an event in JavaScript?
- What is `NaN` in JavaScript?
- What is DOM in JavaScript?
- What is the event loop in JavaScript?
- What is the difference between a function declaration and a function expression?
- What is the purpose of `bind()`, `call()`, and `apply()` in JavaScript, and how are they different?

MODULE 4

Javascript and HDML Documents

JavaScript and DHTML Documents

JavaScript:

- JavaScript is a high-level, dynamic programming language primarily known for its use in web development.
- It is commonly used to make web pages interactive and dynamic.
- JavaScript can be embedded directly into HTML pages or included from external script files.
- It's supported by all major web browsers, making it a versatile language for client-side scripting.

Example of embedding JavaScript in HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
</head>
<body>

<h1>Hello, World!</h1>

<script>
  // JavaScript code alert("This is
  JavaScript!");
</script>

</body>
</html>
```

DHTML (Dynamic HTML):

- DHTML refers to a combination of HTML, CSS, and JavaScript used to create dynamic and interactive web pages.
- It allows for the manipulation of HTML elements in real-time, creating a more

responsive user experience.

- DHTML is often associated with features like animations, dynamic content updates, and user interface enhancements.

Example of DHTML using JavaScript to manipulate HTML elements:

```
<!DOCTYPE html>
<html>
<head>
<title>DHTML Example</title>
<style>
    /* CSS styles for dynamic changes */
    #myDiv {
        width: 100px;
        height: 100px;
        background-color: lightblue;
        transition: width 1s, height 1s, background-color 1s;
    }
</style>
</head>
<body>

<div id="myDiv" onclick="changeStyle()">Click me!</div>

<script>
    // JavaScript code for dynamic changes
    function changeStyle() {
        var div = document.getElementById("myDiv");
        div.style.width = "200px";
        div.style.height = "200px";
        div.style.backgroundColor = "lightcoral";
    }
</script>
```

```
</body>
</html>
```

The document object model

The Document Object Model (DOM) in JavaScript is a programming interface that represents the structure of a document as a tree of objects. The document can be an HTML or XML document, and the DOM provides a way for scripts to dynamically access and manipulate the content, structure, and style of a document.

Here are some key concepts related to the DOM in JavaScript:

Document Object:

- The top-level object in the DOM hierarchy is the document object. It represents the entire HTML or XML document and provides methods and properties for interacting with the document.

```
// Accessing the document object var doc =
document;
```

DOM Tree:

- The DOM represents a document as a tree structure, where each node in the tree corresponds to an element, attribute, or piece of text in the document.

```
<!-- Example HTML document -->
<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a sample paragraph.</p>
```

```
</body>
</html>
```

Node:

- Nodes are the building blocks of the DOM tree. Everything in the DOM is a node, including elements, attributes, and text. Nodes can be accessed and manipulated using JavaScript.

// Accessing nodes in the DOM

```
var heading = document.getElementsByTagName('h1')[0];
```

Element:

- Elements are a specific type of node representing HTML or XML elements in the DOM tree. They have properties and methods for interacting with their attributes and content.

```
// Accessing element properties
```

```
var title = document.title; DOM
```

Manipulation:

- You can dynamically modify the content and structure of a document using JavaScript. Common operations include creating, deleting, and modifying elements.

```
// Creating a new element and appending it to the document
var newParagraph = document.createElement('p');
newParagraph.textContent = 'This is a new paragraph.';
document.body.appendChild(newParagraph);
```

Events:

- The DOM allows you to handle events such as clicks, keypresses, and more. You can use event listeners to respond to user interactions.

```
// Adding a click event listener to a button
```

```
var myButton = document.getElementById('myButton');
myButton.addEventListener('click', function() { alert('Button clicked!');
});
```

Example (JavaScript):

```
<!DOCTYPE html>

<html>

<head>

<title>DOM Example</title>

</head>
<body>

<h1 id="myHeading">Hello, DOM!</h1>

<script>

// JavaScript code to manipulate the DOM

var heading = document.getElementById("myHeading"); heading.innerHTML = "Hello, Updated
DOM!";
</script>

</body>

</html>
```

In this example, JavaScript is used to select an element with the ID "myHeading" and change its content using the innerHTML property.

The DOM is a crucial part of web development, enabling the creation of interactive and dynamic web pages. Developers use it to respond to user actions, update content, and create rich web applications.

DOM Tree

<!DOCTYPE html>	- Document
<html>	- Element: html
<head>	- Element: head
<title>Sample Document</title>	- Element: title
</head>	- Text: "Sample Document"

<body>	- Element: body
<h1>Hello, World!</h1>	- Element: h1
<p>This is a sample paragraph.</p>	- Text: "Hello, World!"
</body>	- Element: p
</html>	- Text: "This is a sample paragraph."

Element Access in JavaScript

In JavaScript, you can access elements in various ways, depending on the type of data structure you are working with. Here are some common ways to access elements:

Arrays:

Accessing elements by index:

```
let myArray = [1, 2, 3, 4, 5];
let firstElement = myArray[0]; // Access the first element (index 0)
let thirdElement = myArray[2];
// Access the third element (index 2)
```

Objects:

Accessing properties by name:

```
let myObject = { name: 'John', age: 25, city: 'New York' };
```

```
let personName = myObject.name; // Access the 'name' property
let personAge = myObject['age']; // Another way to access the 'age'
                                property
```

Strings:

Accessing characters by index:

```
let myMap = new Map(); myMap.set('key1',
'value1');
myMap.set('key2', 'value2');
let valueForKey1 = myMap.get('key1'); // Access the value associated with 'key1'
```

Accessing elements in a Set:

```
let mySet = new Set([1, 2, 3, 4, 5]);
let hasElement = mySet.has(3); // Check if the Set contains the element 3
```

In summary, use a Set when you need a unique collection of values, and use a Map when you need to associate values with specific keys and maintain the order of insertion.

Events and event handling

In JavaScript, events are actions or occurrences that happen in the browser, such as a user clicking a button, a page finishing loading, or a key being pressed. Event handling is the process of writing code to respond to and manage these events. Event handling is crucial for creating interactive and dynamic web pages.

Here's an overview of events and event handling in JavaScript:

Events:

Event Types:

- **Mouse Events:**
- click, mousedown, mouseup, mousemove, mouseenter, mouseleave, etc.
- **Keyboard Events:**

- keydown, keyup, keypress
- **Form Events:**
- submit, reset, change, input
- **Document and Window Events:**
- load, unload, resize, scroll

Common DOM Events:

- **Click:** Triggered when a mouse button is pressed and released on an element.
- **Mouseover/Mouseout:** Fired when the mouse pointer enters/leaves an element.
- **Keydown/Keyup:** Occurs when a keyboard key is pressed or released while the element has focus.
- **Load/Unload:** Fired when the page finishes loading or unloading.

Event Propagation:

Events propagate through the DOM tree in two phases: capturing phase (from the top of the tree to the target element) and bubbling phase (from the target element back up to the top). You can use the `addEventListener` method to specify which phase you want to handle.

Event Object Properties:

- The event object provides various properties and methods, such as:
- `event.target`: The element that triggered the event.
- `event.type`: The type of the event (e.g., 'click', 'keydown').
- `event.preventDefault()`: Prevents the default action of the event.
- `event.stopPropagation()`: Stops the event from propagating further.

Event Handling:

Inline Event Handlers:

You can attach event handlers directly in HTML tags using inline attributes like `onclick`, `onmouseover`, etc.

```
<button onclick="myFunction()">Click me</button>
```

DOM Level 0 Event Handling:

- Assign a function to the event property of an element.

DOM level 0 event handlers include: `onload` : This event is triggered when a webpage has finished loading. `onmouseout` : This event is triggered when a user moves their mouse out of an element. `onsubmit` : This event is triggered when a form is submitted.

```
let button = document.getElementById('myButton'); button.onclick = function() {  
    alert('Button clicked!');  
};
```

DOM Level 2 Event Handling:

- Use the `addEventListener` method to attach event handlers. This method provides more flexibility, allowing multiple handlers for the same event.

```
let button = document.getElementById('myButton');  
button.addEventListener('click', function() { alert('Button clicked!');  
});
```

Event Object:

- Event handlers receive an event object as a parameter, containing information about the event (e.g., mouse coordinates, key pressed). This object allows you to interact with the event and its properties.

Event Listeners Removal:

To remove an event listener, use the `removeEventListener` method. Ensure the same function reference is used for removal. **`function myFunction() {`**

