**THEORY ASSIGNMENT QUESTIONS**

* **JavaScript Introduction: -**

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

* JavaScript is a scripting language that runs on the web browsers to perform task(script) like handling user interactions (click, form submission).
* **Role of JavaScript: -**
* JavaScript allows you to add interactive features to a website, such as button, forms, sliders and animations and Dynamic Content, Form Validation.
* **For Example: -** When you click a button and something changes on the page without reloading, that’s JavaScript work.

1. **How is JavaScript different from other programming languages like Python or Java?**

* JavaScript, Python and Java are all popular programming languages, but they have different purposes.
* JavaScript is mainly used for making websites interactive and runs in web browsers.
* Python is known for its simplicity and is used in various areas like web development, data Science, automation.
* Java is often used for building large applications, especially for businesses and Android apps.

1. **Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML Document?**

* The <script> tag in HTML is used to include JavaScript code inside web page.
* This tag allows you to either write JavaScript directly within the HTML document or link to an external JavaScript file.
* The <script> can be placed in the <head> or <body> section of an HTML document, depending on when you want the script to run.
* **Linking an External JavaScript File to an HTML document: -**
* To link an external JavaScript file, you use the <script> tag with the src attribute, which specifies the path to the JavaScript file.
* <script src=” path-to-your-file”></script>.
* **Variables & Data Types: -**

1. **What are the variables in JavaScript? How do you declare a variable using var, let and const? ­­­**

* Variables are Containers used to store data values. You can use three keywords to declare variables: var, let and const.
* **Using Var**: -
* Function Scoped.
* Can be updated and re-declared within its scope.
* **Example: -** var name = “Nisarg”;

Console.log(“name”); // output – Nisarg

name = “Patel”;

Console.log(“name”); // output – Patel

Var name = “JavaScript”; // variable re-declared

Console.log(“name”); // output – JavaScript

* **Using Let: -**
* Block-Scoped.
* Can be updated but cannot be re-declared in the same scope.
* **Example**: - let age = 25;

Console.log(“age”); // output – 25

age = 26;

Console.log(“age”); // output – 26

let age = 30 // Error – can not re-declare the variable in the same scope.

* **Using Const**: -
* Block-Scoped.
* Can not be updated and re-declared after initialization.
* **Example**: - const PI = 3.14;

Console.log(“PI”); //output – 3.14

PI = 3.14159; // Error – Cannot update a const variable

Const PI = 22/7; // Error – Cannot Re-declare the const variable.

1. **Explain the different datatype in JS. Provide example for each.**

* JavaScript has two categories of datatypes – Primitive & Non-Primitive
* **1. Primitive Datatype: -**
* These are basic data types that represent single value.
* **Number** – used for numeric values (integer, decimals).
* **Example** – let age = 25; // Integer

let price = 99.99 //Decimal

console.log (age, price); // output – 25, 99.99

* **String** – used for text enclosed in single (‘) or (“) quotes.
* **Example** – let name = “Nisarg”;

Let greeting = ‘Hello’;

Console.log (name, greeting); // output – Nisarg, Hello

* **Boolean –** Represent either true or false.
* **Example –** let isAdult = true;

Let isStudent = false;

console.log (isAdult, isStudent); // output – true, false

* **Undefined –** A variable declared but not assigned a value is undefined.
* **Example –** **let a;**

Console.log(a); // output – Undefined

* **Null –** Represent an intentional absence of value.
* **Example –** let emptyValue = null;

Console.log(emptyValue); // output – null

* **Symbol – Used to create unique identifiers.**
* **Example –** let symbol1 = Symbol(“id”);

let symbol2 = Symbol(“id”);

console.log (symbol1 === symbol2); output – false (Symbols are unique)

* **BigInt –** Used for numbers larger than Number.MAX\_SAFE\_INTEGER.
* **Example –** let bigNumber = 123456677899675766580039n;

Console.log(bigNumber); //output- 123456677899675766580039n

* **2. Non-Primitive Datatype: -**
* **Object –** A collection key-value pairs.
* **Example –** let person = {

Name = “Nisarg”,

Age = 25

};

Console.log(person.name); //output – Nisarg

* **Array** – A list like collection of values.
* **Example** –

let colors = ["red", "green", "blue"];

console.log (colors [1]); // Output: green

* **Function** – A reusable block of code.
* **Example** –

function greet (){

Return “Hello, I am Nisarg Patel”;

}

Console.log (greet ()); // output – Hello, I am Nisarg Patel

* **Date** – Used to work with the dates and times.
* **Example** –

let today = new Date ();

Console.log(today); // output – Current Date and Time

1. **What is the difference between undefined and null in JavaScript?**

|  |  |
| --- | --- |
| **Undefined** | **Null** |
| A Variable can be declared but cannot assigned a value. | Represents an intentional absence of value. |
| Undefined is a Primitive type. | Null is a Primitive type. |
| Automatically assigned by JS. | Manually Assigned by the developer. |
| let a; (No Value assigned) | Let a = null; (Value Intentionally empty) |
| Console.log(a); // undefined | Console.log(a); // null |

