

MODULE-2

Introduction to CSS

CSS (Cascading Style Sheets) is a stylesheet language used to control the presentation and layout of HTML documents. It allows web developers to apply styles such as colors, fonts, spacing, and positioning to HTML elements, creating visually appealing and consistent web pages.

Each property in CSS has a name-value pair, and each property is separated by a semicolon (;).

Why Use CSS?

- **Separation of Content and Design:** CSS separates the structure (HTML) from the design, making it easier to manage and update styles.
- **Improved Accessibility:** Allows customization for different devices and screen sizes.
- **Efficiency:** Enables reusable styles across multiple pages, reducing code redundancy.
- **Enhanced User Experience:** Creates visually attractive and interactive web pages.

<pre> <!DOCTYPE html> <html lang="en"> <head> <style> h1{ color:white; background-color:red; } p{ color:blue; } </style> </head> <body> <h1>MULTIMEDIA</h1> <p>This is about to learn to create webpages. </p> </body> </html> </pre>	
---	--

Levels of style sheets

CSS can be integrated into an HTML document in three main ways:


1. **Inline CSS:** Directly within an HTML element using the style attribute.
2. **Internal CSS:** Inside a <style> tag within the <head> section of the HTML document.
3. **External CSS:** Through a separate .css file linked to the HTML document.

1. Inline CSS:

Inline CSS is applied directly within an HTML element using the style attribute. It is useful for quick styling of a single element but is not recommended for large projects due to poor maintainability.

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

Ex: `<p style="color: blue; font-size: 16px;">This is a blue paragraph.</p>`

<pre> <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Inline CSS Example</title> </head> <body> <h1 style="color: blue; text-align: center;">Web Technology</h1> </body> </html> </pre>	
---	---

2. Internal or Embedded CSS:

Internal CSS is defined within a `<style>` tag inside the `<head>` section of an HTML document. It is useful for styling a single web page without affecting other pages.

In the following example, we have used the inline CSS in `<p>` and `<h1>` tag.

Ex: `<head>`

```

<style>
  p {
    color: blue;
    font-size: 18px;
  }
</style>
</head>

```

PROGRAM

<pre> <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Internal CSS Example</title> </pre>	
---	--

```
<style>
  body {
    background-color: skyblue;
  }
  h1 {
    color: red;
    text-align: center;
  }
</style>
</head>
<body>
  <h1>Web Technology</h1>
</body>
</html>
```

3. External CSS:

External CSS is written in a separate .css file and linked to an HTML document using the <link> tag. It is the best practice for large projects as it keeps styles separate from content.

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

EX:

This file saves with .css extension.

For Ex: style.css

```
body {
  background-color: lightblue;
}
h1 {
  color: red;
  text-align: center;
}
```

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>External CSS Example</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Hello, World!</h1>
</body>
</html>

```

Web Technology

Creating an External Style Sheet

- Open a new blank document in Notepad
- Type style declarations
- h1 {color:red; font-family:sans-serif;}
- Do not include <style> tags
- Save the document as filename.css

Linking Style Sheets to HTML file

- Open an HTML file
- Between <head> and </head> add
 - <link href=URL rel="relation_type" type="link_type">
- Save this file and the .css file in the same web server directory

An example of an external style sheet with an original html file

html file	Text file named stylesheet.css
<pre> <!DOCTYPE html> <html lang="en"> <head> <title>Getting Started</title> <link href="scraps.css" rel="stylesheet" type="text/css" /> </head> <body> ... </body> </html> </pre>	<pre> h1 {font-family: sans-serif; color: orange} b {color: blue} </pre>

CSS Selectors

CSS selectors define the elements to which a set of CSS rules apply. They allow you to target HTML elements based on attributes, relationships, or positions within the document.

There are several different types of selectors in CSS.

1. CSS Element Selector
2. CSS Id Selector
3. CSS Class Selector
4. CSS Universal Selector
5. CSS Group Selector

1. The CSS element Selector

This selects all elements of a particular type. The element selector selects HTML elements based on the element name.

Syntax:

```
element {  
    property: value;  
}
```

For example, to select all paragraphs in the document, use the selector `p`.

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

Ex:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<style>  
p {  
    text-align: center;  
    color: red;  
}  
  
</style>  
</head>  
<body>  
<p>Every paragraph will be affected by the style. </p>  
<p id="para1">My name is Hareesha </p>
```

```
<p>Department of Computer Applications</p>
</body>
</html>
```

Every paragraph will be affected by the style.

My name is Hareesha

Department of Computer Applications

2. CSS id Selector

The **ID selector** in CSS is used to target a specific HTML element with a unique id attribute. The id of an element is unique within a page, so the id selector is used to select one unique element.

