Web-Programming-Module-2-Important-Topics-PYQs

For more notes visit

https://rtpnotes.vercel.app

        - Relative Positioning
- 2. What is a call back function in JavaScript? State how it is different from normal functions
    - Example of a Callback Function:
    - How is a Callback Different from Normal Functions?
- 3. Name components of box model?
    - The 4 Components of the CSS Box Model
- 4. List any 3 methods associated with javascript String object with their input parameter and return type ?
- 5. Explain id selector and class selector in CSS with suitable examples
    - 1. ID Selector
        - Example of ID Selector:
    - 2. Class Selector
        - Example of Class Selector:
- 6. Explain array creation in JavaScript with examples.
    - 1. Using Array Literal Syntax
        - Example:
    - 2. Using the Array Constructor
        - Example 1 (Empty Array with a Defined Length):
        - Example 2 (Array with Initial Elements):
    - 3. Using Array.of() Method
        - Example:
    - 4. Using Array.from() Method
        - Example 1 (From a String):
        - Example 2 (From a Set):
    - Summary of Array Creation Methods:
- 7. Write CSS style rules to implement the following in a web page to display the content of hyperlinks with red background colour and in italics, to display the contents of unordered lists in Times New Roman font, to display "Ocean.gif" as the background image of the page.
    - CSS Style Rules
    - Explanation:
    - Example HTML

- 8. Write CSS code for the following:
- 9. Discuss various types of control statements in JavaScript
    - 1. Conditional Statements
    - 2. Looping Statements
    - 3. Jump Statements
    - 4. Exception Handling Statements
- 10. How are event handlers registered in JavaScript? Write the code for an HTML document with embedded JavaScript scripts, which initially displays a paragraph with text “Welcome” and a button titled “Click”. When the button is clicked, then message “Hello from JavaScript” in bold should replace the paragraph text.
    - How are event handlers registered?
    - 1. Inline Event Handlers
        - Example:
    - 2. Using DOM Property
        - Example:
    - 3. Using addEventListener() Method
        - Example:
    - Html Code
- 11. What are class selectors in CSS? Suppose that it is required to display the text content of p tags in two different styles depending on the context.
    - Class Selectors in CSS
    - Solution:
        - CSS Code:
    - HTML Example:
- 12. Write CSS and the corresponding HTML code for the following:
    - HTML Code:
- 13. Develop embedded JavaScript that displays the cube of the number with font - weight 400 and font-color green, entered through a prompt box?
- 14. Using Javascript function validate an HTML form containing textboxes for entering username, password and a submit button. Validation should check the following criteria, if it’s not satisfied an alert must be given.
- 15. Write JavaScript program to find factorial of a number, use prompt dialog box to get the input from user.
- 16. How can CSS be used to display a XML document? Illustrate with an example.

- Example of Using CSS to Style an XML Document:
  - XML Document:
  - CSS File (styles.css):
- Explanation:

Important Topics

1. In-line, embedded, and external style sheets (Types of CSS)

1. **Inline Styles**
   - Styles are applied directly to an HTML element using the `style` attribute.
   - Example:

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

   - Use: Quick, small changes to a specific element.
2. **Embedded Styles (Internal)**
   - Styles are written inside a `<style>` tag in the `<head>` section of an HTML document.
   - Example:

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

   - Use: Styles specific to one HTML document.
3. **External Styles**
   - Styles are written in a separate CSS file and linked to the HTML document using a `<link>` tag.
   - Example:

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

- Use: Best for maintaining consistent styling across multiple pages.

2. z-index property in CSS

The `z-index` property in CSS controls the **stacking order** of elements on a web page. It determines which elements appear in front of or behind others when they overlap.

1. **Default Value**:
   - The default `z-index` is `auto` (treated as `0` if not set explicitly).
   - Elements with the same stacking context will appear in the order they are in the DOM.
2. **Higher Values**:
   - Elements with a higher `z-index` value appear **in front** of elements with lower values.
3. **Stacking Context**:
   - The `z-index` only works if the element has a **position** of `relative`, `absolute`, `fixed`, or `sticky` (not `static`, which is the default).

Example:

```
<div style="position: relative; z-index: 1; background: red; width: 100px;
height: 100px;">
  Red Box
</div>
<div style="position: relative; z-index: 2; background: blue; width: 100px;
height: 100px; margin-top: -50px;">
  Blue Box
</div>
```

In this example:

- The **blue box** will appear in front of the **red box** because its `z-index` is `2`, which is higher than `1`.

