Module 2: Introduction to CSS, Levels of CSS, Selectors, Font, Color and Text Properties, Box Model, Introduction to JavaScript, JavaScript variables, operators, Conditional and loop statements in JavaScript, Functions and Arrays in JavaScript, Validations using JavaScript, Screen output and Keyboard input

CASCADING STYLE SHEETS (CSS)

A **stylesheet** is a set of CSS rules used to control the layout and design of a webpage or document. HTML style sheets are called *cascading* style sheets because they can be defined at three different levels to specify the style of a document.

- CSS (Cascading Style Sheets) is a language designed to simplify the process of making web pages presentable.
- It allows you to apply styles to HTML documents by prescribing colors, fonts, spacing, and positioning.
- The purpose of CSS is to format web pages. Any formatting—color, organization, layout, etc.—of a web page is created by using CSS.
- The main advantages are separation of content (in HTML) and styling (in CSS) and the same CSS rules can be used across all pages and does not need to be rewritten.
- HTML uses tags where as CSS uses rule sets.
- CSS styles are applied to the HTML element using selectors.

CSS consists of style rules that are interpreted by the browser and applied to the corresponding elements. A style rule set includes a selector and a declaration block.

Selector: Targets specific HTML elements to apply styles.
Declaration: Combination of a property and its corresponding value.

Using CSS, you can specify a number of style properties for a given HTML element. Each property has a name and a value, separated by a colon (:). Each property declaration is separated by a semi-colon (;).

LEVELS OF STYLE SHEETS

The three levels of style sheets, in order from lowest level to highest level are: **inline styles**, **internal styles** and **external style sheets**.

- **Inline style sheet:** applies to the content of a single HTML element.
- **Internal style sheet:** applies to the whole body of a document. Internal style sheet is also known as **document level style sheet or embedded style sheet.**
- **External style sheet:** can be applied to multiple web pages/documents.

Inline Style Sheet

- Inline CSS involves applying styles directly to individual HTML elements using the **"style"** attribute.
- This should be done only when you are interested to make a particular change in any HTML element only.
- This method allows for specific styling of elements within the HTML document, overriding any external or internal styles.
- Inline style specifications appear within the opening tag and apply only to the content of that tag.

- Rules defined inline with the element overrides the rules defined in an external CSS file as well as the rules defined in <style> element.

Inline Style Specification:
style="property1**:**value1; property2**:**value2; property3**:**value3; ..... property**n:**value**n**;"

Example of Inline style sheet

```
<html>
  <head>
    <title>HTML Inline CSS</title>
  </head>
  <body>
    <p style = "color:red;">This is red</p>
    <p style = "font-size:20px;">This is thick</p>
    <p style = "color:green;">This is green</p>
    <p style = "color:green;font-size:20px;">This is thick and green</p>
  </body>
</html>
```

Internal Style Sheet (Document level or Embedded style sheet)

- If you want to apply style sheet rules to a single document/web page only, then you can include those rules in header section of the HTML document using **<style>** tag.
- Rules defined in internal style sheet overrides the rules defined in an external CSS file.

Document-level Style Specification
<**style** type = "text/css"> *Rule list* <**/style**>

Each style rule in a rule list has two parts:
A **selector**, which indicates the tag or tags affected by the rule, and a **list of property–value pairs**. The list has the same form as the quoted list for inline style sheets, except that it is delimited by braces rather than double quotes. So, the form of a style rule is as follows:

```
Selector {
            property1:value1;
            property2:value2;
            property3:value3;
            .....
            propertyn:valuen;
        }
```

Example of Document-level style sheet

```
<html>
  <head>
    <title>HTML Internal CSS</title>

    <style>
      .red { color: red;  }
      .thick{ font-size:20px; }
      .green { color:green;   }
    </style>
  </head>
  <body>
```

```
    <p class = "red">This is red</p>
    <p class = "thick">This is thick</p>
    <p class = "green">This is green</p>
    <p class = "thick green">This is thick and green</p>
  </body>
</html>
```

External Style Sheet

- External style sheets are stored separately and are referenced in all documents that use them.
- External style sheets are written as text files with the **.css** extension.
- They can be stored on any computer on the Web. The browser fetches external style sheets just as it fetches documents.
- The **<link>** tag is used to specify external style sheets. Within <link>, the "**rel**" attribute is used to specify the relationship between the current document and the linked document. The **href** attribute of <link> is used to specify the URL of the style sheet document.

Example:
"mystyles.css" file

```
.red {
  color: red;
}
.thick {
  font-size:20px;
}
.green {
  color:green;
}
```

page1.html

```
<html>
  <head>
    <title>HTML External CSS</title>
    <link rel = "stylesheet" type = "text/css" href = "mystyles.css">
  </head>
  <body>
    <p class = "red">This is red in first page</p>
    <p class = "thick">This is thick in first page</p>
    <p class = "green">This is green in first page</p>
    <p class = "thick green">This is thick and green in first page</p>
  </body>
</html>
```

page2.html

```
<html>
  <head>
    <title>HTML External CSS</title>
    <link rel = "stylesheet" type = "text/css" href = "mystyles.css">
```

```
  </head>
  <body>
    <p class = "red">This is red in second page</p>
    <p class = "thick">This is thick in second page</p>
    <p class = "green">This is green in second page</p>
    <p class = "thick green">This is thick and green in second page</p>
  </body>
</html>
```

SELECTOR FORMS

A CSS selector is the first part of a CSS Rule. It is a pattern of elements and other terms that tell the browser which HTML elements should be selected to have the CSS property values inside the rule applied to them.
The following are selector forms used in CSS:

- Simple/Element selector form
- Class selectors
- Id selectors
- Generic selectors
- Group selectors
- Universal selectors
- Pseudo classes

ELEMENT/SIMPLE SELECTOR FORMS

In case of simple selector, a tag is used. If the properties of the tag are changed, then it reflects at all the places when used in the program. The selector can be any tag. If the new properties for a tag are not mentioned within the rule list, then the browser uses default behavior of a tag.

```
<html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
   p { font-family: 'Lucida Handwriting'; font-size: 50pt; color: Red; }
 </style>
 </head>
 <body>
 <p>Sample paragraph 1</p>
<p>Sample paragraph 2</p>
<p>Sample paragraph 3</p>
</body>
</html>
```

CLASS SELECTORS

In class selector, it is possible to give different properties for different elements

```
<html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
  p.one { font-family: 'Lucida Handwriting'; font-size: 25pt; color: Red; }
  p.two { font-family: 'Monotype Corsiva'; font-size: 50pt; color: green; }
   </style>
```

```
 </head>
<body>
<p class = "one">This is paragraph 1 to test internal style sheet</p>
<p class = "two"> This is paragraph 2 to test internal style sheet </p>
 </body>
</html>
```