```
    console.log('Event handled!');  
}  
element.addEventListener('click', myFunction);  
// Removing the event listener element.removeEventListener('click', myFunction);
```

Moving elements

In JavaScript, you can move an HTML element on a webpage by changing its CSS properties, such as

left, top, right, and bottom. You can achieve this by using the style property of the element and updating its positioning values.

Here's how to move an element in JavaScript using DOM manipulation:

1. Select the element you want to move using the document. ...
2. Select the new parent element where you want to move the element to using the same document. ...
3. Use the `appendChild()` method of the new parent element to move the element to its new location.

In JavaScript, you can move HTML elements within the DOM (Document Object Model) using various methods and properties. Here are some common approaches to move elements:

1. `appendChild` and `insertBefore`:

Both `appendChild` and `insertBefore` are methods used to manipulate the DOM by moving or inserting elements. Let's explore each of them in detail:

1. `appendChild` Method:

The `appendChild` method is used to append a node as the last child of a specified parent node. It's commonly used when you want to move an element to the end of another element.

Example:

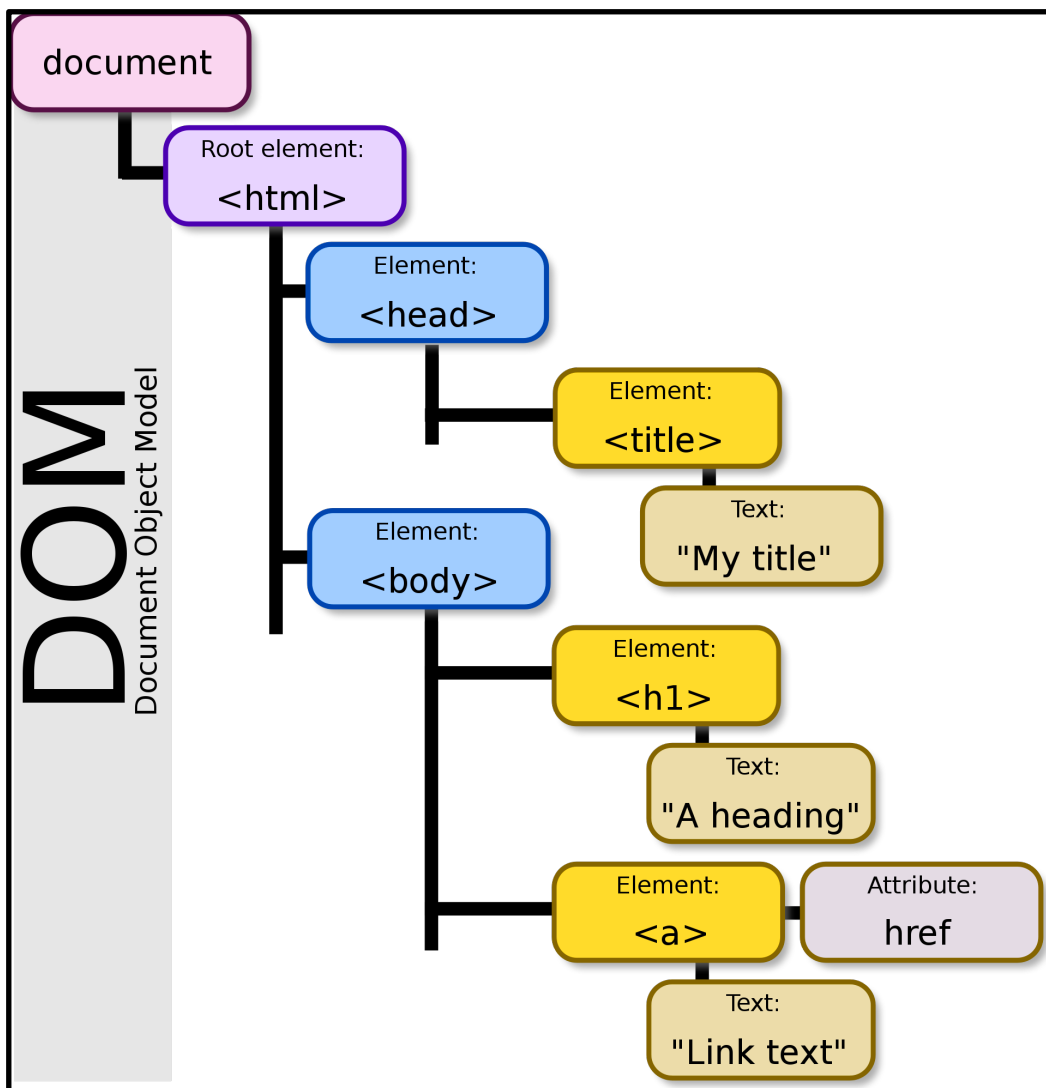
```
<!DOCTYPE html>
<html lang="en">
<head>
<title>appendChild Example</title>
<style>
.container {
border: 1px solid #ccc; padding: 10px;
}
</style>
</head>
<body>
<div id="parent1" class="container">
```

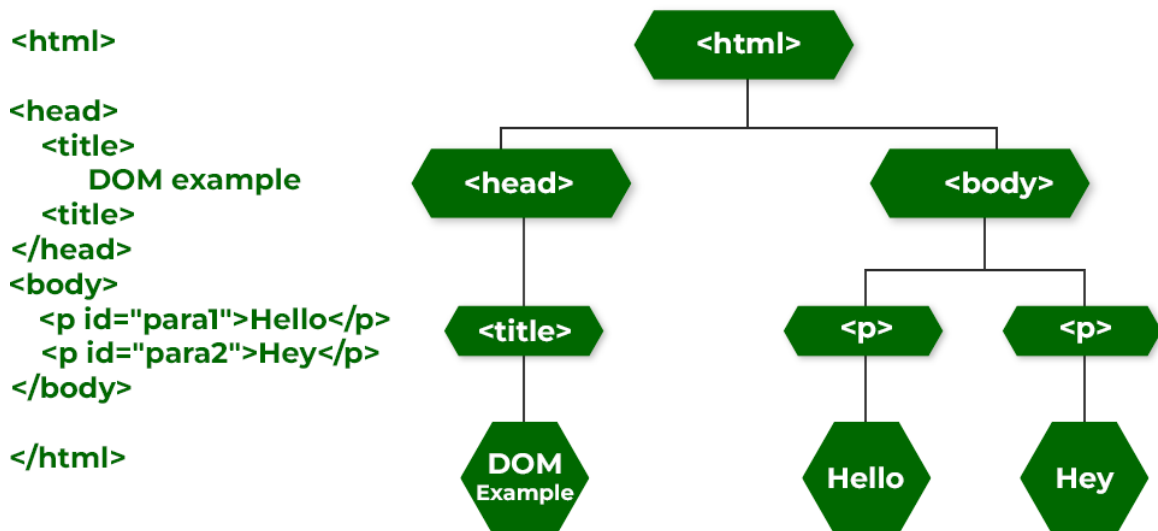
```
<p>Parent 1</p>
</div>
```

```
<div id="parent2" class="container">
<p>Parent 2</p>
</div>
```

```
<script>
// JavaScript
let parent1 = document.getElementById('parent1'); let parent2 = document.getElementById('parent2');
let elementToMove = document.createElement('div'); elementToMove.textContent = 'Element to
Move';

// Append the element to the end of parent1 parent1.appendChild(elementToMove);
</script>
</body>
</html>
```





Element Visibility

In the context of web development and the Document Object Model (DOM), "element visibility" refers to whether an HTML element is currently visible or hidden on a webpage. The visibility of an element can be controlled using CSS or JavaScript. Here are some common ways to handle element visibility:

Visibility can be set with values visible, hidden, collapse, initial, inherit.

- **visible:** The property value is set as visible to show contents on the screen. By default property value set to visible.
- **hidden:** This property value is to hide the contents. Here the element is hidden but still occupies the space on the screen.
- **collapse:** Collapse is the property value only going to be used with the table element. It's mainly used to remove a particular row or column. So the table's layout will remain as it is. By using collapse property in the table, the space occupied by row and column will get free for other elements in the layout.
- **initial:** This property value is used to the default value.
- **inherit:** It's one of the property values used o inherit contents from its parent

element.

CSS

A. Display Property:

- The displayproperty can be used to control the visibility of an element.
- display: none;will hide the element, while display: block;or other display values will make it visible.