* **JavaScript Operators: -**

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

* JavaScript Provides various types of operators to perform operations on variables and values.
* **1. Arithmetic Operator: -**
* **Example: -** let a = 10, b = 20

Console.log(a+b); // output – 30

Console.log(a\*b); // output – 200

* **2. Assignment Operator: -**
* **Example**: - let a = 10;

a += 23; // output – 33

* **3. Comparison Operator: -**
* **Example**: - let a = 10, b = 20;

Console.log(a<b); // output – true

Console.log (a === b); // output – false

* **4. Logical Operator: -**
* **Example**: - let x = 10, y = 20;

console.log(x > 5 && y > 15); // Output: true (both conditions true)

console.log (x > 15 || y > 15); // Output: true (at least one condition true)

console.log (! (x > 15)); // Output: true (negation of false)

1. **What is the difference between ==and ===in JavaScript?**

|  |  |
| --- | --- |
| **==** | **===** |
| Compares values only, ignores data types. | Compares values and data types. |

* **Control Flow (if-else, switch): -**

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

* Control flow refers to the order in which statements in a program are executed.
* JavaScript executes code sequentially, but control flow structures like conditional statements(if-else), loops and functions allow for decision making and repeating block of code.
* **How if-else statements work: -**
* The if-else statements are used to execute a block of code only if a condition is true. If the condition is false, the code in the else block is executed.
* **Example: -** let age = 18;

If (age >= 18) {

Console.log (“You are eligible for a vote.”)

} else {

Console.log (“You are not eligible for a vote.”)

}

1. **Describe how switch statements work in JavaScript. When should you use a switch statements instead of if-else?**

* The switch statements are used to perform different actions based on different conditions.
* It is a cleaner way to compare a variable or expression against multiple possible values.
* **Example: -**

let day = 3;

switch (day) {

case 1:

console.log("Monday");

break;

case 2:

console.log("Tuesday");

break;

case 3:

console.log("Wednesday"); // This block runs

break;

case 4:

console.log("Thursday");

break;

default:

console.log ("Invalid day");

}

* **Loops (For, While, Do-while): -**

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

* Loops allow you to repeat a block of code until a specific condition.
* **JavaScript Provides Three main types of loops: -**
* **1. For Loop: -**
* Used when the number of iterations is known.
* Consist of three parts – initialization, condition & increment/decrement.
* **Example –** for (let I=1; I<=5; I++) {

Console.log (“Iteration:”, i);

}

* **2. While Loop: -**
* Used when the number of iterations is not known beforehand.
* **Example: -** let count = 1;

While (count <= 5) {

Console.log (“count:”, count);

Count++;

}

* **3. Do-while Loop: -**
* The condition is checked **after** executing the loop.
* **Example: -** let num = 1;

do {

console.log ("Number:", num);

num++;

} while (num <= 5);

1. **What is the difference between a while loop and do-while loop?**

|  |  |
| --- | --- |
| **While loop** | **Do-while loop** |
| The condition is checked before executing the loop body. | The condition is checked after executing the body. |
| May not execute at all if the condition is false initially. | Executes the loop body at least once, even if the condition is false. |
| Use while-loop condition-dependent execution. | Do-while loop at lease one-time execution, even if the condition is initially false. |

* **Functions: -**

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

* A function is a reusable block of code that performs a specific task.
* Functions make your code modular, reduce repetition and improve readability.
* **Syntax for declaring & calling a function: -**
* **1. Function Declaration: -**
* A named function is defined using the function keyword.
* **Syntax – function functionName(parameters){**

// code to be executed

}

* **2. Calling a Function: -**
* A function is called by its name followed by parentheses.
* If the function accepts parameters, pass them inside the parentheses.
* **Syntax – function(arguments);**
* **Complete Example – function addNumbers (a, b) {**

**Return a + b;**

**}**

**let result = addNumbers (5, 10) {**

**Console.log (“The Sum is:”, result); // output – 15**

1. **What is the difference between function declaration and a function expression?**

|  |  |
| --- | --- |
| Function Declaration | Function Expression |
| A function declaration must have a function name. | A function expression is similar to the function declaration without the function name. |
| Function declaration does not require a variable assignment. | Function expressions can be stored in a variable assignment. |
| They are executed before any other code. | Function expressions load and execute only when the program interpreter reaches the line of code. |
| Function declaration can be accessed before and after the function definition. | Function expression can be accessed only after the function definition. |
| Function declarations are hoisted. | Function expressions are not hoisted. |

