JavaScript Assignment

❖ JavaScript Introduction

❖ Theory Assignment

Question 1: What is JavaScript? Explain the role of JavaScript in web development.

Answer: -

JavaScript is a high-level, dynamic, and interpreted programming language primarily used for adding interactivity and responsiveness to web pages. It's a client-side scripting language, meaning it runs on the user's web browser, rather than the server.

Question 2: How is JavaScript different from other programming languages like Python or Java?

Answer: -

JavaScript's unique characteristics, such as its scripting nature, dynamic typing, and focus on client-side web development, set it apart from Python and Java. While all three languages share same similarities, their design goals and use cases have led to distinct differences in syntax, libraries, and programming paradigms.

Question 3: Discuss the use of <script > tag in HTML. How can you link an external JavaScript file to an HTML document?

Answer: -

The <script> tag in HTML is used to embed or reference JavaScript code. To link an external JavaScript file, use the src attribute: <script src="file.js"></script>.

❖ Variables and Data Types

❖ Theory Assignment

Question 1: What are variables in JavaScript? How do you declare a variable using var, let, and const?

Answer: -

1. **var: Function-scoped, can be redeclared and updated.**
   Example: var x = 10;

2. **let: Block-scoped, cannot be redeclared but can be updated.**
   Example: let y = 20;

3. **const: Block-scoped, cannot be redeclared or updated (must be initialized).**
   Example: const z = 30;

Question 2: Explain the different data types in JavaScript. Provide examples for each.

Answer: -

- String**: Represents text. Example: let name = "Madhav";**
- Number**: Represents numeric values (integers or decimals). Example: let age = 23;**
- Boolean**: Represents true or false. Example: let isActive = true;**
- Null**: Represents an intentional absence of value. Example: let emptyValue = null;**
- Undefined**: Represents a variable declared but not assigned. Example: let unassigned;**
- Object**: Represents complex data structures (key-value pairs). Example: let user = { name: "Madhav", age: 23 };**
- Array**: A type of object for lists. Example: let colors = ["red", "green", "blue"];**

Question 3: What is the difference between undefined and null in JavaScript?

Answer: -

- **undefined: Indicates a variable has been declared but not assigned a value.**
- **null: Represents an intentional absence of value, explicitly set by the developer**

❖ JavaScript Operators

❖ **Theory Assignment**

Question 1: What are the different types of operators in JavaScript? Explain with examples.

Answer: -

Arithmetic Operators: Perform basic math operations.

```
<script>
  let sum = 5 + 3; // Addition
  let product = 5 * 3; // Multiplication
</script>
```

Assignment Operators: Assign values to variables.

```
<script>
  let x = 10; // Assign
  x += 5; // Add and assign (x = x + 5)
</script>
```

Comparison Operators: Compare values and return a Boolean.

```
<script>
  5 == "5"; // true (loose equality)
  5 === "5"; // false (strict equality)
</script>
```

Logical Operators: Perform logical operations.

```
<script>
  true && false; // false (AND)
  true || false; // true (OR)
</script>
```

Bitwise Operators: Work on binary representations.

```
<script>
  5 & 1; // 1 (AND)
  5 | 1; // 5 (OR)
</script>
```

Question 2: What is the difference between == and === in JavaScript?

Answer: -

== (Equality Operator): Compares values after type conversion (loose equality).
=== (Strict Equality Operator): Compares values without type conversion (strict equality).

❖ Control Flow (If-Else, Switch)

❖ Theory Assignment

Question 1: What is control flow in JavaScript? Explain how if-else statements work with an example.

Answer: -

Control flow in JavaScript refers to the order in which code is executed, determined by structures like conditionals, loops, and functions.

If-else statements allow conditional execution of code based on a condition:

- **if: Executes code if the condition is true.**
- **else: Executes code if the if condition is false.**
- **else if: Adds additional conditions.**

```
<script>
let score = 75;

  if (score >= 90) {
    console.log("Grade: A");
  } else if (score >= 75) {
    console.log("Grade: B");
  } else {
    console.log("Grade: C");
  }

</script>
```

Question 2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

Answer: -

A switch statement in JavaScript executes a block of code based on the value of an expression, making it useful for multiple conditions.

```
<script>
let day = 3;

switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
    break;
  default:
    console.log("Invalid day");
}

</script>
```

❖ **Loops (For, While, Do-While)**
❖ **Theory Assignment**

Question 1: Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.
Answer: -

1. for Loop
Executes a block of code a fixed number of times.
Syntax: for (initialization; condition; increment/decrement)

```
<script>

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

</script>
```

2. while Loop
Executes a block of code as long as the condition is true.
Syntax: while (condition)

```
<script>

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

</script>
```

3. do-while Loop
Executes the block of code at least once, and then repeats as long as the condition is true.
Syntax: do { // Code to execute } while (condition);

```
<script>

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

</script>
```

Question 2: What is the difference between a while loop and a do-while loop?