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Syntax:

```
#id-name {
  property: value;
}
```

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
<style>
#para1 {
  text-align: center;
  color: red;
}
</style>
</head>
<body>
<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>
</body>
</html>
```

Hello World!

This paragraph is not affected by the style.

3. CSS class Selector

The **class selector** in CSS is used to target **multiple elements** that share the same class name. Unlike the **ID selector (#)**, which applies to only one unique element, a **class (.)** can be reused across multiple elements.

To select elements with a specific class, write a period (.) character, followed by the class name.

Syntax:

```
.class-name {  
    property: value;  
}
```

Program:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<style>  
.center {  
    text-align: center;  
    color: red;  
}  
</style>  
</head>  
<body>  
<h1 class="center">Red and center-aligned heading</h1>  
<p class="center">Red and center-aligned paragraph. </p>  
</body>  
</html>
```

Red and center-aligned heading

Red and center-aligned paragraph.

4. CSS Universal Selector

The **universal selector (*)** in CSS selects **all** elements on a webpage. It is useful when you want to apply a style to every element or reset default styles.

Syntax:

```
* {  
    property: value;  
}
```

```

<!DOCTYPE html>
<html lang="en">
<head>
<style>
* {
  color: green;
  font-size: 20px;
}
</style>
</head>
<body>
<h2>This is heading</h2>
<p>This style will be applied on every paragraph.</p>
<p id="para1">Me too!</p>
<p>And me!</p>
</body>
</html>

```

This is heading

This style will be applied on every paragraph.

Me too!

And me!

5. CSS Group Selector

The **group selector** in CSS is used to apply the same style to multiple elements at once. It reduces redundancy and makes your CSS cleaner and more efficient.

Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

Syntax:

```

h1, h2, p {
  color: blue;
  font-family: Arial, sans-serif;
}

```

```

<!DOCTYPE html>
<html lang="en">
<head>
<style>
h1, h2, p {
  text-align: center;
  color: blue; }
</style>
</head>

```

Web Technology

Web Technology (In smaller font)

This is a paragraph.


```
<body>
<h1>Web Technology</h1>
<h2>Web Technology (In smaller font)</h2>
<p>This is a paragraph.</p>
</body>
</html>
```

Font

CSS provides various properties to control the appearance of text, including font family, size, weight, style, spacing, and more.

These are some important font attributes:

CSS Font Color

CSS font color is a standalone attribute in CSS although it seems that it is a part of CSS fonts. It is used to change the color of the text.

There are three different formats to define a color:

1. By a color name
Predefined color names (e.g., red, blue, green)
2. By hexadecimal value (#RRGGBB)
A six-digit hex code (values from 00 to FF for Red, Green, and Blue).
3. By RGB
Defines colors using Red, Green, and Blue values (0–255).

```
<!DOCTYPE html>
<html lang="en">
<head>
<style>
body {
  font-size: 100%;
}
h1 { color: red; }
h2 { color: #9000A1; }
p { color: rgb(0, 220, 98); }
</style>
</head>
<body>
```

This is heading 1

This is heading 2

This is a paragraph.

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<p>This is a paragraph.</p>
</body>
</html>
```

CSS Font Family

In CSS, the font-family property defines the typeface (font) of the text. It allows you to specify a list of fonts, ensuring a fallback if the preferred font is unavailable.

CSS font family can be divided in two types:

1. Generic family: It includes Serif, Sans-serif, and Monospace.

Serif: Serif fonts include small lines at the end of characters. Example of serif: Times new roman, Georgia etc.

Sans-serif: A sans-serif font doesn't include the small lines at the end of characters. Example of Sans-serif: Arial, Verdana etc.

2. Specific Font family: Specific fonts should be used in a **font stack** with generic families as a fallback. It specifies the font family name like Arial, New Times Roman etc.

```
<!DOCTYPE html>
<html lang="en">
<head>
<style>
body {
font-size: 100%;
}
h1 { font-family: sans-serif; }
h2 { font-family: serif; }
p { font-family: monospace; }
}
</style>
</head>
<body>
<h1>This heading is shown in sans-serif.</h1>
<h2>This heading is shown in serif.</h2>
```

This heading is shown in sans-serif.

This heading is shown in serif.

This paragraph is written in monospace.

<pre><p>This paragraph is written in monospace.</p> </body> </html></pre>	
---	--

CSS Font Size

The font-size property in CSS defines the size of text. It plays a key role in typography and affects readability, accessibility, and design consistency.

Syntax:

```
p {
  font-size: 16px;
}
```

For example: *font-size*: 10pt

Font Size Value	Description
xx-small	used to display the extremely small text size.
x-small	used to display the extra small text size.
small	used to display small text size.
medium	used to display medium text size.
large	used to display large text size.
x-large	used to display extra large text size.
xx-large	used to display extremely large text size.
smaller	used to display comparatively smaller text size
larger	used to display comparatively larger text size.
size in pixels or %	used to set value in percentage or in pixels.

<pre><!DOCTYPE html> <html lang="en"> <head> <title>Practice CSS font-size property</title> </head> <body></pre>	
--	--

<pre> <p style="font-size:xx-small;"> This font size is extremely small.</p> <p style="font-size:x-small;"> This font size is extra small</p> <p style="font-size:small;"> This font size is small</p> <p style="font-size:medium;"> This font size is medium. </p> <p style="font-size:large;"> This font size is large. </p> <p style="font-size:x-large;"> This font size is extra large. </p> <p style="font-size:xx-large;"> This font size is extremely large. </p> <p style="font-size:smaller;"> This font size is smaller. </p> <p style="font-size:larger;"> This font size is larger. </p> <p style="font-size:200%;"> This font size is set on 200%. </p> </body> </html> </pre>	<p>This font size is extremely small.</p> <p>This font size is extra small</p> <p>This font size is small</p> <p>This font size is medium.</p> <p>This font size is large.</p> <p>This font size is extra large.</p> <p>This font size is extremely large.</p> <p>This font size is smaller.</p> <p>This font size is larger.</p> <p>This font size is set on 200%.</p> <p>This font size is 20 pixels.</p>
--	---

CSS Font Style

Font styles

The font-style property in CSS defines the style of the text, such as normal, italic, or oblique.

Syntax:

