Html

<!DOCTYPE html>

<html>

<head>

<title> ---title of the document>

<meta> --- page description,character set

<link> used to link other files like CSS

<style> Internal Css

<script> Java script

<body>

<h1> headings

<p> paragraph

<img> embed the image

<a> Hyperlink

<div> division/section

<ul>,<ol><li> ordered or unordered lists

<table> can create table

<span> group inline elements

Semantic Elements in HTML

Many web sites contain HTML code like: <div id="nav"> <div class="header"> <div id="footer"> to indicate navigation, header, and footer.

In HTML there are some semantic elements that can be used to define different parts of a web page:

* <article> --- self-contained content.
* <aside> --- It is aside from the content
* <details>
* <figcaption>
* <figure>
* <footer> -- footer of the document
* <header>
* <main>
* <mark>
* <nav> -- nav bar contains navigation links
* <section> -- defines a section in a document
* <summary>
* <time>

<Table>

<form> used to take users information(textfield,submit,radio,checkboxes)

<frames> --- An HTML iframe is used to display a web page within a web page.

<div> - division/ block (used with id/class)

**Understanding CSS selectors, properties, and values**

CSS selectors are used to "find" (or select) the HTML elements you want to style.

---- Type Selector(Tag Selector): It targets the elements by their tag ex: <p>

---- Class Selector : It targets the elements by their class attribute ex: class = “min”

In css : .min{ }

Id selector : It targets the elements by their id ex: id=”header”

In css: #header { }

Attribute selectors : select elements based on an attribute or attribute value

Type=”text”

Pseudo-class selectors (select elements based on a certain state)

Ex: a:hover{

}

Pseudo-elements selectors (select and style a part of an element)

P::first-letter{

font-weight: bold;

}

**Implementing CSS box model**

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:

div {  
 width: 320px;  
 height: 50px;  
 padding: 10px;  
 border: 5px solid gray;  
 margin: 0;  
}

320px (width of content area)  
+ 20px (left padding + right padding)  
+ 10px (left border + right border)  
**= 350px (total width)** 50px (height of content area)  
+ 20px (top padding + bottom padding)  
+ 10px (top border + bottom border)

**= 80px (total height)**

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

.main

{

font-size:30px;

font-weight:bold;

Text-align:center;

}

.class

{

margin-left:100px;

border:50pxsolid Purple;

width:300px;

height:200px;

text-align:center;

padding:50px;

}

.class1

{

font-size:40px;

font-weight:bold;

color:black;

margin-top:80px;

margin-left: 10px;

background-color:purple;

}

.class2

{

font-size:20px;

font-weight:bold;

background-color:white;

margin-top: 50px;

}

</style>

</head>

<body>

<divclass = "main">CSS Box-Model Property</div>

<divclass = "class">

<divclass = "class1"> We are learning</div>

<divclass = "class2">CSS BoX Model</div>

</div>

</body>

</html>

**Working with CSS layouts: flexbox and grid**

**Flexbox:**

The flexbox or flexible box layout module (usually referred to as flexbox) is a one-dimensional layout model for distributing space between items and includes numerous alignment capabilities.

FlexBox:

----------- Aligning items either horizontally or vertically with in a container.

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

.flex-container{

display: flex;

padding: 30px;

margin: 30px;

height: 300px;

width: 300px;

border: 2pxsolid black;

justify-content: space-evenly;

align-items: center;

}

.flex1-container{

display: inline-flex;

height: 200px;

width: 400px;

padding: 30px;

margin: 30px;

background-color: brown;

justify-content: space-evenly;

}

</style>

</head>

<body>

<divclass="flex-container">

<div>one</div>

<div>two</div>

<div>three</div>

</div>

<divclass="flex1-container">

<div>1</div>

<div>2</div>

<div>3</div>

</div>

</body>

</html>

**Grid Layout**

Grid Layout is two dimensional , we can work in both row and coloum.