Answer: -

The primary difference between a while loop and a do-while loop lies in when the condition is checked: -

while Loop:

- **The condition is checked before the loop's body is executed.**
- **If the condition is false initially, the loop body will not execute even once.**

do-while Loop:

- **The loop's body is executed at least once, regardless of the condition.**
- **The condition is checked after the loop's body runs.**

❖ Functions

❖ **Theory Assignment**

Question 1: What are functions in JavaScript? Explain the syntax for declaring and calling a function.

Answer: -

In JavaScript, a function is a block of code designed to perform a particular task. Functions allow for reusable code and can accept inputs (parameters) and return a value.

Syntax for Declaring a Function:
function functionName(parameters) { // Code to be executed return value; // Optional return statement }

Question 2: What is the difference between a function declaration and a function expression?

Answer: -

- **Function Declaration: Hoisted, defined by function keyword.**
- **Function Expression: Not hoisted, can be anonymous, stored in variables.**

Question 3: Discuss the concept of parameters and return values in functions?

Answer: -

- **Parameters: Variables that accept values when the function is called.**
- **Return Values: The output produced by the function, accessible outside the function after execution.**

❖ **Arrays**
❖ **Theory Assignment**

Question 1: What is an array in JavaScript? How do you declare and initialize an array?

Answer: -

An array in JavaScript is a special variable used to store multiple values in a single variable. It allows you to work with collections of data, such as a list of numbers, strings, or objects.
Declaring an Array:
You can declare an array using either the array literal syntax or the Array constructor.

- **Array Literal Syntax:**

The most common and simple way to declare an array.
let colors = ["red", "green", "blue"];

- **Array Constructor Syntax:**

You can also use the Array constructor, but it's less common.
let colors = new Array("red", "green", "blue");

Question 2: Explain the methods push(), pop(), shift(), and unshift() used in arrays.

Answer: -

- **push(): Adds elements to the end of the array.**
- **pop(): Removes the last element from the array.**
- **shift(): Removes the first element from the array.**
- **unshift(): Adds elements to the beginning of the array.**

❖ Objects
❖ **Theory Assignment**

Question 1: What is an object in JavaScript? How are objects different from arrays?

Answer: -

In JavaScript, an object is a collection of key-value pairs, where each key (also known as a property) is a string (or symbol), and each value can be any data type, including other objects or arrays. Objects are often used to represent real-world entities with attributes and behaviours.

- **Objects are collections of key-value pairs and are typically used for more complex structures (e.g., representing a person or a car).**
- **Arrays are ordered collections of elements and are best suited for lists or sequences of data.**

Question 2: Explain how to access and update object properties using dot notation and bracket notation.

Answer: -

Dot notation is the most commonly used method to access or update object properties. It uses a period (.) followed by the property name.

Accessing Properties:

```
<script>

let person = {
    name: "John",
    age: 30
  };

  console.log(person.name);  // Output: John

</script>
```

Updating Properties:

```
<script>

  person.age = 31;
  console.log(person.age);  // Output: 31

</script>
```

❖ **JavaScript Events**

❖ **Theory Assignment**

Question 1: What are JavaScript events? Explain the role of event listeners.

Answer: -

In JavaScript, events are actions or occurrences that happen in the system (such as user interactions or changes to the state of the browser) that can be detected and handled by JavaScript. Common events include actions like clicking a button, moving the mouse, or submitting a form.

Types of JavaScript Events:

- **Mouse Events: click, dblclick, mouseover, mouseout, etc.**
- **Keyboard Events: keydown, keyup, keypress**
- **Form Events: submit, focus, blur**
- **Window Events: load, resize, scroll**
- **Touch Events: touchstart, touchmove, touchend**
- **Input Events: change, input**

Event Listeners:

An event listener is a function or callback that listens for specific events and executes a specified action when that event occurs. You attach an event listener to an element (like a button, input field, etc.) to trigger a response when a user interacts with that element.

Adding an Event Listener:

You use the addEventListener() method to attach an event listener to an element. The method requires two arguments:

1. **The name of the event (e.g., 'click', 'keydown').**
2. **The callback function to be executed when the event occurs.**

Question 2: How does the addEventListener() method work in JavaScript? Provide an example.

Answer: -

Syntax:

element.addEventListener(event, callback, useCapture);

```html
<button id="myButton">Click Me</button>

<script>
    // Select the button element
    let button = document.getElementById("myButton");

    // Attach a click event listener
    button.addEventListener("click", function () {
        alert("Button was clicked!");
    });
</script>
```

❖ DOM Manipulation
❖ Theory Assignment

Question 1: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

Answer: -

The DOM (Document Object Model) is a programming interface for web documents. It represents the structure of a web page in a tree-like format, where each element (such as HTML tags, text, and attributes) is a node in the tree. The DOM allows programming languages (like JavaScript) to interact with the structure of the web page dynamically. JavaScript uses the DOM to read, modify, and manipulate the content and structure of an HTML or XML document.
In simpler terms, the DOM is a representation of the HTML document as an object-oriented tree structure, where each part of the page is an object that can be accessed and modified using JavaScript.