```

p {
  font-style: italic;
}

```

For example: ***font-style***: italic

<pre> <html> <head> <style> body { font-size: 100%; } h2 { font-style: italic; } h3 { font-style: oblique; } h4 { font-style: normal; } } </style> </pre>	<p><i>This heading is shown in italic font.</i></p> <p><i>This heading is shown in oblique font.</i></p> <p>This heading is shown in normal font.</p>
---	---

```
</head>
<body>
<h2>This heading is shown in italic font.</h2>
<h3>This heading is shown in oblique font.</h3>
<h4>This heading is shown in normal font.</h4>
</body>
</html>
```

CSS Font Variant

CSS font variant property specifies how to set font variant of an element. It may be normal and small-caps.

```
<html>
<head>
<style>
p { font-variant: small-caps; }
h3 { font-variant: normal; }
</style>
</head>
<body>
<h3>This heading is shown in normal font.</h3>
<p>This paragraph is shown in small font.</p>
</body>
</html>
```

This heading is shown in normal font.

THIS PARAGRAPH IS SHOWN IN SMALL FONT.

CSS Font Weight

The font-weight property in CSS controls the **thickness or boldness** of text.

Syntax:

```
p {  
  font-weight: bold;  
}
```

Colors

The color property in CSS is used to set the color of HTML elements. Typically, this property is used to set the background color or the font color of an element.

In CSS, we use color values for specifying the color. We can also use this property for the border-color and other decorative effects.

We can define the color of an element by using the following ways:

1. RGB format.
2. RGBA format.
3. Hexadecimal notation.
4. HSL.
5. HSLA.
6. Built-in color

1. RGB Format

RGB format is the short form of 'RED GREEN and BLUE' that is used for defining the color of an HTML element simply by specifying the values of R, G, B that are in the range of 0 to 255.

Syntax

```
color: rgb(R, G, B);
```

2. RGBA Format

It is almost similar to RGB format except that RGBA contains A (Alpha) that specifies the element's transparency. The value of alpha is in the range 0.0 to 1.0, in which 0.0 is for fully transparent, and 1.0 is for not transparent.

Syntax

```
color: rgba(R, G, B, A);
```

3. Hexadecimal notation

Hexadecimal can be defined as a six-digit color representation. This notation starts with the # symbol followed by six characters ranges from 0 to F. In hexadecimal notation, the first two digits represent the

red (RR) color value, the next two digits represent the green (GG) color value, and the last two digits represent the blue (BB) color value.

The black color notation in hexadecimal is #000000, and the white color notation in hexadecimal is #FFFFFF. Some of the codes in hexadecimal notation are #FF0000, #00FF00, #0000FF, #FFFF00, and many more.

Syntax

```
color: #(0-F)(0-F)(0-F)(0-F)(0-F)(0-F);
```

3. Short Hex codes

It is a short form of hexadecimal notation in which every digit is recreated to arrive at an equivalent hexadecimal value.

For example, #7B6 becomes #77BB66 in hexadecimal.

The black color notation in short hex is #000, and the white color notation in short hex is #FFF. Some of the codes in short hex are #F00, #0F0, #0FF, #FF0, and many more.

4. HSL

It is a short form of Hue, Saturation, and Lightness.

Where,

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.

Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.

Lightness is also a percentage. 0% is black, 50% is neither light or dark, 100% is white

Syntax

```
color: hsl(H, S, L); // Ex: hsl(0, 100%, 50%)
```

5. HSLA

It is entirely similar to HSL property, except that it contains A (alpha) that specifies the element's transparency. The value of alpha is in the range 0.0 to 1.0, in which 0.0 indicates fully transparent, and 1.0 indicates not transparent.

Syntax