3. CSS method to position different HTML elements

1. Static (Default)

- Every element starts with this by default.
- The element stays where it is in the normal flow (how elements naturally stack on the page).
- **Example**:

```
div {
  position: static; /* This is the default, no movement */
}
```

2. Relative

- Moves the element **from its normal place** on the page.
- You can shift it using `top`, `left`, `right`, or `bottom`.
- **Example**:

```
div {
  position: relative;
  top: 10px; /* Moves the element 10px down */
  left: 20px; /* Moves the element 20px to the right */
}
```

3. ****Absolute***

- Removes the element from the normal flow (it no longer affects other elements).

- It is positioned **relative to its closest parent** that has positioning (or the whole page if no parent is positioned).
- **Example**:

```
div {
  position: absolute;
  top: 50px;
  left: 100px; /* Moves it to 50px from the top and 100px from the left
*/
}
```

4. Fixed

- Similar to `absolute`, but the element is positioned **relative to the screen**.
- It stays in the same spot even when you scroll.
- **Example**:

```
div {
  position: fixed;
  bottom: 0;
  right: 0; /* Stays in the bottom-right corner of the screen */
}
```

5. Sticky

- This is like a mix of `relative` and `fixed`.
- The element acts normal until you scroll to a certain point, then it **sticks** to a spot on the page.
- **Example**:

```
div {
  position: sticky;
```

```
  top: 0; /* Sticks to the top of the page when scrolling */
}
```

4. Arrays in Javascript

JavaScript arrays are a special type of object that allows you to store multiple values in a single variable. They are versatile and can hold various data types, including numbers, strings, objects, and even other arrays. Arrays are zero-indexed, meaning the first element has an index of 0.

```
const fruits = ['apple', 'banana', 'cherry'];
```

Accessing Array Elements

```
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits[0]); // Outputs: apple
```

Modifying Arrays

```
fruits[1] = 'orange'; // Changes 'banana' to 'orange'
```

Adding Elements

```
fruits.push('date'); // fruits is now ['apple', 'orange', 'cherry', 'date']
```

Popping Elements

```
fruits.pop(); // fruits is now ['kiwi', 'apple', 'orange', 'cherry']
```

Length of array

```
console.log(fruits.length); // Outputs: 3
```

5. Document Object Model concept in JavaScript

The **Document Object Model (DOM)** represents the structure of an HTML document as a tree of objects. Each element, attribute, and piece of text in the HTML document is represented as a node in the DOM tree, enabling developers to manipulate the document's content and structure programmatically.

Accessing the DOM: JavaScript provides several methods to access and manipulate DOM elements:

- `document.getElementById(id)` : Selects a single element with the specified ID.
- `document.getElementsByClassName(className)` : Selects all elements with the specified class name.
- `document.getElementsByTagName(tagName)` : Selects all elements with the specified tag name.
- `document.querySelector(selector)` : Selects the first element that matches a specified CSS selector.
- `document.querySelectorAll(selector)` : Selects all elements that match a specified CSS selector.

Examples

Changing content

```
const header = document.getElementById('header');
header.textContent = 'New Header Text';
```

```
const element = document.querySelector('.item');
element.classList.add('active'); // Adds class
element.classList.remove('active'); // Removes class
```

```
const newDiv = document.createElement('div');
newDiv.textContent = 'Hello, World!';
document.body.appendChild(newDiv); // Appends to the body
```

6. Media Queries

Media Queries in CSS

Media queries are a way to make your website **responsive**, meaning it adjusts to different screen sizes or devices like phones, tablets, and desktops.

How Media Queries Work

- Media queries apply CSS styles **only if certain conditions are met**, such as screen width or device type.
- They help create designs that look good on all devices.

Basic Syntax

```
@media (condition) {
  /* CSS rules for this condition */
}
```

Example: Changing Styles for Small Screens

```
/* Default style for all devices */
body {
  font-size: 18px;
}

/* For screens smaller than 600px */
@media (max-width: 600px) {
  body {
    font-size: 16px; /* Smaller font for small screens */
  }
}
```

Common Media Query Conditions

1. **max-width**: Applies styles if the screen width is **less than or equal** to the value.
   Example:

```
@media (max-width: 768px) {
  p {
    color: blue;
  }
}
```

2. **min-width**: Applies styles if the screen width is **greater than or equal** to the value.
   Example:

```
@media (min-width: 1024px) {
  p {
    color: red;
  }
}
```

3. **orientation**: Targets screen orientation (landscape or portrait).
   Example:

```
@media (orientation: landscape) {
  body {
    background-color: lightblue;
  }
}
```

Previous Year Questions

1. Compare Absolute Positioning and Relative positioning.

Feature	Absolute Positioning	Relative Positioning
Definition	Positions an element **relative to its nearest positioned ancestor** (or the entire page if no ancestor is positioned).	Positions an element **relative to its original position** in the normal flow.
Affects Other Elements?	**No**: The element is removed from the normal flow and doesn't affect the position of other elements.	**Yes**: The element remains in the flow, so it affects other elements around it.