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

.wrapper{

display: grid;

grid-template-columns: repeat(5, 1fr);

grid-gap: 30px;

}

</style>

</head>

<body>

<divclass="wrapper">

<div>One</div>

<div>Two</div>

<div>Three</div>

<div>Four</div>

<div>Five</div>

</div>

</body>

</html>

**Working with CSS3 transitions and animations**

CSS Transitions

CSS transitions allows you to change property values smoothly, over a given duration.

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

div{

transition: background-color 0.5sease; }

div:hover{

background-color: red;

}

</style>

</head>

<body>

<div> jsvhdskjhsdkhfds</div>

</body>

</html>

## **CSS Animations**

An animation lets an element gradually change from one style to another.

You can change as many CSS properties you want, as many times as you want.

To use CSS animation, you must first specify some keyframes for the animation.

Keyframes hold what styles the element will have at certain times.

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

@keyframesexample {

from {background-color: red;}

to {background-color: yellow;}

}

/\* The element to apply the animation to \*/

div {

width: 100px;

height: 100px;

background-color: red;

animation-name: example;

animation-duration: 4s;

}

</style>

</head>

<body>

<divexample="animation"> hjdsgcdsjhcgdjshgcs</div>

</body>

</html>

**Understanding responsive design with CSS3 media queries**

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

* width and height of the viewport
* orientation of the viewport (landscape or portrait)
* resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<style>

@mediascreen and (max-width: 900px) {

body{

background-color: blue;

}

}

@mediascreen and (max-width: 600px) {

body{

background-color: red;

}

}

</style>

<body>

</body>

</html>

**Working with Bootstrap for responsive design**

Bootstrap is a free, open source front-end development framework for the creation of websites and web apps. Designed to enable responsive development of mobile-first websites, Bootstrap provides a collection of syntax for template designs.

As a framework, Bootstrap includes the basics for responsive web development, so developers only need to insert the code into a pre-defined grid system. The Bootstrap framework is built on Hypertext Markup Language (HTML), cascading style sheets (CSS) and JavaScript. Web developers using Bootstrap can build websites much faster without spending time worrying about basic commands and functions.

Where to Get Bootstrap?

There are two ways to start using Bootstrap on your own web site.

You can:

* Download Bootstrap from getbootstrap.com
* Include Bootstrap from a CDN

**Containers**

Bootstrap also requires a containing element to wrap site contents.

There are two container classes to choose from:

1. The .container class provides a responsive **fixed width container**
2. The .container-fluid class provides a **full width container**, spanning the entire width of the viewport

Assignment 1: Implement a navigation bar using an unordered list with links to different sections of your HTML page. Use CSS to style the list as a horizontal menu and highlight the current page or section.

Assignment 2: Create a simple HTML page that includes the use of headings, paragraphs, and at least two semantic tags like <article> or <section>. Add a table with data of your choice and a form with fields for a user's name, email, and a submit button.

Assignment 3: Apply the CSS box model to ensure that your page content has appropriate margins and padding. Create a layout using div tags and style them to arrange content in a multi-column format using floats or flexbox.

Assignment 4: Enhance the page by adding CSS3 animations to the menu and form elements. Also, use media queries to make the page responsive, ensuring it looks good on both desktop and mobile screen sizes.

Assignment 5: Take the static HTML page from Day 1 and integrate Bootstrap. Refactor the navigation bar, table, and form to use Bootstrap components. Ensure the page is responsive using Bootstrap's grid system."

Java Script

It is one of the core technologies of web page creation along with Html and CSS.

It is used to create interactive( give response to user) web page.

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

**Variables are Containers for Storing Data**

JavaScript Variables can be declared in 4 ways:

* Automatically
* Using var
* Using let
* Using const

In this first example, x, y, and z are undeclared variables.

They are automatically declared when first used:

Example

x = 5;  
 y = 6;  
 z = x + y;

const price1 = 5;  
 const price2 = 6;  
 let total = price1 + price2;

The two variables price1 and price2 are declared with the const keyword.

These are constant values and cannot be changed.

The variable total is declared with the let keyword.

The value total can be changed.

Variables declared inside a { } block cannot be accessed from outside the block:

{  
 let x = 2;  
 }  
 //

JavaScript Display Possibilities

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().