```
#myElement {  
    display: none; /* Hide the element */  
}
```

B. Visibility Property:

- **The visibilityproperty can be used to hide an element without affecting the layout.**

```
#myElement {  
    visibility: hidden; /* Element is hidden, but it still takes up space in the layout */  
}
```

JavaScript

1. classList Property:

- You can toggle classes on an element to control its visibility through CSS rules.

```
// Hide the element document.getElementById("myElement").classList.add("hidden");  
  
// Show the element document.getElementById("myElement").classList.remove("hidden");
```

2. Style Property:

- You can directly manipulate the styleproperty of an element using JavaScript.

```
// Hide the element document.getElementById("myElement").style.display = "none";
```

```
// Show the element document.getElementById("myElement").style.display = "block";
```

3. Visibility Property:

- Similar to the CSS approach, you can use the visibility property in JavaScript.

```
// Hide the element document.getElementById("myElement").style.visibility = "hidden";
```

```
// Show the element document.getElementById("myElement").style.visibility = "visible";
```

Dynamic content in javascript

Dynamic content in JavaScript refers to the ability of JavaScript to manipulate the content of a web page dynamically. Instead of having static, fixed content, dynamic content can be updated or modified in response to user actions, events, or other changes in the application state. This enhances the interactivity and responsiveness of web pages.

Dynamic content loading refers to the process of updating or loading content on a web page without requiring a full page refresh. JavaScript plays a vital role in achieving dynamic content loading by facilitating asynchronous data retrieval and manipulation.

Here are key aspects of dynamic content in JavaScript:

DOM Manipulation:

- JavaScript interacts with the Document Object Model (DOM), which represents the structure of an HTML document. Through DOM manipulation, you can add, remove, or

modify HTML elements and their attributes. This allows for dynamic updates to the content displayed on a webpage.

Event Handling:

- Dynamic content often involves responding to user interactions, such as clicks, key presses, or form submissions. JavaScript allows you to attach event listeners to specific HTML elements, enabling you to execute code in response to these events. For example, you might update the content of a paragraph when a button is clicked.