ID SELECTORS

An id selector allows the application of a style to one specific element.

```
<html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
#one { font-family: 'lucida calligraphy'; color: magenta; }
#two { font-family: 'comic sans ms'; color: red; }
</style>
</head>
 <body>
  <p id = "two">This is a paragraph</p>
  <h1 id = "one">This is heading 1</h1>
 </body>
 </html>
```

GENERIC SELECTORS

In case of generic selector, when the class is created, it would not be associated to any particular tag. In other words, it is generic in nature.

```
<html>
 <head>
 <title>Sample CSS</title>
<style type = "text/css">
.one { font-family: 'Monotype Corsiva'; color: blue; }
</style>
</head>
<body>
<p class = "one">This is paragraph 1 to test Generic selector</p>
<h1 class = "one">This is heading 1</h1>
<h6 class = "one">This is heading 6</h6>
</body>
 </html>
```

GROUP SELECTOR

The grouping selector is used to select all the elements with the same style definitions. Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

```
<html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
   h1,h2,p{ font-family: 'lucida calligraphy'; color: purple; }
</style>
```

```
</head>
 <body>
 <h1>This is heading 1</h1>
 <h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
 <p>This is a paragraph</p>
</body>
</html>
```

UNIVERSAL SELECTORS

The universal selector, denoted by an asterisk (*), applies its style to all elements in a document.

```
 <html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
* { font-family: 'lucida calligraphy'; color: purple; }
</style>
</head>
 <body>
 <h1>This is heading 1</h1>
 <h2>This is heading 2</h2>
<h3>This is heading 3</h3>
 <p>This is a paragraph</p>
</body>
</html>
```

PSEUDO CLASSES

Pseudo class selectors are used if the properties are to be changed dynamically. For example: when mouse movement happens, in other words, hover happens or focus happens.

```
<html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
 input:focus { font-family: 'lucida calligraphy'; color: purple; font-size:100; }
input:hover { font-family: 'lucida handwriting'; color: violet; font-size:40; }
</style>
 </head>
 <body>
<form action = " ">
 <p>
 <label> NAME: <input type = "text" /> </label>
 </p>
</form>
</body>
 </html>
```

FONT PROPERTIES

CSS font properties allow the developers to manipulate the font look and feel on the webpage. It is used to control the look of texts. You can use the CSS font property to change the text size, color, style, and more

.

FONT PROPERTIES

Property	Description	Values
font	Sets all the font properties in one declaration	*font-style, font-variant, font-weight, font-size/line-height, font-family,* caption, icon, menu, message-box, small-caption, status-bar, inherit
font-family	Specifies the font family for text	*family-name, generic-family,* inherit
font-size	Specifies the font size of text	xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger, *length, %,* inherit
font-style	Specifies the font style for text	normal, italic, oblique, inherit
font-variant	Specifies whether or not a text should be displayed in a small-caps font	normal, small-caps, inherit
font-weight	Specifies the weight of a font	normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900, inherit **Careful, many of these are not supported!**

Font Families

The font-family property is used to specify a list of font names. The browser uses the first font in the list that it supports. For example, the property:

font-family: Arial, Helvetica, Futura

tells the browser to use Arial if it supports that font. If not, it will use Helvetica if it supports it. If the browser supports neither Arial nor Helvetica, it will use Futura if it can. If the browser does not support any of the specified fonts, it will use an alternative of its choosing. If a font name has more than one word, the whole name should be delimited by single quotes, as in the following example:

font-family: 'Times New Roman'

Font Sizes

The font-size property does what its name implies. For example, the following property specification sets the font size for text to 10 points: **font-size: 10pt**

Many relative font-size values are defined, including xx-small, x-small, small, medium, large, x-large, and xx-large. In addition, smaller or larger can be specified. Furthermore, the value can be a percentage relative to the current font size.

Font Variants

The default value of the font-variant property is normal, which specifies the usual character font. This property can be set to small-caps to specify small capital characters. These characters are all uppercase, but the letters that are normally uppercase are somewhat larger than those that are normally lowercase.

Font Styles

The font-style property is most commonly used to specify italic, as in **font-style: italic**

Font Weights

The font-weight property is used to specify the degree of boldness, as in **font-weight: bold.** Besides bold, the values normal, bolder, and lighter can be specified. Specific numbers also can be given in multiples of 100 from 100 to 900, where 400 is the same as normal and 700 is the same as bold.

Font Shorthands:

If more than one font property must be specified, the values can be stated in a list as the value of the font property. The order in which the property values are given in a font value list is important. The order must be as follows: The font names must be last, the font size must be second to last, and the font style, font variant, and font weight, when they are included, can be in any order but must precede the font size and font names. **font: bold 14pt 'Times New Roman'**

```
<html>
 <head>
 <title>Font Properties</title>
 <style type = "text/css">
p.one { font-family: 'lucida calligraphy'; font-weight:bold; font-size:75pt; color: purple; }
h1.two { font-family: 'cambria'; color: violet; font-style:italics; }
p.three { font: small-caps italic bold 50pt 'times new roman' }
</style>
 </head>
 <body>
 <p class = "one">Paragraph  used to test style sheet</p>
<h1 class = "two">This is heading 1</h1>
<p class = "three">This is paragraph</p>
</body>
 </html>
```

TEXT PROPERTIES

Property	Description	Values
color	Sets the color of a text	RGB, hex, keyword
line-height	Sets the distance between lines	normal, *number*, *length*, *%*
letter-spacing	Increase or decrease the space between characters	normal, *length*
text-align	Aligns the text in an element	left, right, center, justify
text-decoration	Adds decoration to text	none, underline, overline, line-through
text-indent	Indents the first line of text in an element	*length*, *%*
text-transform	Controls the letters in an element	none, capitalize, uppercase, lowercase

CSS Text Formatting allows you to control the visual presentation of text on a webpage.
From changing fonts to adjusting spacing and alignment, CSS provides powerful properties to enhance the readability and aesthetic of text.
CSS lets you adjust font properties, text alignment, spacing, and decorations.

It helps in creating visually appealing text and improving the user experience.
Various text-related properties can be combined to achieve unique text styles and layouts.

Example:

font-size: 40px;
font-weight: bold;
color: #4CAF50;
text-transform: uppercase;
font-family: Arial, sans-serif;
Text-decoration: text-decoration: dashed | dotted | double | line-through | none | overline | solid | underline | wavy;
Text-decoration-line:Sets various text decorations like underline, overline, line-through.
Text-decoration-color: Sets color of text decorations like overlines, underlines, and line-throughs.
Text-justify: Justifies text by spreading words into complete lines.
Text-shadow: Adds shadow to text
Letter-spacing: Specifies space between characters of text.
Line-height: sets space between lines
Word-spacing: Specifies space between words of line.

