# PHP Module 2 (Previous Year QP Discussion)

Sem 4 BCA/B.Sc. Computer Science MGU

# Module 2 Overview

- CSS introduction, <link> and <style> elements, CSS properties, Controlling Fonts, Text formatting, Text- pseudo classes, Selectors, Links, Backgrounds, lists
- Introduction to Java Script, Java Script variables, operators, decision control statements, looping, functions, arrays, events, popup boxes- alert, prompt, conform box, built-in objects, writing JavaScript, form validation

# 1. What is CSS shorthand property? Give Example [2019] [2 Marks]

- CSS shorthand properties are a way to combine multiple related CSS properties into single, more concise declaration.
- Instead of specifying  each individual properties separately, you can define it together using shorthand properties.
- Makes CSS code cleaner & more efficient.
- Eg:  To make border without using shorthand property

  border-width: 1px;

  border-style: solid

  border-color: #000;

If we are using shorthand property

  border: 1px solid #000;

# 2. What are pseudo classes in CSS [2019] [2 Marks]

- Special keywords used to select & style elements based on their current state or position in the document tree.
- allows you to apply styles to elements that cannot be selected using regular class or ID selectors.
- Denoted by a colon ":" followed by the name of the pseudo-class.
- Example for pseudo-classes in CSS:
  - :hover : Applies style to element when user hovers over it with the mouse pointer.
  - :disable: Applies style to disabled input element

# 3. List any four number object methods in JavaScript [2019] [2 Marks]

1. toFixed() : Converts a number into a string, keeping a specified number of decimals.
   - Eg:

```
const num = 10.456;
const formattedNum = num.toFixed(2); // Returns "10.46" as a string
```

2. toPrecision() : Converts a number into a string, using the specified precision. It represents the total number of significant digits (including both integer and decimal digits).
   - Eg:

```
const num = 123.456789;
const formattedNum = num.toPrecision(5); // Returns "123.46" as a string
```

3. toString() : Converts a number to a string. You can also specify the base for the string representation (2 for binary, 8 for octal, 16 for hexadecimal)
   - Eg:

```
const num = 42;
const numAsStr = num.toString(); // Returns "42" as a string
const numAsBinary = num.toString(2); // Returns "101010" as a binary string
```

# 3. List any four number object methods in JavaScript [2019] [2 Marks]

4. parseInt() : This method parses a string and returns an integer based on the specified radix (base). If the radix is not specified, it assumes base 10. It stops parsing when it encounters a non-numeric character.

```
const str = "42";
const num = parseInt(str); // Returns 42 as a number

const binaryStr = "101010";
const binaryNum = parseInt(binaryStr, 2); // Returns 42 as a number (base 2)

const invalidStr = "Hello";
const invalidNum = parseInt(invalidStr); // Returns NaN
```

# 4. Explain any five number array methods in JavaScript [2019] [5 Marks]

1. push() - adds one or more elements to the end of an array and returns the updated length of the array.

```
const numbers = [1, 2, 3];
numbers.push(4, 5); // Adds 4 and 5 to the end of the array
console.log(numbers); // Output: [1, 2, 3, 4, 5]
```

2. pop() : removes the last element from an array and returns that element.

```
const numbers = [1, 2, 3, 4, 5];
const lastNumber = numbers.pop(); // Removes 5 from the array
console.log(numbers); // Output: [1, 2, 3, 4]
console.log(lastNumber); // Output: 5
```

3. indexOf() : returns the index of the first occurrence of a specified element in an array. If the element is not found, it returns -1.

```
const fruits = ["apple", "banana", "orange", "banana"];
const index = fruits.indexOf("banana"); // Returns 1 (index of the first "banana")
```

4. forEach() : executes a provided function once for each element in the array.

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach((number) => { console.log(number * 2); // Outputs the double of each number });
```

5. map() : creates a new array by applying a provided function to each element of the original array. It returns a new array with the results of calling the provided function on each element of the original array.

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map((number) => number * 2);
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

# 5. Explain difference between Prompt Box &  Confirm Box [2019] [5 Marks]

- Both prompt boxes and confirm boxes are used to interact with users in JavaScript, but they serve different purposes.

1. Prompt Box
    - used to get input from the user.
    - displays a dialog box with a message, an input field, and "OK" and "Cancel" buttons.
    - user can type a response in the input field and choose to click "OK" or "Cancel.“
    - prompt() function is used for creating a prompt
    - Example code:

```
const userInput = prompt("Please enter your name:");
if (userInput !== null)
{
    console.log("Hello, " + userInput);
} else
{
    console.log("You cancelled the prompt.");
}
```

2. Confirm Box
    - used to get a binary choice from the user, typically for confirmation purposes.
    - displays a dialog box with a message and "OK" and "Cancel" buttons.
    - The user can choose to click "OK" or "Cancel.“
    - confirm() function is used for creating a confirm box.
    - Example Code:

```
const result = confirm("Are you sure you want to delete this item?");
if (result)
{
    console.log("Item deleted.");
}
else
{
    console.log("Deletion cancelled.");
}
```

# 6. Explain different types of Selectors in CSS. Give Examples [2019] [15 Marks]

- In CSS, selectors are patterns used to select and target HTML elements to apply specific styles. There are several types of selectors in CSS, each with its own way of targeting elements.

1. Element Selector: The most basic selector that targets elements by their HTML tag name.

   Example: HTML: Hi
   CSS: p { color: blue; }

2. Class Selector: Targets elements with a specific class attribute value.

   Example:
   HTML : This is a highlighted paragraph.

   CSS: .highlight { background-color: yellow; }

3. ID Selector : Targets an element with a specific ID attribute value. Note that IDs should be unique within an HTML document.

   Example:

   HTML This is the header.
   CSS #header { font-size: 24px; }

4. Descendent Selector : Targets an element that is a descendant of another element. It selects elements that are inside other elements.

   Example: HTML This is a paragraph inside a div.
   CSS:

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

# 6. Explain different types of Selectors in CSS. Give Examples [2019] [15 Marks]

- Child Selector: Targets an element that is a direct child of another element. It selects elements that are immediate children of a parent.

  Example:

  HTML: <ul> <li>Item 1</li> <li>Item 2</li> </ul>

  CSS: ul > li { list-style: none; }

- Attribute Selector: Targets elements based on their attribute values.

  Example:

  HTML: <a href="https://www.example.com">Visit Example</a>

  CSS: a[href^="https://"] { color: blue; }

- Pseudo-class Selector: Targets elements based on their state or position in the document tree.

  Example:

  HTML: <a href="#">Link</a>
  CSS : a:hover { text-decoration: underline; }

# 7. When is an internal CSS is used in HTML document? [2 Marks] [2020]

Internal CSS is used in an HTML document when you want to apply styles directly to specific elements or groups of elements within that particular HTML file. Internal CSS is defined within the `<style>` element, which is placed in the `<head>` section of the HTML document.

- Example

```
<html>
    <head>
     <title>Internal CSS Example</title>
        <style>
         h1 {
           color: blue;
         }
        </style>
    </head>
    <body>
     <h1>Welcome to My Website</h1>
    </body>
</html>
```

- Internal CSS is suitable for smaller HTML documents or when you need to apply specific styles unique to a particular HTML file. However, it is not recommended for larger projects or when you want to reuse styles across multiple HTML files, as this can lead to redundant code.

## 8. What is the use of new operator in JavaScript? [2 Marks] [2020]

New operator is used to create an instance of an object from a constructor function. Constructors are functions that are designed to be used with the new keyword to create objects with specific properties and methods.

# 9. Discuss CSS Font properties in detail [ 5 marks ] [ 2020 ]

1. font-family: sets the font family for an element, specifying the font(s) to be used for displaying text content. You can specify multiple font families in order of priority, separated by commas. If the browser doesn't support the first font, it will try the next one in the list.
   Example: `p { font-family: Arial, sans-serif; }`
2. font-size: sets the size of the text. It can be specified in various units like pixels (px), ems (em), or percentages (%).
   Example: p{font-size:24px;}
3. font-weight: defines the thickness or boldness of the font. It can have values like normal, bold, bolder, lighter, or numeric values (e.g., 400, 700)
   Example: p{font-weight: bold;}
4. font-style: used to specify the style of the font. It can be set to normal, italic, or oblique.
   Example: `p{font-style: italic;}`
5. text-decoration : used to add decorative elements to text, such as underlines, overlines, line-through, or blinking text.

   Example: `a { text-decoration: underline; }`

# 10. Create a String object in JavaScript & apply any four built-in String object Methods [ 5 marks ] [ 2020 ]

In JavaScript, you can create a String object by using the String constructor or by simply using a string literal (a sequence of characters enclosed in single or double quotes). When a string is created using a string literal, JavaScript automatically converts it to a String object behind the scenes, so you can use built-in String methods on it.