```
color: hsla(H, S, L, A);
```

6. Built-in Color

As its name implies, built-in color means the collection of previously defined colors that are used by using a name such as red, blue, green, etc.

Syntax: color: color-name;

Text Formatting

CSS Text Formatting refers to applying styles to text elements to control appearance and layout. This includes properties for color, alignment, decoration, indentation, justification, shadows, spacing, and direction. These properties enhance readability and aesthetics, improving the presentation of textual content on web pages.

Syntax:

property-name : /*value*/

Property	Description
text-color	Sets the color of the text using color name, hex value, or RGB value.
text-align	Specifies horizontal alignment of text in a block or table-cell element.
text-justify	Justifies text by spreading words into complete lines.
text-shadow	Adds shadow to text.
word-spacing	Specifies space between words of line.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Basic Text Formatting</title>
  <style>
    .text-color {
      color: blue;
    }
    .text-align-center {
      text-align: center;
    }
    .text-decoration {
      text-decoration: underline;
    }
    .text-indent {
      text-indent: 20px;
    }
    .letter-spacing {
      letter-spacing: 2px;
    }
  </style>
</head>
<body>
```



```

<p class="text-color">Changing Text Color</p>
<p class="text-align-center">Aligning Text</p>
<p class="text-decoration">Adding Text Decoration</p>
<p class="text-indent">Setting Text Indentation</p>
<p class="letter-spacing">Adjusting Letter Spacing</p>
</body>
</html>

```

Changing Text Color

Aligning Text

Adding Text Decoration

Setting Text Indentation

Adjusting Letter Spacing

Box model

Box Model is a Fundamental concept in CSS that defines how elements are structured and positioned on a webpage. It is used to develop the design and structure of a web page.

The box model in CSS is a container that contains various properties, including borders, margins, padding, and the content itself. These properties collectively determine the dimensions and spacing of an element.

The key components:

1. Content:

In the CSS Box Model, the content is the innermost part of an element where text, images, or other media are displayed.

It is affected by width and height but can be styled using CSS properties like color, font-size, background, etc.

2. Padding:

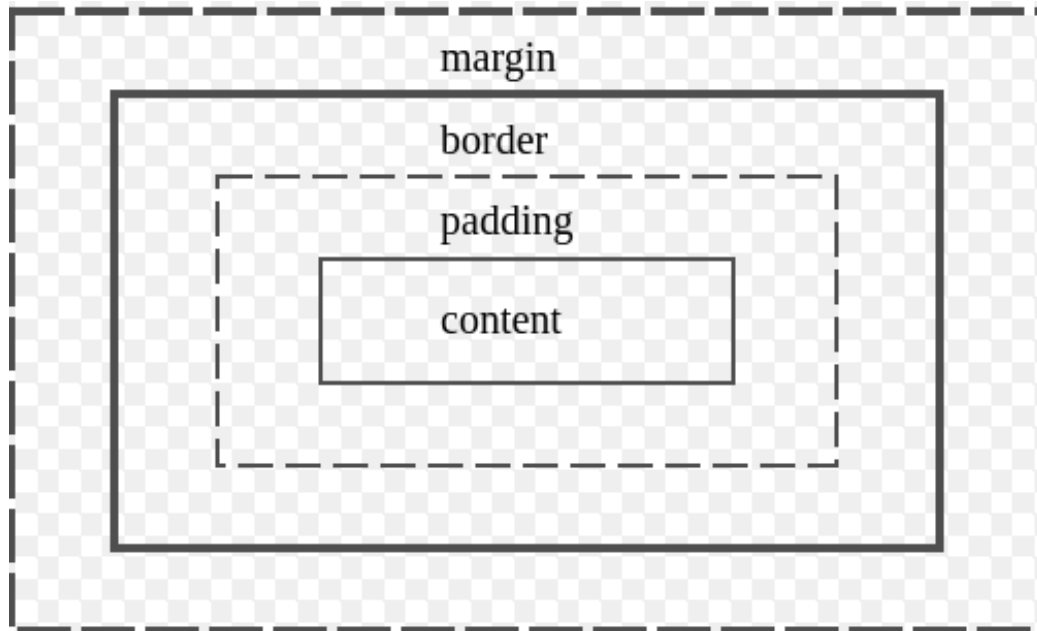
Padding is the space between the content and the border of an element. It creates inner spacing within the element, making content more readable.

3. Border:

A border is the outer edge of an element that surrounds the padding and content. It helps in defining boundaries and improving visual structure.

4. Margin:

The **margin** is the outermost space in the CSS **Box Model**. It creates space between an element and its neighbouring elements, preventing them from touching.



1. Content Area

- Contains the actual data, such as text, images, or other media.
- Sized using the width and height properties.
- Bounded by the content edge.

2. Padding Area

- Surrounds the content area.
- Space within the border box.
- Dimensions are determined by the width and height of the padding box.

3. Border Area

- Lies between the padding and margin.
- Width and height are defined by the border.

4. Margin Area

- Separates the element from adjacent elements.
- Dimensions specified by the margin-box width and height.

```
<head>
<title>CSS Box Model</title>
<style>
    .main
    {
        font-size:25px;
        font-weight:bold;
        Text-align:center;
    }
    .gfg
```

```
{
    margin-left:50px;
    border:50px solid skyblue;
    width:300px;
    height:200px;
    text-align:center;
    padding:50px;
}

.cbx
{
    font-size:40px;
    font-weight:bold;
    color:black;
    margin-top:60px;
    background-color:lightgreen;
}

.ctxt
{
    font-size:20px;
    font-weight:bold;
    background-color:white;
}

</style>
</head>
<body>
<div class = "main">CSS Box-Model Property</div>
    <div class = "gfg">
        <div class = "cbx">BOX Model</div>
        <div class = "ctx">It is used to develop the design and structure of a web page.</div>
    </div>
</body> </html>
```

CSS Box-Model Property

BOX Model

It is used to develop the design and structure of a web page.