Text Decoration
The text-decoration property is used to specify some special features of text. The available values are line-through, overline, underline, and none, which is the default.

```
<html>
 <head>
 <title>Text Decoration</title>
 <style type = "text/css">
h1.one {text-decoration: line-through;}
h1.two {text-decoration: overline;}
h1.three {text-decoration: underline;}
</style>
</head>
<body>
<h1 class = "one">This is heading 1</h1>
<p>[This is line-through]</p><br/>
<h1 class = "two">This is heading 1 with overline </h1>
<p>[This is overline]</p><br/>
 <h1 class = "three">This is heading 1 with underline</h1>
<p>[This is underline]</p><br/>
</body>
</html>
```

BACKGROUND IMAGES
The background-image property is used to place an image in the background of an element.

```
<html>
 <head>
 <title>Background Image</title>
 <style type = "text/css">
 body {background-image:url(computer.jpg);}
 p {text-align: justify; color:white;font-size:25pt;}
```

```
</style>
</head>
<body>
<p> Twinkle twinkle little star, how I wonder what you are? Up above the world so high, like a diamond in the sky. Twinkle twinkle little star, how I wonder what you are? Up above the world so high, like a diamond in the sky. </p>
</body>
</html>
```

In the example, notice that the background image is replicated as necessary to fill the area of the element. This replication is called *tiling*.
Tiling can be controlled with the background-repeat property, which can take the value repeat (the default), no-repeat, repeat-x, or repeat-y. The no-repeat value specifies that just one copy of the image is to be displayed. The repeat-x value means that the image is to be repeated horizontally; repeat-y means that the image is to be repeated vertically. In addition, the position of a non-repeated background image can be specified with the background-position property, which can take a large number of different values. The keyword values are top, center, bottom, left, and right, all of which can be used in many different combinations.

COLORS

Color Groups

Three levels of collections of colors might be used by an XHTML document. The smallest useful set of colors includes only those that have standard names and are guaranteed to be correctly displayable by all browsers on all color monitors. This collection of 17 colors is called the *named colors*.

Name	Hexadecimal Code	Name	Hexadecimal Code
aqua	00FFFF	olive	808000
black	000000	orange	FFA500
blue	0000FF	purple	800080
fuchsia	FF00FF	red	FF0000
gray	808080	silver	C0C0C0
green	008000	teal	008080
lime	00FF00	white	FFFFFF
maroon	800000	yellow	FFFF00
navy	000080		

Larger set of colors, called the Web palette, consists of 216 colors. The colors of the Web palette can be viewed at **http://www.web-source.net/216_color_chart.htm**

Color Properties

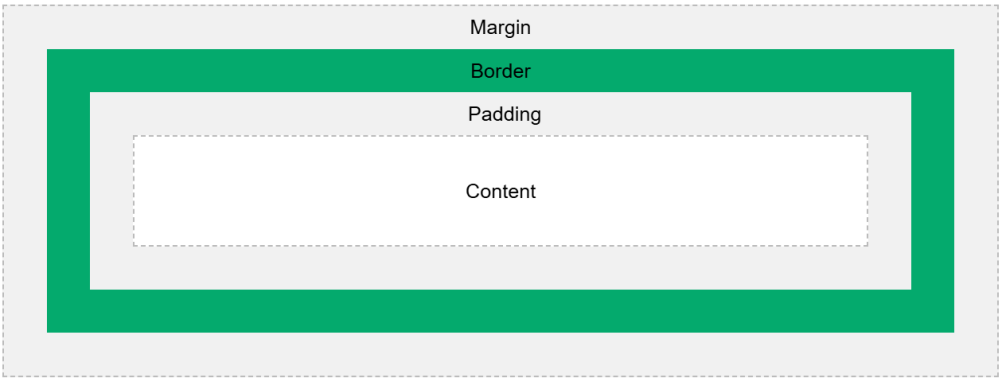
The color property is used to specify the foreground color of XHTML elements.

```
<html>
<head>
<title>Colours</title>
<style type = "text/css">
p.one {color: pink; }
p.two {color: # 9900FF;}
p.three {background-color:#99FF00;}
</style>
```

```
</head>
<body>
<p class = "one">This is paragraph 1</p>
<p class = "two">This is paragraph 2</p>
<p class = "three">This is paragraph 3</p>
</body>
</html>
```

THE BOX MODEL

In CSS, the term "box model" is used when we talk about design and layout.
The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins. The image below illustrates the box model:



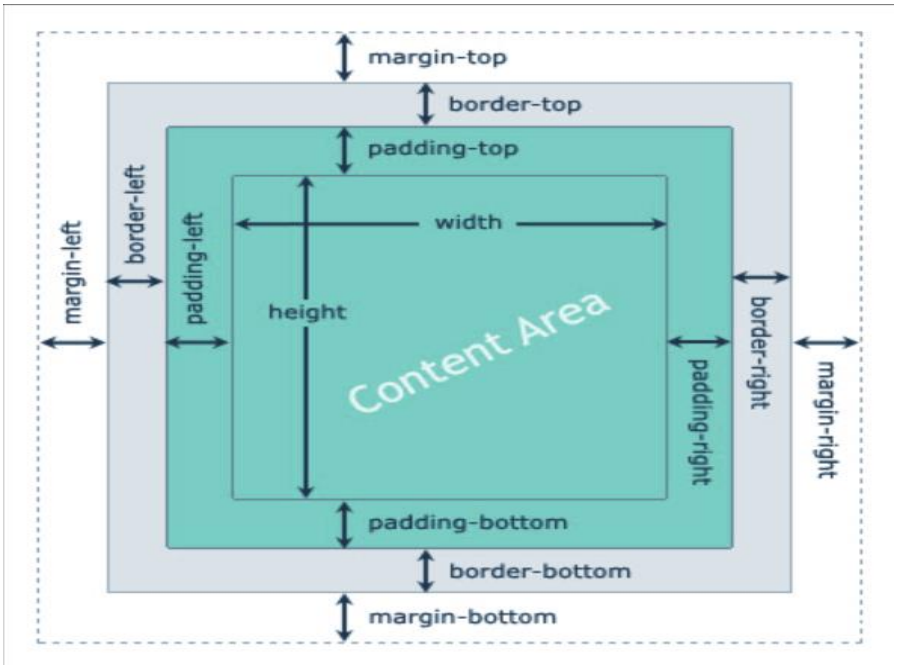
The CSS Box Model defines how elements are sized, positioned, and rendered on a webpage. When a browser loads an HTML document, it creates a DOM tree and assigns a box to each element. This box calculates the element's dimensions and position relative to its parent or the root <html> element, ensuring accurate layout and spacing.