Feature	Absolute Positioning	Relative Positioning
Requires a Positioned Parent?	Yes, the element uses the nearest parent with `position: relative`, `absolute`, `fixed`, or `sticky`.	No, it doesn't depend on the parent's position.
Usage	Used to place elements in a **specific spot**, like a tooltip, modal, or overlay.	Used for **slight adjustments** to the element's position (e.g., nudging it up/down/left/right).
Example	`css div { position: absolute; top: 50px; left: 100px; }`	`css div { position: relative; top: 10px; left: 20px; }`

Example in Action

Absolute Positioning

```
<div style="position: relative; width: 200px; height: 200px; background: lightgray;">
  <div style="position: absolute; top: 50px; left: 50px; background: blue; width: 100px; height: 100px;">
    I am absolutely positioned!
  </div>
</div>
```

- The **blue box** is placed 50px from the top and 50px from the left of its parent (the gray box).

Relative Positioning

```
<div style="position: relative; width: 200px; height: 200px; background: lightgray;">
  <div style="position: relative; top: 20px; left: 20px; background: green; width: 100px; height: 100px;">
    I am relatively positioned!
  </div>
</div>
```

- The **green box** is shifted 20px down and 20px right from its original position.

2. What is a call back function in JavaScript? State how it is different from normal functions

A **callback function** is a function that is **passed as an argument** to another function. It gets **called back** (executed) after the main function has completed its operation.

Example of a Callback Function:

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

function processUserInput(callback) {
  const name = "Alice";
  callback(name); // Calls the callback function
}

processUserInput(greet); // Output: Hello, Alice!
```

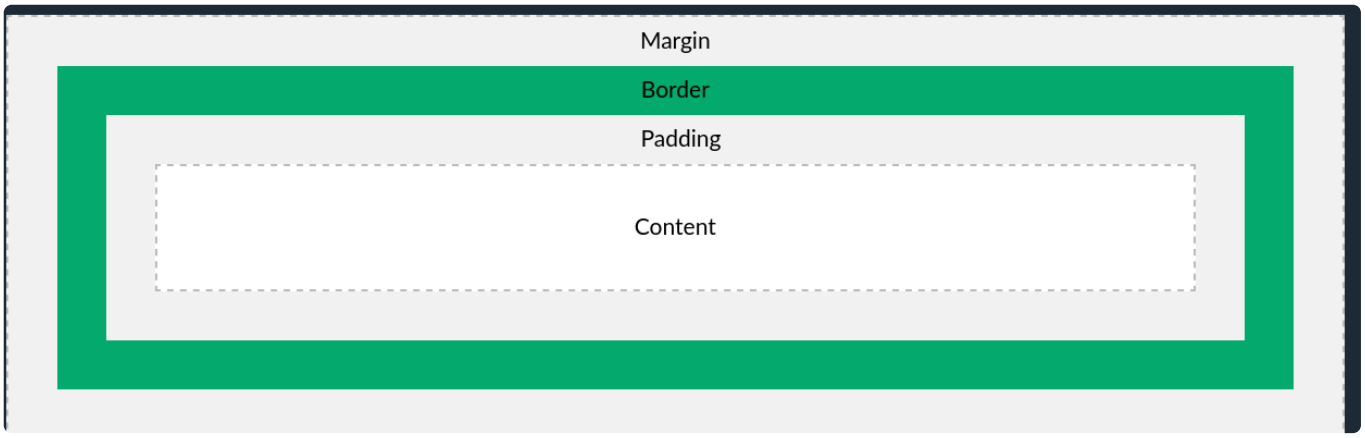
In this example:

- `greet` is the callback function passed to `processUserInput`.
- `processUserInput` calls `greet` after getting the user's name.

How is a Callback Different from Normal Functions?

Feature	Callback Function	Normal Function
Definition	A function passed as an argument to another function.	A standalone function that is called directly.
Execution Timing	Executed by the function it is passed to, usually after an event or task is complete.	Executes immediately when called.
Use Case	Commonly used in asynchronous operations like API calls, events, or timers.	Used for general tasks that don’t depend on other operations.

3. Name components of box model?



The 4 Components of the CSS Box Model

The **CSS Box Model** explains how every HTML element is represented as a rectangular box. It consists of the following components:

1. **Content**
   - The actual stuff inside the box (like text, images, or other elements).
   - The size is controlled using `width` and `height`.

   Example:

```
div {
  width: 200px;
  height: 100px;
}
```

2. **Padding**
   - The space **between the content and the border**.
   - It creates space **inside** the box around the content.
   - Controlled using `padding`, `padding-top`, `padding-right`, etc.
     Example:

```
div {
  padding: 10px;
```

```
}
```

3. **Border**

   - The edge **around the padding and content**.
   - You can style it with color, thickness, and patterns.
   - Controlled using `border`, `border-width`, `border-style`, and `border-color`.

   Example:

```
div {
  border: 2px solid black;
}
```

4. **Margin**

   - The space **outside the border**.
   - It creates space **between the element and other elements** around it.
   - Controlled using `margin`, `margin-top`, `margin-right`, etc.

   Example:

```
div {
  margin: 20px;
}
```

Component	Purpose	Position
Content	The actual content inside the box (text, images, etc.).	Innermost part
Padding	Space between the content and the border.	Inside the border
Border	A visible line around the content and padding.	Surrounds padding
Margin	Space between the border and neighboring elements.	Outermost part

4. List any 3 methods associated with javascript String object with their input parameter and return type ?

1. `charAt()`
   - **Description**: Returns the character at a specified index in the string.
   - **Input Parameter**:
     - An integer (index) specifying the position of the character.
   - **Return Type**:
     - A string containing the character at the specified index.
   - **Example**:

```
const str = "Hello";
console.log(str.charAt(1)); // Output: "e"
```

2. `toUpperCase()`
   - **Description**: Converts all characters in the string to uppercase.
   - **Input Parameter**:
     - None.
   - **Return Type**:
     - A new string with all characters in uppercase.
   - **Example**:

```
const str = "hello";
console.log(str.toUpperCase()); // Output: "HELLO"
```

3. `substring()`
   - **Description**: Extracts a part of the string between two specified indices.
   - **Input Parameters**:
     - `start` (required): The starting index.
     - `end` (optional): The ending index (non-inclusive).
   - **Return Type**:
     - A new string containing the specified part of the original string.
   - **Example**:

```
const str = "Hello World";
console.log(str.substring(0, 5)); // Output: "Hello"
```

5. Explain id selector and class selector in CSS with suitable examples

1. ID Selector

- **Definition**: An **ID selector** targets an element with a specific `id` attribute.
- **Syntax**: The `id` selector is written with a **hash (#)** symbol followed by the ID name.
- **Uniqueness**: An ID is meant to be unique within a page, meaning no two elements should share the same `id`.

Example of ID Selector:

```
<!DOCTYPE html>
<html>
<head>
  <title>ID Selector Example</title>
  <style>
    #header {
      color: blue;
      font-size: 24px;
    }
  </style>
</head>
<body>
  <h1 id="header">This is the Header</h1>
</body>
</html>
```

- **Explanation**: The `#header` selector styles the `<h1>` element with the `id="header"` and changes its text color to **blue** and font size to **24px**.

2. Class Selector

- **Definition**: A **Class selector** targets elements that have a specific `class` attribute.
- **Syntax**: The class selector is written with a **dot (.)** symbol followed by the class name.
- **Reusability**: Unlike IDs, classes can be reused on multiple elements within a page.

Example of Class Selector:

```
<!DOCTYPE html>
<html>
<head>
  <title>Class Selector Example</title>
  <style>
    .highlight {
      background-color: yellow;
      padding: 10px;
    }
  </style>
</head>
<body>
  <p class="highlight">This paragraph is highlighted.</p>
  <div class="highlight">This div is also highlighted.</div>
</body>
</html>
```

- **Explanation**: The `.highlight` class applies **yellow background** and **padding** to both the `<p>` and `<div>` elements, demonstrating that a class can be applied to multiple elements.

6. Explain array creation in JavaScript with examples.

1. Using Array Literal Syntax

The **array literal** is the most common and simple way to create an array. It is created by placing elements inside square brackets `[]`, separated by commas.

Example:

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits);  // Output: ["Apple", "Banana", "Cherry"]
```

- Here, `fruits` is an array containing three string elements: `"Apple"`, `"Banana"`, and `"Cherry"`.