Introduction to JavaScript

Introduction:

JavaScript (JS) is a high-level, interpreted programming language that is primarily used for making web pages interactive. It is one of the core technologies of web development, alongside HTML (structure) and CSS (styling).

Advantages of JavaScript

JavaScript is one of the most widely used programming languages due to its flexibility and efficiency.

1. Fast & Efficient

- JavaScript runs directly in the browser, reducing the need for server requests.
- No compilation is required, making it lightweight and quick.

2. Easy to Learn & Use

- The syntax is simple compared to other languages like Java or C++.
- Many online resources, tutorials, and documentation are available.

3. Runs in the Browser

- JavaScript does not need a separate runtime environment.
- Works on all modern web browsers (Chrome, Firefox, Edge, Safari).

4. Versatile & Cross-Platform

- Used for front-end (React, Vue, Angular) and back-end (Node.js) development.
- Can build mobile apps (React Native, Ionic) and games (Phaser.js, Three.js).
- Supports IoT, AI, and Machine Learning (TensorFlow.js).

5. Rich Libraries

- Thousands of libraries and frameworks (React, jQuery, Express, etc.).

6. Enhances Web Interactivity

- Adds dynamic content like animations, sliders, pop-ups, and forms.
- Can update content without reloading the page (AJAX, Fetch API).

8. Large Developer Community

- Huge community support with forums, Stack Overflow, and GitHub.
- Constantly evolving with new updates and features.

9. Compatible with Other Technologies

- Can work alongside HTML, CSS, Python, Java, and databases (MongoDB, MySQL).
- Easily integrates with web APIs and cloud services.

10. Open Source & Free

- No licensing fees or costs to use JavaScript.
- Open-source frameworks and tools are widely available.

Limitations of JavaScript

1. Security Risks

- Since JavaScript runs in the browser, it is **vulnerable to attacks** like Cross-Site Scripting (**XSS**) and Cross-Site Request Forgery (**CSRF**).
- Malicious scripts can be injected into web pages, stealing user data.

2. Client-Side Execution & Performance Issues

- Since JavaScript runs on the user's device, heavy scripts can slow down performance.
- Too many scripts running simultaneously can **freeze or crash** the page.

3. Lack of Multithreading

- JavaScript is **single-threaded**, meaning it can only execute one task at a time

The general Syntactic Characteristics

- JavaScript can be implemented using JavaScript statements that are placed within the `<script>... </script>` HTML tags in a web page.
- We can place the `<script>` tags, containing the JavaScript, anywhere within the web page, but it is normally recommended to keep it within the `<head>` tags.
- The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script.

A simple syntax of your JavaScript will appear as follows.

```
<script language="javascript" type="text/javascript">
```

```
JavaScript code
```

```
</script>
```

The script tag takes two important attributes:

- Language: This attribute specifies what scripting language you are using. Typically, its value will be `javascript`.
- Type: This attribute is what is now recommended to indicate the scripting language in use and its value should be set to `"text/javascript"`

General Syntactic Characteristics

1. Lightweight & Interpreted

- JavaScript does not require compilation; it is interpreted by the browser.
- This makes it fast and efficient for web applications.

2. Client-Side Execution

- Runs directly in the browser, reducing the load on servers.
- Enhances user experience by providing instant feedback and dynamic content.

3. Object-Oriented & Prototype-Based

- Uses objects and prototypes instead of traditional classes.

- Supports inheritance, allowing objects to share properties and methods.

4. Platform-Independent

- JavaScript can run on any device that supports a browser (Windows, Mac, Linux, Mobile).
- It is supported by all modern web browsers without additional installation.

5. High Interactivity

- Can manipulate the Document Object Model (DOM) to create dynamic web pages.
- Allows user interaction via events like clicks, scrolls, and key presses.

JavaScript variables

Variables are used to store data in JavaScript. JavaScript is a dynamically typed language so the type of variables is decided at runtime. Therefore, there is no need to explicitly define the type of a variable. We can declare variables in JavaScript in four ways:

1. Automatically
2. Using var
3. Using let
4. Using const

Rules of naming variables in JavaScript:

There are some rules while declaring a JavaScript variable

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.

Declaring variables

To declare text as a variable, you can use the "var" or "let" keyword. The Following syntax is used for declaring a variable in JavaScript:

Syntax:

```
var variable_name;
```

Example:

```
var a = "Hello Geeks";  
var b = 10;
```

A variable that has been declared but not assigned a value, has the value undefined.

Types of Variables:

There are two types of variables in JavaScript:

1. Local variable
2. Global variable

1. JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.

For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

2. JavaScript global variable

A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable.