Box Model Component Layout
Content: The area where text or other content is displayed.
Padding: Space between the content and the element's border.
Border: A frame that wraps around the padding and content.
Margin: Space between the element's border and neighboring elements.

Border-style
It can be dotted, dashed, double
Border-top-style, Border-bottom-style, Border-left-style,Border-right-style

Border-width
It can be thin, medium, thick or any length value
Border-top-width, Border-bottom-width, Border-left-width, Border-right-width

Border-color
Border-top-color, Border-bottom-color, Border-left-color,Border-right-color




```
<html>
<head>
 <title> Table with border effects </title>
 <style type = "text/css">
table { border-width:thick; border-top-color:red; border-left-color:orange; border-bottom-color:violet; border-right-color:green; border-top-style:dashed;border-bottom-style:double; border-right-style:dotted; }
</style>
</head>
 <body>
 <table border = "border">
 <caption>BORDERS DEMO </caption>
 <tr>
  <td> MCA @ BITM </td>
  <td> <img src = "BITM.PNG" width=200 height=200 alt = "cant display"/></td>
  </tr>
  </table>
  </body>
  </html>
```

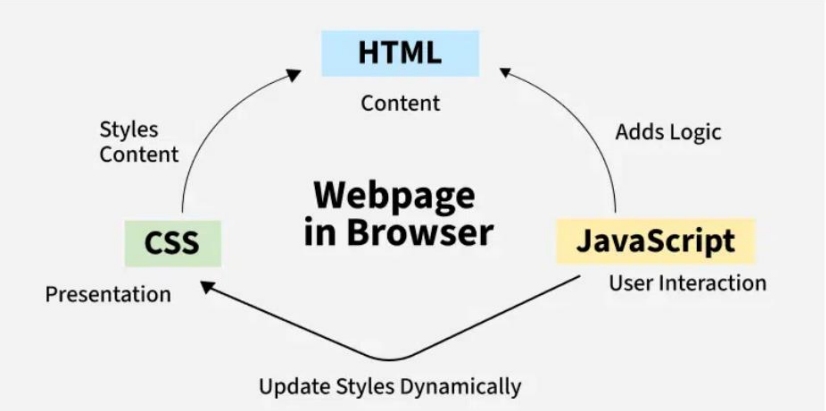
Margins and Padding: The margin properties are named margin, which applies to all four sides of an element: margin-left, margin-right, margin-top, and margin-bottom. The padding properties are named padding, which applies to all four sides: padding-left, padding-right, padding-top, and padding-bottom.

```
<html>
 <head>
 <title> Margins and Padding </title>
 <style type = "text/css">
 p.one { margin:0.1in; padding:0.5in; background-color:#FF33FF; border-style:dotted; }
 p.two { margin:0.5in; padding:0.1in; background-color:#00FF33; border-style:dashed; }
 p.three {margin:0.3in; background-color:#FFFF00; }
 p.four { padding:0.3in; background-color:#FF9900; }
</style>
```

```
</head>
<body>
 <p class = "one"> This line is used for testing styles for indentation <br/> [margin=0.1in, padding=0.5in]
</p>
<p class = "two"> This line is used to check styles for margins and padding<br/> [margin=0.5in, padding=0.1in]
</p>
 <p class = "three"> This line of text has no padding and no border<br/> [margin=0.3in, no padding, no border]
</p>
<p class = "four"> In this line we have padding but no margins and no border <br/> [no margin, padding=0.3in, no border]
</p>
 </body>
 </html>
```

INTRODUCTION TO JAVASCRIPT

- JavaScript is a light-weight scripting language which is used by several websites for scripting the webpages. JavaScript is used to create client-side dynamic pages.
- It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
- It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers.
- With JavaScript, users can build modern web applications to interact directly without reloading the page every time.
- It is *an object-based scripting language* which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.



Features of JavaScript

The following are some of the features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.

2. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
3. JavaScript is an object-oriented scripting language that uses prototypes rather than using classes for inheritance.
4. It is a light-weighted and interpreted language.
5. It is a case-sensitive language.
6. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
7. It provides good control to the users over the web browsers.

Applications of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

Uses of JavaScript

- Client-side validation.
- Displaying date and time.
- To validate the user input before submission of the form.
- Open and close new windows.
- To display dialog boxes and pop-up windows.
- To change the appearance of HTML documents.
- To create the forms that respond to user input without accessing the server.

Need of JavaScript (Additional uses of JS)

JavaScript can do the following things in addition to the above mentioned uses

- JavaScript Can Change HTML Content
- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles (CSS)
- JavaScript Can Hide HTML Elements
- JavaScript Can Show HTML Elements

Adding JavaScript to HTML programs

- JavaScript, also known as JS, is one of the scripting (client-side scripting) languages, that is usually used in web development to create modern and interactive web-pages.
- The term "script" is used to refer to the languages that are not standalone in nature and here it refers to JavaScript which run on the client machine.
- In other words, we can say that the term scripting is used for languages that require the support of another language to get executed.
- For example, JavaScript programs cannot get executed without the help of HTML or without integrated into HTML code.

JavaScript is used in several ways in web pages such as generate warning messages, build image galleries, DOM manipulation, form validation, and more.

These are following three ways in which users can add JavaScript to HTML pages.

1. Embedding code
2. Inline code
3. External file

1. Embedding code:-

- To add the JavaScript code into the HTML pages, we can use the <script>.....</script> tag of the HTML that wrap around JavaScript code inside the HTML program.
- Users can also define JavaScript code in the <body> tag (or we can say body section) or <head> tag because it completely depends on the structure of the web page that the users use.

We can understand this more clearly with the help of an example, how to add JavaScript to html.

Example

```
<html>
<head>
<title> page title</title>
<script>
        document.write("Welcome to BITM");
</script>
</head>
<body>
<p>This example shows how to add JavaScript code in the head section </p>
</body>
</html>
```

We can also define the JavaScript code within the body tag/body section

```
<html>
<head>
<title> page title</title>
</head>
<body>
<script>
        document.write("Welcome to BITM");
</script>
<p> This example shows how to add JavaScript to the body section </p>
</body>
</html>
```

2. Inline code

Generally, this method is used when we have to call a function in the HTML event attributes. There are many cases (or events) in which we have to add JavaScript code directly eg., OnMouseover event, OnClick, etc.