**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

<!DOCTYPE html>  
<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>

**Using Console.log and document.write()**

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p>My first paragraph.</p>

<script>

let i =0;

for(i=0;i<5;i++)

{

console.log(i);

document.write(i+"<br>");

}

</script>

</body>

</html>

**Examples :**

**Ex1.js**

let i =0;

for(i=0;i<5;i++)

{

console.log(i);

document.write(i+"<br>");

}

const day = prompt("Enter the day");

if(day === "Rainy")

document.write("Use umbrella");

else if (day === "Sunny")

document.write("Use sunscream");

else

{

document.write("Enjoy the weather");

alert("You have not enerred the correct day");

}

**Run.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src = Ex1.js></script>

</head>

<body>

</body>

</html>

**Object in JS**

**Create an Object**

const person = {  
 firstName: "John",  
 lastName: "Doe",  
 age: 50,  
 eyeColor: "blue"  
 };

**creates an empty JavaScript object, and then adds 4 properties**:

// Create an Object  
 const person = {};  
  
 // Add Properties  
 person.firstName = "John";  
 person.lastName = "Doe";  
 person.age = 50;  
 person.eyeColor = "blue";

**JavaScript object using new Object(), and then adds 4 properties:**

Example

// Create an Object  
 const person = new Object();  
  
 // Add Properties  
 person.firstName = "John";  
 person.lastName = "Doe";  
 person.age = 50;  
 person.eyeColor = "blue";

**JavaScript Object Methods**

Ex1.js

const person = {

firstName: "John",

lastName: "Doe",

id: 5566,

fullName: function() {

return this.firstName + " " + this.lastName;

}

};

document.write(person.fullName());

**Object Constructor Functions**

Sometimes we need to create many objects of the same **type**.

To create an **object type** we use an **object constructor function**.

function Person(first, last, age, eye) {  
 this.firstName = first;  
 this.lastName = last;  
 this.age = age;  
 this.eyeColor = eye;  
 }

const myFather = new Person("John", "Doe", 50, "blue");  
 const myMother = new Person("Sally", "Rally", 48, "green");  
 const mySister = new Person("Anna", "Rally", 18, "green");  
  
 const mySelf = new Person("Johnny", "Rally", 22, "green");

JavaScript Arrays

An array is a special variable, which can hold more than one value:

const cars = ["Saab", "Volvo", "BMW"];

You can also create an array, and then provide the elements:

Example

const cars = [];  
 cars[0]= "Saab";  
 cars[1]= "Volvo";  
 cars[2]= "BMW";

**Understanding arrow functions**

const x = function(x,y)

{

return x+y;

};

// arrow function

const m = (x,y)=>{ return x+y};

document .write(m(2,3));

**Implementing classes in JavaScript**

**Class in Java script is template for javascript object.**

classStudent{

constructor(name,age){

this.name=name;

this.age=age;

}

display(){

console.log(this.name);

console.log(this.age);

document.write(this.name+" "+this.age+"<br>");

}

}

consts1 =new Student(`Madhura`, 22);

consts2 =new Student(`Sai`, 23);

s1.display();

s2.display();

**For Of**

const numbers = [22,45,78,90];

let sum =0;

for(let x of numbers)

{

Sum = sum+x;

}

JavaScript Maps

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript Maps</h1>

<h2>The new Map Method()</h2>

<p>Creating a map from an array:</p>

<p id="demo"></p>

<script>

// Create a Map

const fruits = new Map([

["apples", 500],

["bananas", 300],

["oranges", 200]

]);

let numb = fruits.get("apples");

document.getElementById("demo").innerHTML = "There are " + numb + " apples.";

</script>

</body>

</html>

**Understanding Asynchronous JavaScript**

Functions running in parallel with other functions are called asynchronous

A good example is JavaScript setTimeout().