2. Using the `Array` Constructor

You can also create an array using the `Array()` constructor. This method can be used in two ways:

- To create an empty array with a specified size.
- To create an array with initial elements.

Example 1 (Empty Array with a Defined Length):

```
let numbers = new Array(3);  // Creates an empty array with 3 slots
console.log(numbers);  // Output: [empty × 3]
```

- This creates an array with 3 empty slots.

Example 2 (Array with Initial Elements):

```
let colors = new Array("Red", "Green", "Blue");
console.log(colors);  // Output: ["Red", "Green", "Blue"]
```

- Here, the array is created with three colors as its elements.

3. Using `Array.of()` Method

The `Array.of()` method creates a new array instance with a variable number of elements, passed as arguments.

Example:

```
let numbers = Array.of(1, 2, 3, 4, 5);
console.log(numbers);  // Output: [1, 2, 3, 4, 5]
```

- This method is useful when you need to create an array from a set of values.

4. Using `Array.from()` Method

The `Array.from()` method creates a new array from an **array-like object** or an **iterable object** (like strings, NodeLists, etc.).

Example 1 (From a String):

```
let word = "Hello";
let letters = Array.from(word);
```

```
console.log(letters);  // Output: ["H", "e", "l", "l", "o"]
```

Example 2 (From a Set):

```
let set = new Set([1, 2, 3, 4]);
let arrayFromSet = Array.from(set);
console.log(arrayFromSet);  // Output: [1, 2, 3, 4]
```

- This method is handy for converting other data types to arrays.

Summary of Array Creation Methods:

Method	Syntax	Use Case
Array Literal	`let arr = [1, 2, 3]`	Simple and most commonly used for array creation.
Array Constructor	`let arr = new Array(3)`	Used when you need an empty array of a specific size.
Array.of()	`let arr = Array.of(1, 2, 3)`	Creates an array from a list of values.
Array.from()	`let arr = Array.from("abc")`	Converts array-like objects or iterables to arrays.

Each method has its specific use case, and you can choose the one that best fits your needs when creating arrays in JavaScript.

7. Write CSS style rules to implement the following in a web page to display the content of hyperlinks with red background colour and in italics, to display the contents of unordered lists in Times New Roman font, to display "Ocean.gif" as the background image of the page.

CSS Style Rules

```
/* 1. Hyperlinks with red background and italics */
a {
  background-color: red;
```

```
  font-style: italic;
}

/* 2. Unordered list in Times New Roman font */
ul {
  font-family: "Times New Roman";
}

/* 3. Ocean.gif as the background image of the page */
body {
  background-image: url('Ocean.gif');
  background-size: cover; /* Optional: ensures the image covers the whole
background */
}
```

Explanation:

1. **Hyperlinks (`<a>` tag):**
   - `background-color: red;` : This changes the background color of hyperlinks to **red**.
   - `font-style: italic;` : This makes the text of the hyperlinks appear in **italics**.
2. **Unordered List (`<ul>` tag):**
   - `font-family: "Times New Roman";` : This applies the **Times New Roman** font to all unordered lists.
3. **Background Image for the Page (`body` tag):**
   - `background-image: url('Ocean.gif');` : This sets **Ocean.gif** as the background image of the page.
   - `background-size: cover;` : This ensures that the background image covers the entire page area, even if the page is resized.

Example HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>CSS Styling Example</title>
  <link rel="stylesheet" href="styles.css">
</head>
```

```
<body>

  <a href="#">This is a hyperlink</a>

  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>

</body>
</html>
```

8. Write CSS code for the following:

- i) set the background color for the hover and active link states to "yellow".
- ii) Set the list style for ordered lists to "lower case alphabet".
- iii) Set "Boat.gif" as the background image of the page.
- iv) Set dotted border for the document.

```
/* i) Set the background color for the hover and active link states to
"yellow" */
a:hover, a:active {
  background-color: yellow;
}

/* ii) Set the list style for ordered lists to "lower case alphabet" */
ol {
  list-style-type: lower-alpha; /* Uses lowercase alphabet for ordered list
items */
}

/* iii) Set "Boat.gif" as the background image of the page */
body {
  background-image: url('Boat.gif');
  background-size: cover; /* Optional: Ensures the image covers the entire
page */
}
```

```
/* iv) Set a dotted border for the document */
* {
  border: 1px dotted black; /* Applies a dotted black border around every
element */
}
```

9. Discuss various types of control statements in JavaScript

Control statements are used to control the flow of the program based on certain conditions or repeat actions.

1. Conditional Statements

Conditional statements allow you to execute different blocks of code based on certain conditions.

- `if` **statement**: Executes a block of code if the condition is `true`.