Let's see with the help of an example, how we can add JavaScript directly in the html without using the <script>.... </script> tag.

```
<html>
<head>
<title> page title</title>
</head>
<body>
<p>
<a href="#" onClick="alert('Welcome to BITM!');">Click Me</a>
</p>
<p> In this example we saw how to use inline JavaScript directly in an HTML tag. </p>
</body>
</html>
```

3. External file

We can also create a separate file to hold the code of JavaScript with the (.js) extension and later incorporate it into our HTML document using the **src** attribute of the <script> tag.

It becomes very helpful if we want to use the same code in multiple HTML documents.

It also saves us from the task of writing the same code over and over again and makes it easier to maintain web pages.
In this example, we will see how we can include an external JavaScript file in an HTML document.

```
<html>
<head>
        <title>Including a External JavaScript File</title>
</head>
<body>
        <form>
                <input type="button" value="Result" onclick="display()"/>
        </form>
<script src="myfile.js">
</script>
</body>
</html>
```

myfile.js

```
function display()
{
alert("Hello World!");
}
```

Both of the above programs are saved in the same folder, but you can also store JavaScript code in a separate folder, all just you need to provide the address/path of the (.js) file in the src attribute of <script> tag.

Note:

JavaScript files are common text files with (.js) extensions such as we created and used in the above program.
External JavaScript file only contains JavaScript code and nothing else, even the <script>.... </script>tag are also not used in it.

JAVASCRIPT VARIABLES

Declaring Variables: JavaScript allows three types of variable declarations:

- `var`: Older way to declare variables, function-scoped.
- `let`: Newer method, block-scoped, and can be reassigned.
- `const`: Block-scoped, used for constants (values that do not change).

Declaring variables using 'let'

The 'let' keyword was introduced in ES6 (2015)
Variables declared with 'let' have Block Scope
Variables declared with 'let' must be declared before use
Variables declared with 'let' cannot be redeclared in the same scope

Block Scope

Before ES6 (2015), JavaScript did not have Block Scope.
JavaScript had Global Scope and Function Scope.
ES6 introduced the two new JavaScript keywords: 'let' and 'const'.
These two keywords provided Block Scope in JavaScript:
Variables declared inside a { } block cannot be accessed from outside the block:
Ex:

```
  {
    let x = 2;
  }
 // x can NOT be used here
```

- Variables defined with 'let' can not be redeclared.

You can not accidentally redeclare a variable declared with let.
With let you can not do this:

```
let x = "Disha";
let x = 10;         //invalid redeclaration
```

- Variables defined with 'var' can be redeclared.

With var you can do this:

```
var x = "Disha";
var x = 10;       //valid redeclaration
```

Redeclaring Variables

- Redeclaring a variable using the 'var' keyword can impose problems.
- Redeclaring a variable inside a block will also redeclare the variable outside the block:
- Example

```
var x = 10;
// Here x is 10
{
var x = 2;
// Here x is 2
}
// Here x is 2
```

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.

2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Incorrect JavaScript variables

```
var  123=30;  //variable name cannot start with digit
var *aa=320;  //variable name cannot contain special character *
```

Ex: <script>

```
var x = 10;  var y = 20;
var z=x+y;
document.write(z);
```

</script>

DATATYPES IN JAVASCRIPT

There are altogether **8** basic data types in JavaScript.

Data Type	Description	Example
`String`	Textual data.	`'hello'`, `"hello world!"`, etc.
`Number`	An integer or a floating-point number.	`3`, `3.234`, `3e-2`, etc.
`BigInt`	An integer with arbitrary precision.	`900719925124740999n`, `1n`, etc.
`Boolean`	Any of two values: `true` or `false`.	`true` and `false`
`undefined`	A data type whose variable is not initialized.	`let a;`
`null`	Denotes a `null` value.	`let a = null;`
`Symbol`	A data type whose instances are unique and immutable.	`let value = Symbol('hello');`
`Object`	Key-value pairs of collection of data.	`let student = {name: "John"};`

Ex:

```
var x=10;  //number data type
var y=20.2343;   //number data type
var name=”hayath”   //string data type
var flag=true       //Boolean data type
var a,b=null;    //a - not defined and b is null
    let bi=1234567890123456790;   //bigint value
```

Non-Primitive data types

Object: Represents an instance through which we can access members
Array : Represents group of similar values

OPERATORS IN JAVASCRIPT

JavaScript supports the following operators

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- String Operators
- Miscellaneous Operators

ARITHMETIC OPERATORS

Operator	Name	Example
`+`	Addition	`3 + 4 // 7`
`-`	Subtraction	`5 - 3 // 2`
`*`	Multiplication	`2 * 3 // 6`
`/`	Division	`4 / 2 // 2`
`%`	Remainder	`5 % 2 // 1`
`++`	Increment (increments by **1**)	`++5` or `5++ // 6`
`--`	Decrement (decrements by **1**)	`--4` or `4-- // 3`
`**`	Exponentiation (Power)	`4 ** 2 // 16`

ASSIGNMENT OPERATORS

Operator	Name	Example
`=`	Assignment Operator	`a = 7;`
`+=`	Addition Assignment	`a += 5;  // a = a + 5`
`-=`	Subtraction Assignment	`a -= 2;  // a = a - 2`
`*=`	Multiplication Assignment	`a *= 3;  // a = a * 3`
`/=`	Division Assignment	`a /= 2;  // a = a / 2`
`%=`	Remainder Assignment	`a %= 2;  // a = a % 2`
`**=`	Exponentiation Assignment	`a **= 2;  // a = a**2`

COMPARISON OPERATORS

Operator	Meaning	Example
`==`	Equal to	`3 == 5` gives us `false`
`!=`	Not equal to	`3 != 4` gives us `true`
`>`	Greater than	`4 > 4` gives us `false`
`<`	Less than	`3 < 3` gives us `false`
`>=`	Greater than or equal to	`4 >= 4` gives us `true`
`<=`	Less than or equal to	`3 <= 3` gives us `true`
`===`	Strictly equal to	`3 === "3"` gives us `false`
`!==`	Strictly not equal to	`3 !== "3"` gives us `true`

LOGICAL OPERATORS

Operator	Syntax	Description
`&&` (Logical AND)	`expression1 && expression2`	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	`expression1 || expression2`	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	`!expression`	`false` if `expression` is `true` and vice versa

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

MISCELLANEOUS OPERATORS

Operator	Description	Example
`,`	**Comma**: Evaluates multiple operands and returns the value of the last operand.	`let a = (1, 3, 4); // 4`
`?:`	**Ternary**: Returns value based on the condition.	`(50 > 40) ? "pass" : "fail"; // "pass"`
`typeof`	Returns the data type of the variable.	`typeof 3; // "number"`
`instanceof`	Returns t `rue` if the specified object is a valid object of the specified class.	`objectX instanceof ClassX`
`void`	Discards any expression's return value.	`void(x) // undefined`

JAVASCRIPT CONDITIONAL STATEMENTS

Decision making statements: These allow you to execute specific blocks of code based on conditions. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.
There are several methods that can be used to perform Conditional Statements in JavaScript.