```
const myString = new String("Hello, World!");


console.log(myString.length); // Output: 13 (length of the string)
console.log(myString.toUpperCase()); // Output: HELLO, WORLD! (converts to uppercase)
console.log(myString.indexOf("World")); // Output: 7 (index of the substring "World")
console.log(myString.replace("World", "Universe")); // Output: Hello, Universe! (replaces "World" with "Universe")
```

length: This property returns the length of the string, which is the number of characters in the string.

toUpperCase(): This method returns a new string with all characters converted to uppercase.

indexOf(substring): This method returns the index of the first occurrence of the specified substring in the original string. If the substring is not found, it returns -1.

replace(searchValue, newValue): This method returns a new string where all occurrences of the searchValue are replaced with the newValue.

# 11. Explain various operators supported by JavaScript. Give Example [ 15 marks ] [ 2020 ]

JavaScript supports various types of operators, which are symbols or special keywords that perform operations on values or variables. These operators can be used for arithmetic, comparison, logical, assignment, and more. Some of the most commonly used operators in JavaScript along with examples:

**Arithmetic Operators**:

+: Addition operator, used to add two or more values.

-: Subtraction operator, used to subtract one value from another.

*: Multiplication operator, used to multiply two or more values.

/: Division operator, used to divide one value by another.

%: Modulus operator, used to find the remainder of a division.

Example:

```
let a = 10;
let b = 5;
let sum = a + b;  // 15
let product = a * b;  // 50
let remainder = a % b;  // 0
```

# 11. Explain various operators supported by JavaScript. Give Example [ 15 marks ] [ 2020 ]

**Comparison Operators: [Asked in 2021 for 5 marks]**

==: Equal to, checks if two values are equal (value comparison).

!=: Not equal to, checks if two values are not equal.

===: Strict equal to, checks if two values are equal and have the same data type (value and type comparison).

!==: Strict not equal to, checks if two values are not equal or have different data types.

>: Greater than, checks if the left value is greater than the right value.

<: Less than, checks if the left value is less than the right value.

>=: Greater than or equal to, checks if the left value is greater than or equal to the right value.

<=: Less than or equal to, checks if the left value is less than or equal to the right value.

```
let x = 5;
let y = 10;
let isEqual = x == y; // false
let isNotEqual = x != y; // true
let isStrictEqual = x === "5"; // false (different data types)
let isGreaterThan = y > x; // true
```

# 11. Explain various operators supported by JavaScript. Give Example [ 15 marks ] [ 2020 ]

**Logical Operators:**

&&: Logical AND, returns true if both operands are true.

||: Logical OR, returns true if at least one operand is true.

!: Logical NOT, returns true if the operand is false and vice versa.

```
let isTrue = true;
let isFalse = false;
let result1 = isTrue && isFalse; // false
let result2 = isTrue || isFalse; // true
let result3 = !isFalse; // true
```

# 11. Explain various operators supported by JavaScript. Give Example [ 15 marks ] [ 2020 ]

**Assignment Operators: [Asked in 2021 for 5 marks]**

=: Simple assignment, assigns a value to a variable.

+=: Addition assignment, adds a value to the variable's current value and assigns the result back to the variable.

-=: Subtraction assignment, subtracts a value from the variable's current value and assigns the result back to the variable.

*=: Multiplication assignment, multiplies a value with the variable's current value and assigns the result back to the variable.

/=: Division assignment, divides the variable's current value by a value and assigns the result back to the variable.

```
let num = 10;

num += 5; // num is now 15

num -= 3; // num is now 12

num *= 2; // num is now 24

num /= 4; // num is now 6
```

# 11. Explain various operators supported by JavaScript. Give Example [ 15 marks ] [ 2020 ]

**Conditional (Ternary) Operator:**

Syntax : condition ? expr1 : expr2:
The conditional operator is a shorthand way to write an if-else statement. It evaluates the condition, and if it is true, it returns expr1; otherwise, it returns expr2.