```
// Example: Event handling to update content on button click
document.getElementById('myButton').addEventListener('click', function() {
    document.getElementById('myParagraph').innerHTML = 'New content!';
});
```

AJAX (Asynchronous JavaScript and XML):

- AJAX is a technique that allows you to fetch data from a server without requiring a full page reload. With AJAX, you can dynamically update content on a page by making asynchronous requests to a server, receiving data in various formats (often JSON), and then updating the DOM accordingly. **Dynamic Data Binding:**
- Modern JavaScript frameworks, such as React, Angular, or Vue, facilitate dynamic data binding. Data binding enables automatic synchronization between the application's data model and the UI, so changes to the data automatically reflect in the displayed content, and vice versa.

Slow movement of element in javascript

Such slow movement of elements is accomplished by using two window methods available in JavaScript: `setTimeout` and `setInterval`. The `setTimeout` method has two parameters -- first one is string of JavaScript code to be executed and second parameter is number of mseconds of delay before executing the given script.

1. Animation Basics:

Animation involves transitioning an element from one state to another over a period of time. In web development, animations can be achieved through various techniques, and one common approach is to manipulate the CSS properties of an element.

2. JavaScript Timers:

To create movement over time, JavaScript utilizes timers. The two primary timer functions are:

- `setInterval(callback, delay)`: Repeatedly calls a function or executes a code snippet at a specified interval (in milliseconds).
- `setTimeout(callback, delay)`: Invokes a function or executes a code snippet after a specified delay.

3. Updating Element Properties:

To achieve movement, you typically update CSS properties such as `top`, `left`, `margin`, or `transform` over time. This creates the illusion of motion as the browser renders these changes.

4. Smooth Animation:

For smoother animations, the interval or timeout duration should be small. However, too frequent updates can affect performance.

Balancing smoothness and performance is crucial.

5. Slow Movement Using `setInterval`:

In the provided example:

- An element's initial position is set.
- `setInterval` is used to update the position at regular intervals.
- The movement stops when a certain condition (e.g., reaching a specific position) is met.

6. CSS Transitions and Animations:

While JavaScript can be used for basic animations, CSS transitions and animations provide a more declarative and efficient way to handle visual effects. These CSS features can be triggered by adding or removing classes with JavaScript.

7. Performance Considerations:

When implementing animations, it's crucial to consider performance. Excessive use of timers or frequent updates to DOM elements can lead to janky animations and impact user experience negatively.

8. Modern Approaches:

Many modern frameworks like React, Vue, and libraries like GSAP provide more sophisticated tools for animations, making it easier to create complex and performant animations.\

Question Bank

1.Explain in detail about embedding JavaScript in HTML

2. Explain the following
 - a. DHTML
 - b. The document object model
3. Illustrate the following
 - a. DOM Tree
 - b. Node
 - c. Element
4. Explain in detail about Events and Event handling.
5. Write in detail about Slow movement of element in javascript.
6. Summarize in detail about Element visibility.
7. Illustrate in detail about Dynamic content.
8. Write about Asynchronous JavaScript and XML
9. Explain DOM Manipulation with suitable example.
10. Mention how to move an element in JavaScript using DOM manipulation

MODULE 5

Basics of PHP and XML

Origins and uses of PHP

PHP, which stands for Hypertext Preprocessor, is a widely used server-side scripting language designed for web development. It has a long history and has evolved significantly since its inception. Here are the origins and uses of PHP:

Origins of PHP:

- **Creation by Rasmus Lerdorf (1994):** PHP was originally created by Danish- Canadian programmer Rasmus Lerdorf in 1994. It began as a set of Common Gateway Interface (CGI) binaries written in the C programming language to track visits to his online resume.
- **PHP/FI (1995):** Rasmus released PHP/FI (Personal Home Page/Forms Interpreter) in 1995, which included a wider range of functionality. It allowed the creation of simple dynamic web pages and the ability to interact with databases.
- **Introduction of PHP 3 (1998):** The next major release was PHP 3, which introduced a more robust language structure and support for interacting with databases like MySQL.
- **Zend Engine (1999):** PHP 4 incorporated the Zend Engine, a scripting engine for PHP written by Andi Gutmans and Zeev Suraski. This significantly improved the performance and capabilities of PHP.
- **PHP 5 and Object-Oriented Programming (2004):** PHP 5 introduced extensive support for object-oriented programming (OOP) and brought improvements in performance, error handling, and support for XML.
- **PHP 7 (2015):** PHP 7 marked a significant leap in performance with the introduction of the Zend Engine 3. It brought improvements in speed, memory usage, and introduced features like scalar type declarations and the spaceship operator.
- **PHP 8 (2020):** PHP 8 continued the evolution of the language, introducing features like the Just-In-Time (JIT) compiler, union types, named arguments, and more.

Uses of PHP:

- **Web Development:**
 1. **Server-Side Scripting:** PHP is primarily used for server-side scripting to generate

dynamic web pages. It can be embedded directly into HTML code.

2. Web Applications: PHP is used to develop web applications, from simple websites to complex content management systems (CMS) like WordPress, Joomla, and Drupal.
- Database Interaction:
 1. MySQL Integration: PHP has strong support for interacting with MySQL databases. It is frequently used to perform database operations, such as inserting, updating, and retrieving data.
 - User Authentication and Authorization:
 1. PHP is commonly used to implement user authentication systems, managing user sessions, and controlling access to various parts of a website or web application.
 - Server-Side Utilities:
 1. File Handling: PHP can be used for file handling and manipulation on the server, allowing applications to read, write, and modify files.
 - API Development:
 1. PHP can be used to develop server-side components of RESTful APIs, allowing communication between different web services and applications.
 - Command-Line Scripting:
 1. PHP can be executed from the command line, making it suitable for creating command-line scripts and automation tasks.
 - Content Management Systems (CMS):
 1. Many popular CMS platforms, such as WordPress, are built using PHP. It allows developers to create customizable and dynamic websites without extensive coding.
 - E-commerce Applications:
 1. PHP is widely used in the development of e-commerce applications, enabling the creation of online stores, shopping carts, and payment processing systems.

Overview of PHP

PHP, which originally stood for "Personal Home Page," is a widely used server-side scripting language designed for web development. Over the years, its meaning has evolved to "Hypertext Preprocessor." Here's an overview of PHP:

Key Characteristics:

- **Server-Side Scripting:** PHP is a server-side scripting language, meaning it is executed on the server before the result is sent to the client's web browser. It is used to generate dynamic content, interact with databases, and perform various server-side tasks.
- **Open Source:** PHP is an open-source language, and its source code is freely available. This fosters collaboration and community-driven development, resulting in continuous improvements and updates.
- **Cross-Platform Compatibility:** PHP is platform-independent and can run on various operating systems, including Windows, macOS, Linux, and more. It is compatible with different web servers, such as Apache, Nginx, and Microsoft IIS.
- **Interoperability:** PHP can easily integrate with various databases (MySQL, PostgreSQL, SQLite), web servers, and other technologies. It supports numerous protocols, making it versatile for web development.
- **Ease of Learning:** PHP has a syntax that is relatively easy to learn, especially for those with a background in C-style languages. Its simplicity makes it accessible for beginners while providing powerful features for advanced users.
- **HTML Embedded:** PHP code is embedded directly within HTML, allowing developers to mix dynamic server-side logic with static HTML content. This makes it convenient for creating dynamic web pages.
- **Extensibility:** PHP supports extensions and modules that enhance its functionality. These extensions cover a wide range of tasks, from image processing to encryption and beyond.
- **Large Community:** PHP has a vast and active community of developers, which contributes to extensive documentation, libraries, frameworks, and support forums. The community-driven nature ensures ongoing support and improvement.

Evolution:

- **PHP 3 (1998):** Initial version with increased functionality beyond its original purpose of tracking online resume views.
- **PHP 4 (2000):** Introduction of the Zend Engine, providing better performance and additional features.
- **PHP 5 (2004):** Significant improvements, including support for object-oriented programming (OOP), better error handling, and enhanced XML support.
- **PHP 7 (2015):** Major performance improvements, reduced memory usage, and the

introduction of features like scalar type declarations, return type declarations, and the spaceship operator.

- PHP 8 (2020): Continued improvements with the introduction of the Just-In-Time(JIT) compiler, union types, named arguments, and other enhancements.

Use Cases:

- Web Development: PHP is a cornerstone of web development, commonly used for building websites, web applications, and dynamic content.
- Content Management Systems (CMS): Many popular CMS platforms, including WordPress, Joomla, and Drupal, are built with PHP.
- E-commerce: PHP is widely used in developing e-commerce solutions, facilitating the creation of online stores, shopping carts, and payment processing systems.
- Server-Side Scripting: PHP is employed for various server-side scripting tasks, such as form processing, user authentication, and data manipulation.
- API Development: PHP can be used to develop the server-side components of RESTful APIs, enabling communication between different services and applications.
- Command-Line Scripting: PHP scripts can be executed from the command line, making it suitable for automation and scripting tasks.

General syntactic characteristics

PHP has a syntax that is easy to learn, especially for those with a background in C-style languages. Here are some general syntactic characteristics of PHP:

1. Script Tags:

- PHP code is enclosed in `<?php` and `?>` tags.
- Shorter opening tags (`<?`) and short echo tags (`<?=>`) are also available but may be disabled in some environments.

`<?php`

`// PHP code here`

?>

Comments:

- PHP supports both single-line (//) and multi-line (/* */) comments.

<?php

// This is a single-line comment

/*

This is a multi-line comment
spanning multiple lines

*/

?>

Variables:

- Variable names start with a \$sign followed by the name.
- Variable names are case-sensitive.

<?php

\$name = "John";

\$age = 25;

?>

Data Types:

- PHP has dynamic typing, meaning you do not need to declare the data type of a variable.
- Common data types include strings, integers, floats, booleans, arrays, and objects.

<?php

\$name = "John"; // String