- **if statement**: Executes a block of code if a specified condition is true.
- **else statement**: Executes a block of code if the same condition of the preceding if statement is false.

- **else if statement**: Adds more conditions to the if statement, allowing for multiple alternative conditions to be tested.
- **switch statement**: Evaluates an expression, then executes the case statement that matches the expression's value.
- **ternary operator (conditional operator)**: Provides a concise way to write if-else statements in a single line.
- **Nested if else statement:** Allow for multiple conditions to be checked in a hierarchical manner.

Using if Statement

The if statement is used to evaluate a particular condition. If the condition is true, the associated code block is executed.

Syntax:

```
if ( condition ) {
   // If the condition is met,
   //code  will get executed.
}
```

```
Example :
let num = 20;

if (num % 2 === 0) {
   console.log("Given number is even number.");
}
```

Using if-else Statement

The if-else statement will perform some action for a specific condition. Here we are using the else statement in which the else statement is written after the if statement and it has no condition in their code block.

Syntax:

```
if (condition1) {
   // Executes when condition1 is true
   if (condition2) {
      // Executes when condition2 is true
   }
}
```

Example: In this example, we are using if-else conditional statement to check the whether he/she is eligible to vote or not

```
let age = 35;

if (age >= 18) {
   document.write("You are eligible to vote");
}
else {
   document.write("You are not eligible to vote");
};
```

If-else if Statement (if-else ladder)

The else if statement in JavaScript allows handling multiple possible conditions and outputs, evaluating more than two options based on whether the conditions are true or false. Out of

many alternatives any one block will be executed based on condition. If none of the condition is true then the else block will get executed.

Syntax:

```
if (1st condition) {
   // Code for 1st condition
} else if (2nd condition) {
   // ode for 2nd condition
} else if (3rd condition) {
   // Code for 3rd condition
} else {
   //  code that will execute if all above conditions are false
}
```

Example:

```
const num = 0;

if (num > 0) {
   document.write("Given number is positive.");
} else if (num < 0) {
   document.write ("Given number is negative.");
} else {
   document.write ("Given number is zero.");
};
```

Using Switch Statement (JavaScript Switch Case)

As the number of conditions increases, you can use multiple else-if statements in JavaScript. but when we dealing with many conditions, the switch statement may be a more preferred option.

Syntax:

```
switch (expression) {
   case value1:
      statement1;
      break;
   case value2:
      statement2;
      break;
   . . .
   case valueN:
      statementN;
      break;
   default:
      statementDefault;
};
```

Example:

```
<script>
var grade='B';
var result;
switch(grade){
```

```
case 'A':
      result="A Grade";
break;
case 'B':
      result="B Grade";
break;
case 'C':
      result="C Grade";
break;
default:
      result="No Grade";
}
document.write(result);
</script>
```

LOOPS IN JAVASCRIPT

Loops are powerful tools for performing repetitive tasks efficiently. They are used in JavaScript to execute a block of code again and again while the condition is true.
For example, suppose we want to print "Hello World" 20 times. This can be done using JS loop easily. In a loop, the statement needs to be written only once and the loop will be executed 20 times.

```
for (let i = 1; i < 20; i++)
{
   document.write("Hello World!");
}
```

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times
for/in - loops through the properties of an object
for/of - loops through the values of an iterable object
while - loops through a block of code while a specified condition is true
do/while - also loops through a block of code while a specified condition is true

The for Loop

The JS 'for' loop provides a concise way of writing the loop structure. The 'for loop' contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping.

Syntax:

```
for (initialization; testing condition; increment/decrement)
{
   statement(s)
}
```

- **Initialization condition:** It initializes the variable and mark the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.

- **Test Condition:** It is used for testing the exit condition of a for loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to be true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for the next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

```
// JavaScript program to illustrate for loop
let x;

// for loop begins when x = 2 and runs till x <= 4
for (x = 2; x <= 4; x++) {
    document.write("Value of x: " + x);
}
```

The while Loop

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax

```
while (boolean condition)
{
   loop statements...
}
```

- While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the **Entry control loop**
- Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
// JavaScript code to use while loop
let val = 1;
 while (val < 6) {
   document.write(val);
   val += 1;   }
```

The do-while Loop

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an **Exit Control Loop.** It executes loop content at least once even the condition is false.

Syntax

```
do
{
   Statements...
}
while (condition);
```

- The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked and therefore is an example of the exit control loop

```
let test = 1;
do {
   document.write(test);
   test++;
} while(test <= 5)
```

The for-of Loop

JS for-of loop is used to iterate the iterable objects for example – array, object, set and map. It directly iterate the value of the given iterable object and has more concise syntax than for loop.

Syntax:

```
for(let variable_name of  object_name) {
   // Statement
}
```

Example: This example shows the use of for-of loop.

- Javascript

```
let arr = [1, 2, 3, 4, 5];
for (let value of arr) {
    document.write(value);
}
```

For-in loop

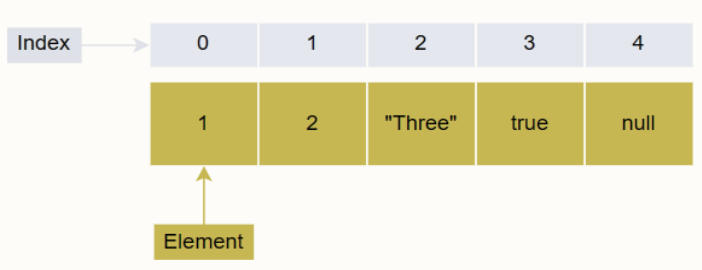
```
const obj = { name: “Vijay", age: 30 };
for (let x in obj) {
document.write(x, ":", obj[x]);
}
```

ARRAYS IN JAVASCRIPT

- Array is an ordered collection of values. Each value in an array is called an element, and each element has a numeric position in an array, known as its index.
- Arrays allow us to store more than one value or a group of values under a single variable name.
- They can store any valid value, including strings, numbers, objects, functions, and even other arrays, thus making it possible to create more complex data structures such as an array of objects or an array of arrays.

- A JavaScript array has the following characteristics:
    - First, an array can hold values of mixed types. For example, you can have an array that stores elements with the types number, string, boolean, and null.

- Second, the size of an array is dynamic and auto-growing. In other words, you don't need to specify the array size up front.



Creating an Array

There are 3 ways to construct array in JavaScript

- By array literal
- By creating instance of Array directly (using new keyword)
- By using an Array constructor (using new keyword)

1) JavaScript array literal

The simplest way to create an array in JavaScript is enclosing a comma-separated list of values in square brackets (`[]`), as shown in the following syntax:

var myArray = [*element0*, *element1*, ..., *elementN*];
Ex: var colors=[“yellow”,”green”,”red”,”blue”];

Ex:

```
<script>
var emp=[“abcd",“xyz",“mnop"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
```

2) JavaScript Array directly (new keyword)

- The syntax of creating array directly is given below:
- var arrayname=new Array();

```
<script>
var i;
var std = new Array();
std[0] = "Arun";
std[1] = "Varun";
std[2] = "John";
for (i=0;i<std.length;i++)
{
document.write(std[i] + "<br>");
}
</script>
```

3) JavaScript array constructor (new keyword)

- Here, we need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

```
<script>
var emp=new Array("Arun","Vijay","Prasad");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

JAVASCRIPT ARRAY METHODS

- concat() : It returns a new array object that contains two or more merged arrays.
- join() : It joins the elements of an array as a string.
- pop() : It removes and returns the last element of an array.
- push() : It adds one or more elements to the end of an array.
- reverse() : It reverses the elements of given array.
- sort() : It returns the element of the given array in a sorted order.
- slice() : It returns a new array containing the copy of the part of the given array.
- toString() : It converts the elements of a specified array into string form, without affecting the original array.

Length: "length property" is used to get the number of elements present in the array object

FUNCTIONS IN JAVASCRIPT

- A JavaScript function is a block of code that consists of a set of instructions to perform a specific task.
- A function can also be considered as a piece of code that can be used over again and again in the whole program, and it also avoids rewriting the same code.
- It also helps programmers/coders to divide a huge program into several small functions.

There are two types of functions in JavaScript like other programming languages.

- *Pre-defined Functions*
- *User-defined Functions*

To create a function in JavaScript we have to use the "function" keyword before writing the name of our function as you can see in given syntax:

```
function functionname( parameters list)
{
  Lines of code to be executed/set of instructions to be executed in order to perform a sp
ecific task.
}
```

Before using a function or we can say before calling a function in our program we have to define its definition in between the curly braces. As per your requirement, we can leave the parameter list blank as you can see in the syntax given above.

```
<html>
<head>
   <script>
function display()     //function defined
     {
   document.write("This is just a simple user-defined  function.<br />");
     }
 display();        //function called
 </script>
</head>
<body>
</body>
```

```
</html>
```

We can also call the display() function in the body section using the script tag as shown below:

```
<html>
<head>
   <title>Functions!!!</title>
   <script type="text/javascript">
function display()       {
   document.write("This is just a simple user-defined  function.<br />");
     }
  </script>
</head>
<body>
  <script>
       display();
  </script>
</body>
</html>
```

Function with Parameters

The function we have used in our program is without parameters (or we can say parameter less) because we have not given any parameter in the parameters list and left it empty. But we can also use parameters with function and we can use any number of parameters in our function but they must be separated by comma. These parameters are captured by the function and later any operation can be performed on these parameters inside the function.

Syntax of creating a function with parameters

```
function functionname( parameter1,parameter2,....parameterN)
{
 //Lines of code to be executed
//set of instructions to be executed in order to perform a specific task.
}
```

Example

```
<html>
<head>
<script>
function display(name,desig,dept)
  {
 document.write (name + " is " + desig + " in " + dept+" department");
   }
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type = "button" onclick = "display('Dr.SDN Hayath Ali', 'Assoc.Prof.','MCA')"
value = "Click Me!">
</form>
```

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

INPUT AND OUTPUT STATEMENTS IN JAVASCRIPT

- Using **prompt()** for User Input

The prompt() function is one of the simplest ways to get user input in JavaScript. This built-in function opens a dialog box where the user can type something, and the value is returned as a string.

Syntax

```
prompt("Message to the user", "Default Value")
```

Ex:

```
let x = prompt("Enter some text:");
```

- Example:

```
let userName = prompt("What is your name?");
if (userName) {
   alert("Hello, " + userName + "!");
} else {
   alert("No name entered.");
}
```

Alert() function in JavaScript

- The alert() method in JavaScript is used to display a virtual alert box. It is mostly used to give a warning message to the users.
- It displays an alert dialog box that consists of some specified message (which is optional) and an OK button. When the dialog box pops up, we have to click "OK" to proceed.
- The alert dialog box takes the focus and forces the user to read the specified message. So, we should avoid overusing this method because it stops the user from accessing the other parts of the webpage until the box is closed.
- Syntax:
  alert(message)
- Examples
  alert ("This is an alert dialog box");

```
alert (" Hello Guys!! \n Welcome to BITM \n This is an alert dialog box ");
```

Displaying output using JavaScript

- **JavaScript can "display" data in different ways**:
- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Document.write(): The `write()` method writes directly to an open (HTML) document stream. The write() method deletes all existing HTML when used on a loaded document.
Ex: document.write("Hello World!!!");

The **console.log()** function logs general information to the console. This is one of the most commonly used methods for debugging in JavaScript.

```
console.log("Hello, World!");
```

Using innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content.
- Example:

```
<html>
<body>
        <h1>My First Web Page</h1>
        <p>My First Paragraph</p>
        <p id="demo"></p>
        <script>
                document.getElementById("demo").innerHTML = 5 + 6;
        </script>
</body>
</html>
```

STRING HANDLING IN JAVASCRIPT

JavaScript strings are the sequence of characters. They are treated as Primitive data types. In JavaScript, strings are automatically converted to string objects when using string methods on them. This process is called auto-boxing. The following are methods that we can call on strings.

1. **slice():** The slice() method in JavaScript is used to extract a portion of a string and create a new string without modifying the original string.

Syntax:
string.slice(startingIndex, endingIndex);

startingIndex: It is the start position and it is required(The first character is 0).
endingIndex: (Optional)It is the end position (up to, but not including). The default is string length.
Ex:

```
let A = 'Hello Raj Kiran';
b = A.slice(0, 5);
c = A.slice(10);

console.log(b);
console.log(c);
```

Output:
Hello
Kiran

2. **substring():**The substring() method in JavaScript is used to extract characters between two indices from a string, without modifying the original string. This method returns a new string that is a part of the original string, starting from a specified position up to (but not including) another specified position.