// sync example

functionsync()

{

document.write("One "+"<br>"+" ");

document.write("Two "+"<br>"+" ");

document.write("Three ");

}

//sync();

// async example

functionasync()

{

setTimeout(()=>

{

document.write("One "+"<br>"+" ");},3000);

document.write("Two "+"<br>"+" ");

document.write("Three "+"<br>"+" ");

}

async();

**JavaScript code Asynchronous**

**We can male javascript code asynchronous by two ways:**

**-----------callback**

**-----------Promises( Alternate to call back)**

**Callback**

A callback is a function passed as an argument to another function.

functioncallbackfunction(){

document.write("callback function called");

}

functioncallbackfunction1(){

setTimeout(()=>{

callbackfunction();

},3000)

}

callbackfunction1();

**callback1.js**

functiondisplay(msg)

{

document.write(JSON.stringify(msg));

}

functiongetIp(callback)

{

fetch("https://api.ipify.org?format=json")

.then(response=>response.json()).then(msg=>callback(msg));

}

getIp(display);

**Callback Hell:**

Callback Hell is essentially nested callbacks stacked below one another forming a pyramid structure. Every callback depends/waits for the previous callback, thereby making a pyramid structure that affects the readability and maintainability of the code.

The callback hell in JavaScript is referred to as a situation where an excessive amount of nested callback functions are being executed. It reduces code readability and maintenance. The callback hell situation typically occurs when dealing with asynchronous request operations, such as making multiple API requests or handling events with complex dependencies.

**Promises**

A Promise contains both the producing code and calls to the consuming code:

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is an Object that links Producing code and Consuming code

A JavaScript Promise object can be:

* Pending
* Fulfilled
* Rejected

The Promise object supports two properties: state and result.

While a Promise object is "pending" (working), the result is undefined.

When a Promise object is "fulfilled", the result is a value.

When a Promise object is "rejected", the result is an error object.

letpromise =new Promise((resolve, reject) => {

letday =new Date().getDay();

if (day ==5) {

resolve("Today is Friday");

} else {

reject("Today is not Friday");

}

});

promise.then((msg) =>{

document.write(msg);

}).catch((error) =>{document.write(error);

})

**Promise2 Example**

functiongetIp()

{

returnfetch("https://api.ipify.org?format=json");

}

letpromise =getIp();

promise.then(response=> response.json()).then(msg=>document.write(JSON.stringify(msg))).catch(error=>document.write(error));

**Async/Await function**

*async and await make promises easier to write"*

async makes a function return a Promise

await makes a function wait for a Promise

The keyword async before a function makes the function return a promise:

Example

async function myFunction() {  
 return "Hello";  
}

Is the same as:

function myFunction() {  
 return Promise.resolve("Hello");  
}

**Await Syntax**

The await keyword can only be used inside an async function.

The await keyword makes the function pause the execution and wait for a resolved promise before it continues:

async function myDisplay() {  
 let myPromise = new Promise(function(resolve) {  
 resolve("I love You !!");  
 });  
 document.getElementById("demo").innerHTML = await myPromise;  
}  
  
myDisplay();

**Asyncawait Example:**

asyncfunction getIp()

{

letresponse = await fetch("https://api.ipify.org?format=json");

letmsg =await response.json();

document.write(JSON.stringify(msg));

}

getIp();

**Understanding the Document Object Model (DOM)**

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The W3C DOM standard is separated into 3 different parts:

* Core DOM - standard model for all document types
* XML DOM - standard model for XML documents
* HTML DOM - standard model for HTML documents

**What is the HTML DOM?**

The HTML DOM is a standard object model and programming interface for HTML. It defines:

The HTML elements as objects

The properties of all HTML elements

The methods to access all HTML elements

The events for all HTML elements

In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

Example

The following example changes the content (the innerHTML) of the <p> element with id="demo":

<html>

<body>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = "Hello World!";

</script>

</body>

</html>

In the example above, getElementById is a method, while innerHTML is a property.

**Assignment 1:** Create a JavaScript object representing a user with properties for name, email, and age. Write functions that manipulate this object, such as changing the name, updating the email, and calculating the user's birth year.

**Assignment 2:** Use regular expressions in JavaScript to validate the email address entered in the form. It should check for the correct format of the email and display a message to the user if the format is incorrect."