```

$age = 25;           // Integer
$height = 5.9;       // Float
$isStudent = true; // Boolean
$colors = array("Red", "Green", "Blue"); // Array
?>

```

Strings:

- Strings can be declared using single (') or double (") quotes.
- Variable interpolation is possible in double-quoted strings.

```

<?php
$name = "John";
echo 'Hello, ' . $name . '!';           // Single-quoted
echo "Hello, $name!";                   // Double-quoted with variable interpolation
?>

```

Operators:

- PHP supports various operators, including arithmetic, comparison, logical, and string concatenation.

```

<?php
$result = 10 + 5;           // Addition
$isGreater = 10 > 5; // Comparison
$logicalAnd = true && false; // Logical AND
$concatenated = "Hello " . "World"; // String concatenation
?>

```

Control Structures:

- PHP supports common control structures like if statements, loops (for, while, do-while), and switch statements.

```

<?php
    $x = 10;

    if ($x > 0) {
        echo "Positive";
    } else {
        echo "Non-positive";
    }

    // Loop
    for ($i = 0; $i < 5; $i++)
        {echo $i;
    }
?>

```

Functions:

- Functions are declared using the `function` keyword.
- Function names are case-insensitive.

```

<?php
    function greet($name)
    { echo "Hello, $name!";
    }

    greet("John");
?>

```

Arrays:

- Arrays can be indexed or associative.

- PHP provides many array functions for manipulation.

```
<?php
```

```
$colors = array("Red", "Green", "Blue"); echo
```

```
$colors[0]; // Accessing array elements
```

```
$person = array("name" => "John", "age" => 25); echo
```

```
$person["name"]; // Associative array
```

```
?>
```

Include and Require:

- PHP includes other PHP files using include or require.
- include produces a warning if the file is not found, while require produces a fatal error.

```
<?php
```

```
include "header.php"; require
```

```
"functions.php";
```

```
?>
```

Output in php

With echo variables

```
<?php
```

```
$name = "John"; echo
```

```
"Hello, $name!";
```

```
// or
```

```
echo 'Hello, ' . $name . '!';
```

```
?>
```

With Html Variable

```
<?php
```

```
echo "<h1>This is a heading</h1>"; echo  
"<p>This is a paragraph.</p>";  
?>
```

Control statements in php

In PHP, control statements are used to control the flow of execution in a script. They allow you to make decisions, repeat actions, and perform different actions based on conditions. Here are some of the key control statements in PHP:

if Statement:

The if statement is used to execute a block of code only if a specified condition is true.

```
<?php  
$x = 10;  
  
if ($x > 5) {  
    echo "x is greater than 5";  
}  
?>
```

if-else Statement:

The if-else statement allows you to execute one block of code if the condition is true and another block if the condition is false.

```
<?php  
$x = 10;  
  
if ($x > 5) {  
    echo "x is greater than 5";  
} else {
```

```
        echo "x is not greater than 5";  
    }  
?>
```

if-elseif-else Statement:

This statement allows you to test multiple conditions and executedifferent blocks of code based on the first condition that is true.

```
<?php  
$x = 10;  
  
if ($x > 10) {  
    echo "x is greater than 10";  
} elseif ($x == 10) {  
    echo "x is equal to 10";  
} else {  
    echo "x is less than 10";  
}  
?>
```

switch Statement:

The switch statement is used to perform different actions based on different conditions.

```
<?php
$day = "Monday";

switch ($day) { case
    "Monday":
        echo "It's the start of the week";break;
    case "Friday":
        echo "It's almost the weekend";break;
    default:
        echo "It's just another day";
}
?>
```

while Loop:

The while loop executes a block of code as long as the specified condition is true.

```
<?php
$i = 1;

while ($i <= 5) { echo
    $i . " ";
    $i++;
}
?>
```

for Loop:

The for loop is used to iterate a block of code a specified number of times.

```
<?php
for ($i = 1; $i <= 5; $i++) {echo $i . "
```



```

        ".
    }
?>

```

foreach Loop:

The foreachloop is used to iterate over elements in an array.