For example:

```
<script>
var data=200;//global variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

Primitives Types:

- JavaScript has five primitive types: **Number, String, Boolean, Undefined, and Null.**
- Each primitive value has one of these types.
- JavaScript includes predefined objects that are closely related to the Number, String, and Boolean types, named **Number, String, and Boolean**, respectively.

Numeric and String Literals:

- All numeric literals are values of type Number. The Number type values are represented internally in double-precision floating-point form.
- Integer literals are strings of digits.
- Floating-point literals can have decimal points, exponents, or both.

Operators

JavaScript Operators are symbols used to perform specific mathematical, comparison, assignment, and logical computations on operands.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Conditional (ternary) operator

1. JavaScript Arithmetic Operators

JavaScript Arithmetic Operators perform arithmetic operations: addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**).

1. + operator:

The + operator in JavaScript is used for **both addition and string concatenation**, depending on the types of the operands.

Syntax

`x + y`

Ex:

```
let a = 5;
let b = 10;
console.log(a + b); // Output: 15
```

2. - operator:

The - (subtraction) operator in JavaScript is used to subtract one number from another. It returns the difference between the two values.

Syntax

`x - y`

Ex:

```
let num1 = 50;
let num2 = 20;
let result = num1 - num2; // 30
console.log(result);
```

3. * Multiplication:

The * operator in JavaScript is used for **multiplication**. It multiplies two numbers and returns the product.

Syntax

`x * y`

Ex:

```
let num1 = 8;
let num2 = 7;
let result = num1 * num2; // 56
console.log(result);
```

4. / (Division)

The / (division) operator in JavaScript is used to divide one number by another. It returns the quotient of the division.

Syntax

`x / y`

Ex:

```
let result = 10 / 2; // 5
console.log(result);
```

5. % (Modulus)

The % (modulus) operator in JavaScript returns the **remainder** when one number is divided by another.

Syntax

`x % y`

Ex:

```
console.log(10 % 0);
```

2. JavaScript Comparison Operators

Comparison operators are used to compare values and return a boolean (true or false).

1. == Equality Operators

The loose equality (==) operator compares two values for equality after type conversion.

Syntax

`x == y`

Ex:

```
console.log(5 == "5");
```

2. != (Not equal)

The loose inequality (!=) operator checks if two values are not equal, after type conversion.

Syntax

`x != y`

Ex:

```
console.log(5 != "5");
```

3. (>) Operator:

The greater than (>) operator is a comparison operator that checks if the value on the left is greater than the value on the right. It returns a boolean (true or false).

Syntax

`x > y`

Ex:

```
let age = 18;
if (age > 16) {
  console.log("You can drive!"); // Output: You can drive!
}
```

4. (>=) Operator

The greater than or equal to (>=) operator is a comparison operator that checks if the left operand is greater than or equal to the right operand. It returns a boolean (true or false).

Syntax

`x >= y`

Ex:

```
let age = 18;
if (age >= 18) {
  console.log("You are eligible to vote!"); // Output: You are eligible to vote!
}
```

4. (<=) Operator

The less than or equal to (<=) operator is a comparison operator that checks if the left operand is less than or equal to the right operand. It returns a boolean (true or false).

Syntax:

`x <= y`

Ex:

```
let temperature = 30;
if (temperature <= 32) {
```

```
        console.log("It's freezing outside!");  
    } else {  
        console.log("It's not too cold.");  
    }  
}
```

3. JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

1. Bitwise AND (&)

The **bitwise AND (&)** operator compares each bit of two numbers. It returns 1 if **both** corresponding bits are 1, otherwise, it returns 0.

Syntax

`a & b`

Ex:

```
let num = 5; // 0101 in binary  
let mask = 4; // 0100 (Third bit)  
console.log((num & mask) !== 0); // true (third bit is set)
```

2. Bitwise OR (|)

The bitwise OR (|) operator compares each bit of two numbers. It returns 1 if at least one of the corresponding bits is 1, otherwise, it returns 0.

Syntax

`a | b`

Ex:

```
let a = 0;  
let b = 42;  
console.log(a | b);
```

3. Bitwise XOR (^)

The bitwise XOR (^) operator compares each bit of two numbers. It returns 1 if the bits are different and 0 if they are the same.

Syntax

`a ^ b`

4. Bitwise NOT (~)

The bitwise NOT (~) operator inverts each bit of its operand. It flips 1 to 0 and 0 to 1, effectively computing the two's complement of a number.

Syntax

~a

Ex:

```
let num = 5;
console.log(~num);
```

5. Left Shift (<<)

The bitwise left shift (<<) operator shifts the bits of a number to the left by a specified number of positions.

Syntax

a << b

Ex:

```
console.log(5 << 1); // Output: 10
```

4. JavaScript Logical Operators

JavaScript provides several logical operators that allow you to perform logical operations on values.

These operators are commonly used in conditional statements, loops, and boolean expressions.

JavaScript Logical Operators perform logical operations: AND (&&), OR (||), and NOT (!), evaluating expressions and returning boolean values.

1. Logical AND (&&)

The logical AND (&&) operator evaluates two expressions and returns:

- true if both operands are true
- false if either operand is false

Syntax

condition1 && condition2

Ex:

```
console.log(true && true); // true
console.log(5 > 3 && 10 > 5); // true (both conditions are true)
```

2. Logical OR (||)

The Logical OR (||) operator in JavaScript evaluates two expressions and returns:

- The first truthy value it encounters
- If all values are falsy, it returns the last falsy value

Syntax

```
condition1 || condition2
```

Ex:

```
console.log(5 > 10 || 10 > 5); // true (because 10 > 5 is true)
console.log(false || true); // true
```

3. Logical NOT (!)

The Logical NOT (!) operator in JavaScript is used to invert the truthiness of a value.

- If the value is truthy, ! converts it to false.
- If the value is falsy, ! converts it to true

Syntax

```
!value
```

Ex:

```
console.log(!true); // false
```

5. JavaScript Assignment Operators

Assignment operators are used to assign values to variables. The basic assignment operator is = (equal sign), but JavaScript also provides several compound assignment operators that perform an operation and assign the result in one step.

1. Basic Assignment (=)

The basic assignment operator (=) is used to assign a value to a variable.

Syntax

```
variableName = value;
```

Ex:

```
let x = 10; // Assigns 10 to x
console.log(x); // Output: 10
```

2. (+=) Addition Assignment Operator

The += (Addition Assignment Operator) adds the value on the right to the existing value of the variable on the left and assigns the result back to the variable.

Syntax

```
variable += value;
```

Ex:

```
let x = 10;  
x += 5; // Equivalent to: x = x + 5  
console.log(x); // Output: 15
```

3. -= (Subtraction Assignment Operator)

The -= (Subtraction Assignment Operator) subtracts the value on the right from the variable on the left and assigns the result back to the variable.

Syntax

```
variable -= value;
```

Ex:

```
let x = 20;  
x -= 5; // Equivalent to: x = x - 5  
console.log(x); // Output: 15
```

4. *= (Multiplication Assignment Operator)

The *= (Multiplication Assignment Operator) multiplies the variable on the left by the value on the right and assigns the result back to the variable.

Syntax

```
variable *= value;
```

Ex:

```
let x = 10;  
x *= 5; // Equivalent to: x = x * 5  
console.log(x); // Output: 50
```

5. /= (Division Assignment Operator)

The /= (Division Assignment Operator) divides the variable on the left by the value on the right and assigns the result back to the variable.

Syntax

```
variable /= value;
```

Ex:

```
let x = 20;
x /= 5; // Equivalent to: x = x / 5
console.log(x); // Output: 4
```

6. Conditional (ternary) operator

The conditional (ternary) operator is the only JavaScript operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is truthy followed by a colon (:), and finally the expression to execute if the condition is false. This operator is frequently used as an alternative to an if...else statement.

Syntax

```
condition ? exprIfTrue : exprIfFalse
```

Control Statements

JavaScript **control statement** is used to control the execution of a program based on a specific condition. 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 control statements in JavaScript:

1. If Statement
2. If-Else Statement
3. Switch Statement
4. Ternary Operator (Conditional Operator)
5. For loop

1: If Statement

In this approach, we are using an if statement to check a specific condition, the code block gets executed when the given condition is satisfied.

Syntax:

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

Ex:

```
const num = 5;
if (num > 0) {
    console.log ("The number is positive.");
};
```


2: If-Else Statement

The if-else statement will perform some action for a specific condition. If the condition meets then a particular code of action will be executed otherwise it will execute another code of action that satisfies that particular condition.

Syntax:

```
if (condition1) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

Ex:

```
let num = -10;  
if (num > 0)  
    console.log ("The number is positive.");  
else  
    console.log ("The number is negative");
```

3: Using Switch Statement

The switch case statement in JavaScript is also used for decision-making purposes. In some cases, using the switch case statement is seen to be more convenient than if-else statements.

Syntax:

```
switch (expression) {  
    case value1:  
        statement1;  
        break;  
    case value2:  
        statement2;  
        break;  
    .  
    .  
    case value N:  
        statement N;  
        break;  
    default:  
        statementDefault;  
}
```

Ex:

```

let num = 5;
switch (num) {
  case 0:
    console.log ("Number is zero.");
    break;
  case 1:
    console.log ("Nuber is one.");
    break;
  case 2:
    console.log ("Number is two.");
    break;
  default:
    console.log ("Number is greater than 2.");
};

```

4: Using the Ternary Operator (Conditional Operator)

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

condition? value if true: value if false

Ex:

```

let num = 10;
let result = num >= 0 ? "Positive" : "Negative";
console.log(`The number is ${result}.`);

```

5: Using For loop

In this approach, we are using for loop in which the execution of a set of instructions repeatedly until some condition evaluates and becomes false

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
- **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:

```

for (statement 1; statement 2; statement 3) {

```

```
// code block to be executed
}
```

Ex:

```
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
<script>
var text = " ";
var i;
for (i = 0; i < 5; i++) {
text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

The While Loop:

The while loop loops through a block of code as long as a specified condition is true.

Syntax:

```
while (condition) {
// code block to be executed
}
```

Ex:

```
<html>
<body>
<h2>JavaScript While Loop</h2>
<p id="demo"></p>
<script>
var text = "";
var i = 0;
while (i < 10) {
text += "<br>The number is " + i;
i++;
}
document.getElementById("demo").innerHTML = text;
```

```

</script>
</body> </html>

```

The Do/While Loop:

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```

do {
  // code block to be executed
}
while (condition);

```

Ex:

```

<html>
<body>
<h2>JavaScript Do/While Loop</h2>
<p id="demo"></p>
<script>
var text = ""
var i = 0;
do {
  text += "<br>The number is " + i;
  i++;
}
while (i < 10);
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>

```

Functions

A function in JavaScript is a reusable block of code that performs a specific task and it defined with the function keyword, followed by a name, followed by parentheses ().

The rules for creating a function in JavaScript:

- Begin with the keyword **function** followed by,
- A user-defined function name
- A list of parameters enclosed within parentheses and separated by commas.
- A list of statements composing the body of the function enclosed within curly braces {} .

Syntax:

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Example:

```
function sum(x, y) {  
    return x + y;  
}  
console.log(sum(6, 9));
```

Return Statement

There are some situations when we want to return some values from a function after performing some operations. In such cases, we make use of the return. This is an optional statement.

.Function Parameters

Parameters are input passed to a function.

Calling Functions

After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters enclosed between the parenthesis.

Arrays in JavaScript

An array in JavaScript is a data structure used to store multiple values in a single variable. It can hold various data types and allows for dynamic resizing. Elements are accessed by their index, starting from 0.

1. Create using Literal

Creating an array using array literal involves using square brackets [] to define and initialize the array.

```
// Creating an Empty Array  
let a = [];  
console.log(a);  
// Creating an Array and Initializing with Values  
let b = [10, 20, 30];  
console.log(b);
```

2. Create using new Keyword (Constructor)

The “**Array Constructor**” refers to a method of creating arrays by invoking the Array constructor function.

```
// Creating and Initializing an array with values
let a = new Array(10, 20, 30);
console.log(a);
```

Arrays in JavaScript

JavaScript array is an object that represents a collection of similar type of elements. JavaScript array methods are built-in functions that allow efficient manipulation and traversal of arrays. They provide essential functionalities like adding, removing, and transforming elements, as well as searching, sorting, and iterating through array elements, enhancing code readability and productivity.

1. Array length

The length property returns the length of the given array.

Syntax:

```
Array.length
```

Ex:

```
let courses = ["HTML", "CSS", "JavaScript", "React"];
console.log(courses.length);
```

2. Array toString() Method

The toString() method converts the given value into the string.

Syntax:

```
arr.toString()
```

Ex:

```
let courses = ["HTML", "CSS", "JavaScript", "React"];
let str = courses.toString();
console.log(str);
```

3. Array join() Method

This join() method creates and returns a new string by concatenating all elements of an array. It uses a specified separator between each element in the resulting string.

Syntax

```
array.join(separator)
```

Ex:

```
let courses = ["HTML", "CSS", "JavaScript", "React"];
console.log(courses.join('|'));
```

4. Array.concat() Method

The concat() method is used to concatenate two or more arrays and it gives the merged array.

Syntax:

```
let newArray = arr.concat() // or
let newArray = arr1.concat(arr2) // or
let newArray = arr1.concat(arr2, arr3, ...) // or
let newArray = arr1.concat(value0, value1)
```

Ex:

```
let arr1 = [11, 12, 13];
let arr2 = [14, 15, 16];
let arr3 = [17, 18, 19];
let newArr = arr1.concat(arr2, arr3);
console.log(newArr);
```

5. Array.push() Method

The push () method is used to add an element at the end of an Array. As arrays in JavaScript are mutable objects, we can easily add or remove elements from the Array. And it dynamically changes as we modify the elements from the array.

Syntax:

```
Array.push(item1, item2 ...)
```

Ex:

```
let numArr = [10, 20, 30, 40, 50];
// Adding elements at the end of an array
numArr.push(60);
numArr.push(70, 80, 90);
console.log(numArr);
let strArr = ["piyush", "gourav", "smruti", "ritu"];
strArr.push("sumit", "amit");
console.log(strArr);
```

6. Array.pop() Method

The pop() method is used to remove elements from the end of an array.

Syntax:

```
Array.pop()
```

Ex:

```
let numArr = [20, 30, 40, 50];
```

```
// Removing elements from end of an array
numArr.pop();
console.log(numArr);
// Declare and initialize array
let strArr = ["amit", "sumit", "anil"];
// Removing elements from end of an array
strArr.pop();
console.log(strArr);
```

7. Array reverse() method

The reverse() method is used to reverse the order of elements in an array. It modifies the array in place and returns a reference to the same array with the reversed order.

Syntax:

```
array.reverse()
```

Ex:

```
let array = [1, 2, 3, 4, 5];
array.reverse();
console.log(array);
```