```
let age = 18;

let message = age >= 18 ? "You are an adult" : "You are a minor";
```

# 12. What is an event. Give Example [ 2 marks ] [ 2021 ]

An event is an action or occurrence that happens within the browser window or a web page. Events can be triggered by user interactions, changes in the web page's state, or specific actions performed by the browser. JavaScript allows you to respond to these events by attaching event handlers to elements on the web page. An event handler is a function that gets executed when a specific event occurs.

Example: (HTML)

```
<button id="myButton">Click Me</button>
<p id="output"></p>
```

(JS)

```
// Get a reference to the button and paragraph elements
const button = document.getElementById('myButton');
const output = document.getElementById('output');
// Define an event handler function
function handleClick() {
  output.textContent = "Button clicked!";
}
// Attach the event handler to the button element
button.addEventListener('click', handleClick);
```

# 13. Explain different decision control statements in JS [ 5 marks ] [ 2021 ]

- Decision control statements are used to make decisions based on certain conditions and execute different blocks of code accordingly. These statements allow the program to take different paths based on the evaluation of a condition. There are four main decision control statements in JavaScript:
- If statement : Used to execute a block of code if a specified condition is true. If the condition is false, the code inside the if block is skipped.

Syntax:

```
if (condition) {
  // Code to be executed if the condition is true
}
```

Example:

```
let age = 20;
if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

# 13. Explain different decision control statements in JS [ 5 marks ] [ 2021 ]

- If…else statement

  The if...else statement is an extension of the if statement. It allows you to execute one block of code if the condition is true and a different block of code if the condition is false.

  Syntax:

```
if (condition) {
      // Code to be executed if the condition is true
    } else {
      // Code to be executed if the condition is false
    }
```

Example:

```
let hour = 14;


if (hour < 12) {
  console.log("Good morning!");
} else {
  console.log("Good afternoon!");
}
```

# 13. Explain different decision control statements in JS [ 5 marks ] [ 2021 ]

- if...else if...else Statement

  The if...else if...else statement allows you to test multiple conditions one by one and execute different blocks of code based on the first condition that evaluates to true.

- Syntax

```
if (condition1) {
    // Code to be executed if condition1 is true
  } else if (condition2) {
    // Code to be executed if condition2 is true
  } else {
    // Code to be executed if all conditions are false
  }
```

- Example

```
let score = 85;
if (score >= 90) {
  console.log("Excellent!");
} else if (score >= 80) {
  console.log("Good job!");
} else {
  console.log("Keep trying!");
}
```

# 13. Explain different decision control statements in JS [ 5 marks ] [ 2021 ]

- **Switch statement**: which is another type of decision control statement. The switch statement is used to select one of many code blocks to be executed based on the value of an expression.
- The switch statement provides an alternative way to write multiple if...else if...else statements when you have to compare a single value against multiple possible values.

Syntax:

```
switch (expression) {
      case value1:
        // Code to be executed when expression matches value1
        break;
      case value2:
        // Code to be executed when expression matches value2
        break;
     // Add more cases as needed
      default:
        // Code to be executed when none of the cases match the expression
        break;
}
```

Example:

```
let day = "Tuesday";
        switch (day) {
          case "Monday":
            console.log("It's Monday!");
            break;
          case "Tuesday":
            console.log("It's Tuesday!");
            break;
          case "Wednesday":
            console.log("It's Wednesday!");
            break;
          default:
            console.log("It's some other day.");
            break;
        }
```

# 14. What are the different methods used to embed CSS into an HTML document ?[ 15 marks ] [ 2021 ]

There are 3 methods to embed CSS into an HTML document. Each method allows you to apply styles to the HTML elements and control the presentation of the content. Here are the different methods to embed CSS:

1. Inline CSS

   In this method, you apply CSS styles directly to individual HTML elements using the style attribute. It's the most immediate and specific way to apply styles but not recommended for large-scale styling as it mixes content with presentation.

   Example:

   ```
   <p style="color: blue; font-size: 16px;">This paragraph has inline CSS styles.</p>
   ```

2. External CSS

   In this method, you create a separate CSS file with the .css extension and link it to the HTML document using the <link> element. This approach keeps the styling separate from the content, promoting better organization and maintainability.

   Example:

   ```
   <!DOCTYPE html>
   <html><head> <title>External CSS Example</title>
      <link rel="stylesheet" type="text/css" href="styles.css"> </head>
     <body>  <p>This paragraph has styles from an external CSS file.</p> </body>
     </html>
   ```

# 14. What are the different methods used to embed CSS into an HTML document ?[ 15 marks ] [ 2021 ]

3. Internal CSS

In this method, you define CSS styles within the <style> element in the <head> section of the HTML document. The styles defined inside the <style> element will apply to the entire HTML document.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal CSS Example</title>
  <style>
    p {
      color: blue;
      font-size: 16px;
    }
  </style>
</head>
<body>
  <p>This paragraph has internal CSS styles.</p>
</body>
</html>
```