   Syntax:

string.substring(startIndex, endIndex);
Parameters

startIndex: describe the part of the string to be taken as a substring
endIndex: describe the part of the string to be taken as a substring(optional**).**
Ex: let s = "Hello, World!";

// Extract substring from index 7 to index 12
let res = s.substring(7, 12);
console.log(res); // output: World

3. **replace():** The replace() method is used for manipulating strings. It allows you to search for a specific part of a string, called a substring, and then replace it with another substring.
   Syntax:

   str.replace(value1, value2);
   Parameters:
   value1: is the regular expression that is to be replaced
   value2: is a string that will replace the content of the given string.
   Ex:
   let string = 'Welcome';
   let newstring = string.replace('Welcome', 'Hello');
   console.log(newstring);

   Output:
   Hello

4. **toUpperCase():** converts all the characters present in the String to upper case and returns a new String with all characters in upper case. This method accepts single parameter **stringVariable** string that you want to convert in upper case.

Syntax:
str.toUpperCase()

Ex:
let str="Hello";

str.toUpperCase();

Output:
HELLO

5. **toLowerCase()**: converts all the characters present in the so lowercase and returns a new string with all the characters in lowercase.

Syntax:
str.toLowerCase()

Ex:
let str="Hello";

str.toLowerCase();

Output:
hello

6. **trim():** This function is used to remove either white spaces from the given string. This method returns a new string with removed white spaces. This method is called on a String object. This method doesn't accept any parameter.

Syntax
str.trim();
It returns a new string without any of the leading or trailing white spaces.

7. **charAt():**The charAt() method retrieves the character at a specified index in a string. The index is passed as an argument to the method, and it returns the character at that position.
JavaScript uses zero-based indexing, meaning the first character is at index 0, the second at index 1, and so on.
Syntax:
character = str.charAt(index);
Ex:
let str = 'JavaScript is object oriented language';

```
let value = str.charAt(0);
let value1 = str.charAt(9);
console.log(value);
console.log(value1);
```

Output:
J
t

8. **split():**The JavaScript split() method divides a string into an array of substrings based on a specified separator, such as a character, string, or regular expression. It returns the array of split substrings, allowing manipulation of the original string data in various ways.

Syntax
str.split(separator, limit);
Ex:

```
let str = 'Welcome to BITM';
let array = str.split(" ");
console.log(array);
```

Output
['Welcome','to','BITM']

EVENTS HANDLING IN JAVASCRIPT

- **Events** are actions that take place in the browser that can be initiated by either the user or the browser itself.
- We can also define an event as, "the change in the state of an object is known as event".

- In html, there are various events which represents that some activity is performed by the user or by the browser.
- When JavaScript code is included in HTML, JavaScript react over these events and allow the execution. This process of reacting over the events is called 'Event Handling'. Thus JS handles the HTML events via Event Handlers.

Following are some of the Events and their Event Handlers

Mouse events

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Form events

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Commonly used JavaScript Events

- **onclick**: Triggered when an element is clicked.
- **onmouseover**: Fired when the mouse pointer moves over an element.
- **onmouseout**: Occurs when the mouse pointer leaves an element.
- **onkeydown**: Fired when a key is pressed down.

- **onkeyup**: Fired when a key is released.
- **onchange**: Triggered when the value of an input element changes.
- **onload**: Occurs when a page has finished loading.
- **onsubmit**: Fired when a form is submitted.
- **onfocus**: Occurs when an element gets focus.
- **onblur**: Fired when an element loses focus.

Click Event Example

```
<html>
<head> Javascript Events </head>
<body>
<script>
   <!--
   function clickevent()
   {
      document.write("This is a demo of Button Click");
   }
   //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Click me"/>
</form>
</body>
</html>
```

Focus Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
   function focusevent()
   {
      document.getElementById("input1").style.background=" aqua";
   }
//-->
</script>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
```

```
<body>
<script>
   <!--
   function mouseoverevent()
   {
      alert("This is JavaTpoint");
   }
   //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

Keydown Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
   function keydownevent()
   {
      document.getElementById("input1");
      alert("Pressed a key");
   }
//-->
</script>
</body>
</html>
```

Load event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
document.write("The page is loaded successfully");
//-->
</script>
</body>
</html>
```

The addEventListener() method

- The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers.

- The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.
- Events are said to be an essential part of the JavaScript.
- A web page responds according to the event that occurred. Events can be user-generated or generated by API's.
- An event listener is a JavaScript's procedure that waits for the occurrence of an event.

VALIDATING FORM ELEMENTS

- It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.
- JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation.
- Most of the web developers prefer JavaScript form validation. Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

Types of Form Validation

- **Client-side validation:** JavaScript is typically used to validate forms on the user's browser before submission. This provides immediate feedback to users and prevents unnecessary server requests.
- **Server-side validation:** Even with client-side validation, it's crucial to re-validate form data on the server-side. This reduce any potential issues that might arise due to disabled JavaScript or browser manipulation.

Example program to validate username and password fields:

Validating a text box for entering only numeric data

```
<html>
<head>
<script>
function numberValidation(n) {

   if (isNaN(n)) {
      alert("Please enter Numeric value");
      return false;
   } else {
      alert("Numeric value is: " + n);
      return true;
   }
}
</script>
</head>
<body>
  <script>
    var x;
    x=prompt("enter any number ");
    numberValidation(x);
  </script>
 </body>
</html>
```

DOM2 Event model

The **DOM2 event model** is an improved way of handling events in JavaScript compared to the original **DOM0 event model**. In the DOM0 model, event handlers are directly assigned to HTML elements using attributes like 'onclick'. This approach can quickly become messy and hard to maintain, especially in larger applications.

The DOM2 event model introduced the concept of event listeners. Event listeners are functions that are registered to handle specific events on elements. Instead of directly assigning event handlers to elements, we can now add event listeners to elements and specify the event type and the function to be executed when the event occurs.

- It does not include DOM 0 features, but browsers still support them
- DOM2 is modularized – one module is Events, which has two submodules, HTMLEvents and MouseEvents, whose interfaces are Event (blur, change, etc.) and MouseEvent(click, mouseup etc.)
- Event Propagation
    - The node of the document tree where the event is created is called the ‘target node’
    - The ‘capturing phase (first phase)’
        - Events begin at the root and move toward the target node
        - Registered and enabled event handlers at nodes along the way are run
- The second phase is at the target node
    - If there are registered but not enabled handlers there for the event, they are run
- The third phase is the ‘bubbling phase’
    - Event goes back to the root; all encountered registered but not enabled handlers are run
- Not all events bubble (mouse events bubble but load and unload do not bubble)
- Any handler can stop further event propagation by calling the ‘stopPropagation’ method of the Event object
- DOM 2 model uses the Event object method, ‘preventDefault’, to stop default operations, such as form submission if an error has been detected
- Event handler registration is done with ‘addEventListener’ method