```
if (x > 5) {
  console.log("x is greater than 5");
}
```

- `if-else` **statement**: Executes one block of code if the condition is `true`, otherwise executes another block.

```
if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is 5 or less");
}
```

- `else if` **statement**: Allows multiple conditions to be checked.

```
if (x > 10) {
  console.log("x is greater than 10");
} else if (x > 5) {
  console.log("x is between 6 and 10");
```

```
} else {
  console.log("x is 5 or less");
}
```

- `switch` **statement**: A simpler way to handle multiple conditions based on a single expression.

```
switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  default:
    console.log("Invalid day");
}
```

2. Looping Statements

Looping statements are used to repeat a block of code multiple times.

- `for` **loop**: Repeats a block of code a specific number of times.

```
for (let i = 0; i < 5; i++) {
  console.log(i); // Prints 0, 1, 2, 3, 4
}
```

- `while` **loop**: Repeats a block of code as long as the condition is `true`.

```
let i = 0;
while (i < 5) {
  console.log(i); // Prints 0, 1, 2, 3, 4
  i++;
}
```

- `do...while` **loop**: Executes a block of code once, and then repeats it while the condition is `true`.

```
let i = 0;
do {
  console.log(i); // Prints 0, 1, 2, 3, 4
  i++;
} while (i < 5);
```

3. Jump Statements

Jump statements allow you to break out of loops or functions and control the flow in other ways.

- **`break` statement**: Exits a loop or a `switch` statement early.

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Stops the loop when i is 5
  }
  console.log(i); // Prints 0, 1, 2, 3, 4
}
```

- **`continue` statement**: Skips the current iteration of a loop and continues with the next iteration.

```
for (let i = 0; i < 5; i++) {
  if (i === 3) {
    continue; // Skips the loop when i is 3
  }
  console.log(i); // Prints 0, 1, 2, 4
}
```

- **`return` statement**: Exits a function and optionally returns a value.

```
function add(a, b) {
  return a + b; // Exits the function and returns the result
}
```

4. Exception Handling Statements

Exception handling is used to deal with errors and exceptional conditions in a graceful manner.

- `try...catch` **statement**: Allows you to handle errors without stopping the execution of the program.

```
try {
  let result = riskyFunction();
} catch (error) {
  console.log("An error occurred: " + error.message);
}
```

- `finally` **statement**: A block of code that executes after `try...catch`, regardless of whether an error occurred or not.

```
try {
  let result = riskyFunction();
} catch (error) {
  console.log("An error occurred: " + error.message);
} finally {
  console.log("This always runs");
}
```

10. How are event handlers registered in JavaScript? Write the code for an HTML document with embedded JavaScript scripts, which initially displays a paragraph with text “Welcome” and a button titled “Click”. When the button is clicked, then message “Hello from JavaScript” in bold should replace the paragraph text.

How are event handlers registered?

1. Inline Event Handlers

An event handler can be defined directly within an HTML tag using attributes like `onclick`, `onmouseover`, etc.

Example:

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

- When the button is clicked, the `alert()` function will be triggered and display a message.

2. Using DOM Property

You can register an event handler by assigning a function to the event property of an HTML element (e.g., `onclick`, `onmouseover`, etc.).

Example:

```
<button id="myButton">Click Me</button>
<script>
  const button = document.getElementById('myButton');
  button.onclick = function() {
    alert('Button clicked using DOM property!');
  };
</script>
```

- When the button is clicked, the assigned function will be executed.

3. Using `addEventListener()` Method

Example:

```
<button id="myButton">Click Me</button>
<script>
  const button = document.getElementById('myButton');

  // Registering the event handler using addEventListener
  button.addEventListener('click', function() {
    alert('Button clicked using addEventListener!');
  });
</script>
```

- When the button is clicked, the event handler attached via `addEventListener()` will be executed.