```

<?php
$colors = array("red", "green", "blue");

foreach ($colors as $color) {echo $color .
    " ";
}
?>

```

Arrays in php

In PHP, an array is a versatile data structure that allows you to store multiple values in a single variable. Each value in an array is assigned a unique key, which can be either an index (numeric or associative) or a string. Here are some fundamental aspects of working with arrays in PHP:

1. Numeric Arrays:

Numeric arrays use sequential numbers as keys.

```

<?php

$numericArray = array(1, 2, 3, 4, 5);

// or, using the shorthand syntax in PHP 5.4 and later

$numericArray = [1, 2, 3, 4, 5];

// Accessing elements

```

```
echo $numericArray[0]; // Outputs 1
```

```
?>
```

Associative Arrays:

Associative arrays use named keys.

```
<?php
```

```
$assocArray = array( "name"
```

```
    => "John", "age" => 30,
```

```
    "city" => "New York"
```

```
);
```

```
// or, using the shorthand syntax
```

```
$assocArray = [ "name" =>
```

```
    "John", "age" => 30,
```

```
    "city" => "New York"
```

```
];
```

```
// Accessing elements
```

```
echo $assocArray["name"]; // Outputs John
```

Multidimensional Arrays:

Arrays can contain other arrays, creating multidimensional arrays.

```
<?php
```

```
$multiArray = array(array(1,  
    2, 3),  
    array("apple", "orange", "banana"),  
  
    array("a", "b", "c")  
  
);
```

```
// Accessing elements
```

```
echo $multiArray[0][1]; // Outputs 2
```

```
echo $multiArray[1][2]; // Outputs banana
```

Array Functions:

PHP provides a variety of built-in functions to manipulate arrays.

- `count()`: Returns the number of elements in an array.
- `array_push()`: Adds one or more elements to the end of an array.
- `array_pop()`: Removes and returns the last element of an array.
- `array_merge()`: Merges one or more arrays into a single array.

```
<?php
```

```
$numbers = [1, 2, 3];
```

```
// Adding an element
```

```
array_push($numbers, 4);
```

```
// Removing the last element
```

```
$lastElement = array_pop($numbers);
```

```
// Merging arrays  
$newArray = array_merge($numericArray, $assocArray);  
?>
```

Functions in PHP

In PHP, a function is a block of reusable code that performs a specific task. Functions help in organizing code, making it more modular and easier to maintain. Here's a guide to creating and using functions in PHP:

Defining a Function:

You can define a function using the `function` keyword. The basic syntax is as follows:

```
<?php
function greet($name)
{ echo "Hello, $name!";
}

greet("John"); // Outputs: Hello, John!
?>
```

Function Parameters:

Functions can accept parameters (input values). These parameters are listed within the parentheses after the function name

```
<?php

function add($a, $b) {

    $sum = $a + $b; return
    $sum;
}

$result = add(5, 3); // $result now holds the value 8
```

?>

Variable Scope:

Variables defined inside a function have local scope, meaning they are only accessible within that function. You can use the global keyword to access global variables.

<?php

```
$globalVar = "I'm global";
```

```
function example() { global  
    $globalVar; echo  
    $globalVar;  
}  
/example(); // Outputs: I'm global
```

Pattern Matching in php

In PHP, you can use various functions and techniques for pattern matching. Here are some commonly used methods:

1. Regular Expressions:

PHP supports regular expressions through the `preg_match()` function. Regular expressions allow you to define complex search patterns. Here's a simple example:

```
$text = "Hello, World!";  
  
if (preg_match("/Hello/", $text))  
    { echo "Pattern found!"; } else  
    { echo "Pattern not found!";  
    }
```

Regular Expressions in PHP:

Regular expressions, often abbreviated as regex or regexp, are sequences of characters that define a search pattern. They are particularly useful for complex pattern matching and manipulation of strings. PHP supports regular expressions through the `preg_*` functions.

Here's a more detailed explanation using the `preg_match()` function:

`preg_match()` Function:

The `preg_match()` function is used to perform a pattern match on a string using a regular expression. It returns true if the pattern is found in the string, and false otherwise.

The `preg_match()` function returns true because the pattern "Hello" is found in the given text.

Regular Expression Modifiers:

Regular expressions can include modifiers that affect the matching behavior. For example, the "i" modifier makes the pattern case-insensitive:

```
$text = "Hello, World!";

if (preg_match("/hello/i", $text))
    {echo "Pattern found!";
} else {

    echo "Pattern not found!";

}
```

Capturing Groups:

Regular expressions can include capturing groups to extract specific parts of a matched string. For example:

```
$phone_number = "Phone: 123-456-7890";
```

```
if (preg_match("/Phone: (\d{3}-\d{3}-\d{4})/", $phone_number, $matches))  
    {echo "Phone number found: " . $matches[1];  
    } else {  
        echo "Phone number not found!";  
    }  
}
```

2. Pattern Matching with Wildcards:

The `fnmatch()` function is used for simple pattern matching with wildcards:

```
<?php  
$input = "user123@example.com"; if  
(fnmatch("abc*", $input)) {  
    echo "Matched user email prefix!";  
} else {  
    echo "No match found.";  
}
```

```
?>
```

3. String Functions:

PHP provides various string functions that can be used for basic pattern matching. For instance, you can use `strpos()` to check if a substring is present in another string:

```
$haystack = "Hello, World!";  
$needle = "World";  
$position = strpos($haystack, $needle);  
  
if ($position !== false) {  
    echo "Substring found at position: $position";  
} else {  
    echo "Substring not found!";  
}
```



```
}
```

4. Pattern Matching with strstr()and stristr():

The strstr()function is used to find the first occurrence of a substring, and stristr() is case-insensitive. Both can be used for simple pattern matching:

```
$var1= "Hello, World!";
```

```
$var2= "World";
```

```
$restOfString = strstr($var1, $var2); if  
($restOfString !== false) {  
    echo "Substring found: $restOfString";  
  
} else {  
  
    echo "Substring not found!";  
  
}
```

Form Handling in PHP

HTML Forms:

HTML forms are used to collect and submit user input on web pages. They are created using the <form> element and can contain various types of input elements such as text fields, radio buttons, checkboxes, and buttons. The action attribute in the form specifies the URL to which the form data will be sent, and the method attribute determines how the data will be submitted (GET or POST).