JavaScript interacts with the DOM to perform various tasks such as:

- **Reading the content of HTML elements.**
- **Modifying the content of elements.**
- **Adding or removing elements.**
- **Changing the style of elements.**
- **Handling user interactions like clicks, key presses, etc.**

Question 2: Explain the methods getElementById(), getElementsByClassName(),and querySelector() used to select elements from the DOM?

Answer: -

In JavaScript, the DOM (Document Object Model) provides several methods to select and interact with elements on a webpage.
Three commonly used methods for selecting elements are:

1. **getElementById():**
   - **Purpose: This method is used to select a single element with a specific id attribute.**
   - **Returns: A single DOM element, or null if no element with the specified id is found.**

Syntax:
```
let element = document.getElementById("idName");
```

2. **getElementsByClassName():**
   - **Purpose: This method is used to select all elements that have a specified class name. It returns a live HTMLCollection of elements.**
   - **Returns: A live HTMLCollection of elements with the given class name. An HTMLCollection is a collection of nodes, but it is not an array (i.e., it lacks array methods like forEach()).**

Syntax:
```
let elements = document.getElementsByClassName("className");
```

3. **querySelector():**
   - **Purpose: This method is used to select the first element that matches a CSS selector. It allows more flexible and powerful selections compared to getElementById() and getElementsByClassName(), as it can use any valid CSS selector.**
   - **Returns: The first element that matches the given CSS selector, or null if no matching element is found.**

Syntax:
```
let element = document.querySelector("selector");
```

❖ JavaScript Timing Events (setTimeout, setInterval)

❖ **Theory Assignment**

Question 1: Explain the setTimeout() and setInterval() functions in JavaScript. However they used for timing events?

Answer: -

In JavaScript, setTimeout() and setInterval() are two functions used to handle timing events, allowing you to execute a function after a certain delay or repeatedly at regular intervals.

1. **setTimeout() Function:**
   Purpose: The setTimeout() function is used to execute a function or a piece of code once after a specified delay (in milliseconds).

Syntax:
setTimeout(callback, delay, arg1, arg2, ...);

2. **setInterval() Function:**
   Purpose: The setInterval() function is used to execute a function or a piece of code repeatedly, with a fixed time delay between each execution.

Syntax:
setInterval(callback, interval, arg1, arg2, ...);

Question 2: Provide an example of how to use setTimeout() to delay an action by 2 seconds.

Answer: -

```
<script>

console.log("Action will happen in 2 seconds...");

 setTimeout(function () {
   console.log("Action executed after 2 seconds!");
}, 2000);  // 2000 milliseconds = 2 seconds


</script>
```

❖ JavaScript Error Handling

❖ **Theory Assignment**

Question 1: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.

Answer: -

Error handling in JavaScript allows you to handle runtime errors gracefully, preventing the application from crashing and ensuring that you can manage unexpected situations in a controlled way. In JavaScript, errors are usually caught using a try...catch block, and you can optionally use a finally block to run code after error handling, regardless of whether an error occurred.

The try, catch, and finally Blocks

1. **try Block:**
    - **The code that might throw an error is placed inside the try block. If an error occurs, the execution will immediately jump to the catch block (if defined).**
2. **catch Block:**
    - **If an error is thrown inside the try block, the code in the catch block will execute. The catch block can access the error object, which contains information about the error.**
3. **finally Block:**
    - **The finally block is optional and executes after the try and catch blocks, regardless of whether an error occurred or not. It is typically used to clean up resources (e.g., closing files, releasing locks).**

Syntax:

```
try {
  // Code that may throw an error
} catch (error) {
  // Code to handle the error
} finally {
  // Code that will always run (optional)
}
```

```
<script>

try {
    let result = 10 / 0;  // This will not throw an error but will result in Infinity
    console.log("Result: " + result);

    // Simulating an error by trying to access an undefined variable
    let obj = undefined;
    console.log(obj.someProperty);  // This will throw a TypeError
} catch (error) {
    console.log("An error occurred: " + error.message);  // This will catch the error
} finally {
    console.log("This will always run, no matter what.");
}

</script>
```

Output:

Result: Infinity
An error occurred: Cannot read properties of undefined (reading 'someProperty')
This will always run, no matter what.

Question 2: Why is error handling important in JavaScript applications?

Answer: -

Error handling in JavaScript is vital for maintaining a stable, user-friendly, and secure application. It allows you to deal with unexpected events gracefully, improves the robustness of your code, and enhances the overall user experience. By catching and handling errors properly, you ensure your application runs smoothly even in the face of problems.