Html Code

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Handler Example</title>
</head>
<body>

  <!-- Paragraph that will initially display "Welcome" -->
  <p id="message">Welcome</p>

  <!-- Button that will trigger the event -->
  <button id="changeTextBtn">Click</button>

  <script>
    // Get references to the HTML elements
    const button = document.getElementById('changeTextBtn');
    const message = document.getElementById('message');

    // Function to change the paragraph text when the button is clicked
    function changeText() {
      message.innerHTML = '<b>Hello from JavaScript</b>';
    }

    // Register the click event handler
    button.addEventListener('click', changeText);
  </script>

</body>
</html>
```

1. **HTML Structure**:
   - A paragraph (`<p>`) with the id `message` that initially displays the text "Welcome".
   - A button (`<button>`) with the id `changeTextBtn` that is titled "Click".
2. **JavaScript**:
   - The `getElementById()` method is used to get references to the paragraph and button.
   - The `changeText()` function changes the content of the paragraph to "Hello from JavaScript" in **bold**.

- The `addEventListener('click', changeText)` is used to register the event handler for the `click` event on the button. When the button is clicked, the `changeText()` function is triggered, and the paragraph content is updated.

11. What are class selectors in CSS? Suppose that it is required to display the text content of p tags in two different styles depending on the context.

- Style 1 – text should be displayed in blue colour, right aligned, underlined, with font style of italics and spacing between the words of text set to 2 cm.
- Style 2 – text should be displayed with a background colour of yellow, blue colour border, and with a padding of 1.5 cm
- Write the equivalent CSS style rules.

Class Selectors in CSS

Class selectors in CSS are used to apply styles to elements with a specific class attribute. The class selector is defined with a dot (`.`) followed by the class name.

For example, if you have an HTML element with a class `blue-text`, you can apply CSS rules to all elements with that class using the following syntax:

```
.blue-text {
  /* CSS rules */
}
```

Solution:

To achieve the two styles for `<p>` tags depending on the context, we will use **class selectors**. Here's how we can define the CSS for both styles:

CSS Code:

```
/* Style 1: Class "style1" */
.style1 {
  color: blue;                    /* Blue text color */
  text-align: right;               /* Right aligned text */
  text-decoration: underline;      /* Underlined text */
```

```
  font-style: italic;              /* Italic font style */
  word-spacing: 2cm;               /* Spacing between words */
}

/* Style 2: Class "style2" */
.style2 {
  background-color: yellow;       /* Yellow background */
  border: 2px solid blue;         /* Blue border */
  padding: 1.5cm;                 /* Padding around the text */
}
```

HTML Example:

Now, in the HTML, you can apply the classes to `<p>` tags

```
<!DOCTYPE html>
<html>
<head>
  <title>Class Selectors Example</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

  <!-- Applying style1 to the paragraph -->
  <p class="style1">This is a blue, right-aligned, underlined, italicized
text with 2 cm spacing between words.</p>

  <!-- Applying style2 to the paragraph -->
  <p class="style2">This text has a yellow background with a blue border and
padding.</p>

</body>
</html>
```

12. Write CSS and the corresponding HTML code for the following:

- i. Set the background color for the hover and active link states to "green"
- ii. Set the list style for unordered lists to "square".

- iii. Set "Flower.png" as the background image of the page .
- iv. Set dashed border for left and right and double border for top & bottom of a table with 2 rows.

```
/* i. Set background color for hover and active link states to green */
a:hover, a:active {
  background-color: green;
}

/* ii. Set list style for unordered lists to square */
ul {
  list-style-type: square;
}

/* iii. Set "Flower.png" as the background image of the page */
body {
  background-image: url('Flower.png');
  background-size: cover; /* Ensures the background image covers the entire
page */
}

/* iv. Set dashed border for left and right and double border for top &
bottom of a table */
table {
  border-left: 2px dashed black;
  border-right: 2px dashed black;
  border-top: 4px double black;
  border-bottom: 4px double black;
  width: 100%; /* Optional, to make the table width 100% of the page */
}

/* Optional: Adding some padding to table cells for better readability */
td {
  padding: 10px;
}
```

HTML Code:

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

```
<head>
  <title>CSS Example</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

  <!-- Unordered List with square bullets -->
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>

  <!-- Links to demonstrate hover and active states -->
  <a href="#">Hover and Active Link</a>

  <!-- Table with dashed and double borders -->
  <table>
    <tr>
      <td>Row 1, Cell 1</td>
      <td>Row 1, Cell 2</td>
    </tr>
    <tr>
      <td>Row 2, Cell 1</td>
      <td>Row 2, Cell 2</td>
    </tr>
  </table>

</body>
</html>
```

13. Develop embedded JavaScript that displays the cube of the number with font - weight 400 and font-color green, entered through a prompt box?