```
<form action="process_form.php" method="post">
```

HTTP Methods:

- GET Method:
 - Submits form data as part of the URL.
 - Limited data capacity (URL length restrictions).
 - Data is visible in the URL.

- Suitable for non-sensitive data retrieval.
- POST Method:
 - Submits form data in the request body.
 - No URL length limitations.
 - Data is not visible in the URL.
 - Suitable for sensitive data and larger amounts of data.

PHP Form Handling:

Superglobals:

- `$_GET`: Retrieves form data submitted with the GET method.
- `$_POST`: Retrieves form data submitted with the POST method.
- `$_REQUEST`: Retrieves data from both GET and POST methods.

```
$name = $_POST["name"];
$email = $_POST["email"];
```

Form Processing in PHP:

1. Checking Form Submission:

- Use `if ($_SERVER["REQUEST_METHOD"] == "POST")` to check if the form is submitted.

2. Retrieving Form Data:

- Use `$_POST` to retrieve form data.

3. Validation:

- Validate form data to ensure it meets required criteria (e.g., required fields, valid email addresses).
- Use functions like `empty()`, `isset()`, and regular expressions for validation.

Handling Form Submission:

Processing Logic:

- Implement the logic for processing form data (e.g., saving to a database, sending emails).
- Perform necessary operations based on the application requirements. `if`

```
($_SERVER["REQUEST_METHOD"] == "POST")
```

Redirection After Form Submission:

- After processing form data, consider redirecting users to another page.
- This helps prevent form resubmission when users refresh the page.

Files in PHP

PHP provides various functions to handle files. Some common file-related operations include reading from and writing to files, checking file existence, deleting files, etc. Here are some examples:

- Reading from a file:

```
$fileContent = file_get_contents('example.txt');
```

Writing to a file:

```
$data = "Hello, World!";
```

```
file_put_contents('example.txt', $data);
```

Checking file existence:

```
if (file_exists('example.txt'))  
    {echo 'File exists!';  
} else {  
    echo 'File does not exist.';  
}
```

Deleting a file:

```
unlink('example.txt');
```

Appending to a file:

```
$data = "Appended content.";
file_put_contents('example.txt', $data, FILE_APPEND);
```

Cookies in PHP

Cookies are small pieces of data that can be stored on the user's computer. They are commonly used to store user preferences, session information, and other data.

Setting a cookie:

```
setcookie('user', 'John Doe', time() + 3600, '/');
```

In this example, a cookie named 'user' is set with the value 'John Doe', and it will expire in one hour (time() + 3600). The last parameter '/' indicates that the cookie is available across the entire domain.

Accessing a cookie:

```
$username = $_COOKIE['user'];
```

This retrieves the value of the 'user' cookie and assigns it to the \$username variable.

Deleting a cookie:

```
setcookie('user', "", time() - 3600, '/');
```

This deletes the 'user' cookie by setting its expiration time to the past.

Checking if a cookie exists:

```
if (isset($_COOKIE['user']))
    {echo 'Cookie exists!';
} else {
    echo 'Cookie does not exist.';
```

}

Difference between JavaScript and PHP

JavaScript	PHP
Currently, it is a full-stack programming language. It means it can serve both the client as well as server sites.	It is a server-side scripting language. It serves only the
It is supportable by every web browser, such as Mozilla, Google Chrome, and many more.	It is supportable on Windows, Linux, Mac, etc. platforms .
It carries less securable code.	PHP code is highlysecurable.
It requires an environment for accessing the database.	It allows easy and direct access to the
The code is written within <script>...</script> tags.	The code is written within <?php....?> tag.
Earlier, javascript was able to create interactive pages for the clients. But, at present, it is able to build real-time games and applications, mobile applications, too.	A PHP program is able to generate dynamic pages, send cookies, receivecookies, collect form data, etc.
The extension used to save an external javascript file is '.js'.	Files are saved using'.php' extension.
We can embed the js code in HTML, XML, and AJAX.	We can embed the PHP code only with HTML.
It supports fewer features.	It supports more advanced features as compared to JavaScript.

Popular frameworks of JavaScript are Angular, React, Vue.js, Meteor, etc.	Popular frameworks of PHP are Laravel, Symfony, FuelPHP, CakePHP, and many more.
Websites built in JavaScript are Twitter, LinkedIn, Amazon, etc.	Websites built in PHP are Wordpress , Tumblr, MailChimp, iStockPhoto, etc.

QUESTION BANK

1. Write the General syntactic characteristics of PHP.
2. Explain in detail about the Origins and uses of PHP.
3. Summarize in detail about Control statements in php.
4. Illustrate about the different Functions in PHP.
5. Discuss in detail about Pattern Matching in PHP.
6. Illustrate the following:a.HTML Forms b. HTTP Methods.
7. Explain in detail about Form Processing in PHP.
8. Discuss in detail about Files in PHP.
9. Describe in detail about Cookies in PHP.
10. Write the Differences between Javascript and PHP.