```
<!DOCTYPE html>
<html>
<head>
  <title>Cube of a Number</title>
```

```
  <style>
    /* CSS for styling the output */
    .output {
      font-weight: 400;
      color: green;
      font-size: 20px;
    }
  </style>
</head>
<body>

  <script>
    // Prompt user to enter a number
    var number = prompt("Enter a number to calculate its cube:");

    // Calculate the cube of the number
    var cube = number * number * number;

    // Create a div to display the result
    var resultDiv = document.createElement("div");
    resultDiv.className = "output"; // Apply the CSS class for styling
    resultDiv.textContent = "The cube of " + number + " is: " + cube;

    // Append the result to the body
    document.body.appendChild(resultDiv);
  </script>

</body>
</html>
```

14. Using Javascript function validate an HTML form containing textboxes for entering username, password and a submit button. Validation should check the following criteria, if it's not satisfied an alert must be given.

- Username must not be blank
- Password must not be less than 6 characters

```
<!DOCTYPE html>
<html>
<head>
  <title>Form Validation</title>
</head>
<body>

  <h2>Login Form</h2>
  <form name="loginForm" onsubmit="return validateForm()">
    <!-- Username Field -->
    <label for="username">Username:</label>
    <input type="text" id="username" name="username"><br><br>

    <!-- Password Field -->
    <label for="password">Password:</label>
    <input type="password" id="password" name="password"><br><br>

    <!-- Submit Button -->
    <input type="submit" value="Submit">
  </form>

  <script>
    // Function to validate the form
    function validateForm() {
      var username = document.forms["loginForm"]["username"].value;
      var password = document.forms["loginForm"]["password"].value;

      // Check if username is blank
      if (username == "") {
        alert("Username must not be blank");
        return false; // Prevent form submission
      }

      // Check if password is less than 6 characters
      if (password.length < 6) {
        alert("Password must be at least 6 characters long");
        return false; // Prevent form submission
      }

      // If both conditions are satisfied, allow form submission
```

```
      return true;
    }
  </script>

</body>
</html>
```

15. Write JavaScript program to find factorial of a number, use prompt dialog box to get the input from user.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Factorial Calculator</title>
</head>
<body>

  <h2>Factorial Calculator</h2>

  <script>
    // Function to calculate factorial
    function calculateFactorial(num) {
      let factorial = 1;

      // Loop to calculate factorial
      for (let i = 1; i <= num; i++) {
        factorial *= i;
      }
      return factorial;
    }

    // Prompt the user for input
    var number = prompt("Enter a number to find its factorial:");

    // Convert input to a number
    number = parseInt(number);
```

```
    // Check if the input is a valid number and greater than or equal to 0
    if (isNaN(number) || number < 0) {
      alert("Please enter a valid positive number.");
    } else {
      // Calculate and display the factorial
      var result = calculateFactorial(number);
      alert("The factorial of " + number + " is: " + result);
    }
  </script>

</body>
</html>
```

16. How can CSS be used to display a XML document? Illustrate with an example.

CSS can be used to style an **XML document** in a similar way to HTML, but the XML document needs to be properly linked to a CSS file. CSS can target specific XML elements, attributes, or values and apply styles accordingly.

To style an XML document, we can create a **CSS file** and reference it within the XML document using the `<?xml-stylesheet?>` processing instruction.

Example of Using CSS to Style an XML Document:

XML Document:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="styles.css"?>

<catalog>
  <book>
    <title>Learn CSS</title>
    <author>John Doe</author>
    <price>29.99</price>
  </book>
  <book>
```

```
    <title>Advanced JavaScript</title>
    <author>Jane Smith</author>
    <price>39.99</price>
  </book>
</catalog>
```

CSS File (styles.css):

```
/* styles.css */
catalog {
  background-color: lightgray;
  padding: 10px;
}

book {
  margin-bottom: 15px;
  padding: 5px;
  border: 1px solid #ccc;
}

title {
  font-size: 18px;
  font-weight: bold;
  color: darkblue;
}

author {
  font-style: italic;
}

price {
  color: green;
  font-weight: bold;
}
```

Explanation:

1. **Linking CSS to XML**: The line `<?xml-stylesheet type="text/css" href="styles.css"?>` is used to link the XML document to the external CSS file (`styles.css`).
2. **XML Elements Styled by CSS**:

- The `catalog` element has a light gray background and padding.
- Each `book` element has a margin and a border.
- The `title` element is styled with a larger font, bold text, and dark blue color.
- The `author` element is italicized.
- The `price` element has a green color and bold text.