1. **Discuss the concept of parameters and return values in functions.**

* Functions in JavaScript can accept inputs(parameters) and produce outputs (return values).
* **1. Parameters**
* Parameters are variables listed in the function definition.
* They act as placeholders for values(arguments) passed when the function is called.
* Parameters are optional. A function can have a zero, one or multiple parameters.
* Default values can be assigned to parameters to handle cases where no argument is passed.
* **2. Return values**
* The return statement stops further execution of the function.
* If no return statements are used, the function is undefined.
* The return statement stops further execution of the function.
* Functions can return any type of value (number, string, array, object).
* **Arrays**

1. **What is an array in JavaScript? How do you declare and initialize an array?**

* An array is a data structure in JavaScript that can store multiple values in a single variable.
* It is used to group related data, such as a list of numbers, strings, or objects.
* Arrays can store values of **different data types.**
* **Declaring and Initializing an Array: -**
* **1. Using Square Brackets –**

let fruits = ["Apple", "Banana", "Cherry"];

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

* **2. Using Array Constructor –**

let numbers = new Array (1, 2, 3, 4);

console.log(numbers); // Output: [1, 2, 3, 4]

1. **Explain the methods push (), pop (), shift (), and unshift () used in arrays.**

* These methods are used to **add or remove elements** from arrays.
* **1. Push () –**
* Adds one or more elements to the end of array.
* **Syntax –** array.push(elements1, elements2,…..);
* **Example –**

let fruits = ["Apple", "Banana"];

fruits.push("Cherry", "Mango");

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

* **2. Pop () –**
* Removes the last elements of an array.
* **Syntax** - array.pop ();
* **Example** –

let fruits = ["Apple", "Banana", "Cherry"];

let removed = fruits.pop ();

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

console.log(removed); // Output: "Cherry"

* **Objects**

1. **What is an object in JavaScript? How are objects different from arrays?**

* An **object** in JavaScript is a collection of key-value pairs, where each is a string and its corresponding value can be of any data type (number, string, array, function, or even another object).
* Objects are used to represent and organize data in a structured way.
* **Syntax - Creating object (Using Object Literal Notation)**

let objectName = {

key1: value1,

key2: value2,

// ...

};

* **Example** –

let person = {

name: "Nisarg",

age: 25,

isStudent: true

};

console.log(person); // Output: {name: "Nisarg", age: 25, isStudent: true}

* **How Objects are different from arrays –**

|  |  |
| --- | --- |
| Object | Array |
| Stores key-value pairs. | Stores elements in an ordered list. |
| keys are strings or symbols. | Indices are numbers starting from 0. |
| Values can be any datatype (number, string, array, function). | Elements can also be any datatype, but they are accessed by index. |
| Access values using dot notation(object.key) or bracket notation(object[“key”]). | Access elements using their index (array [0]). |

1. **Explain how to access and update object properties using dot notation and bracket notation.**

* **1. Dot Notation –**
* **Syntax –** objectName.propertyName
* **Example –**

let person = {

name: "Nisarg",

age: 25

};

console.log(person.name); // Output: "Nisarg"

person.age = 26;

console.log(person.age); // Output: 26

* **2. Bracket Notation –**
* **Syntax -** objectName["propertyName"]
* **Example –**

let person = {

"first name": "Nisarg",

age: 25

};

console.log (person ["first name"]); // Output: "Nisarg"

person["age"] = 26;

console.log(person["age"]); // Output: 26

let key = "age";

console.log(person[key]); // Output: 26

* **JavaScript Events**

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

* JavaScript events are actions or occurrences that happen in the browser, such as a user clicking a button, hovering over an element, or a page finishing loading.
* **Role of Event Listeners –**
* An **event listener** is a function in JavaScript that waits for a specific event to occur on an element, then executes a piece of code in response.
* **Adding event listeners –**
* **1. Using addEventListerners () Method**
* The addEventListener () method allows you to attach an event listener to an element.
* **Syntax** - element.addEventListener(eventType, functionName);
* **Example** –

let button = document. GetElementById("myButton");

button.addEventListener ("click", function () {

alert ("Button clicked!");

});

* **2. Using Inline Attributes –**
* You can add event handlers directly in the HTML using attributes like onclick.
* **Example –**

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

* **3. Using DOM Properties –**
* You can assign a function to an event property of an element.
* **Example –**

let button = document.getElementById ("myButton");

button.onclick = function () {

alert ("Button clicked!");

};

1. **How does the addEventListener()method work in JavaScript? Provide an example.**

* The addEventListener () method allows you to attach an event listener to a specified element.
* When the specified event occurs, the listener executes a function (called the **callback function**) to handle the event.
* **Example** –

<!DOCTYPE html>

<html>

<head>

<title>addEventListener Example</title>

</head>

<body>

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

<script>

const button = document.getElementById("myButton");

button.addEventListener("click", function () {

alert ("Button was clicked!");

});

</script>

</body>

</html>

* **DOM Manipulation**

1. **What is the DOM (Document Object Model) in JavaScript? How does JavaScriptinteract with the DOM?**

* The **Document Object Model (DOM)** is a programming interface provided by the browser to represent an HTML or XML document as a tree of objects.
* Each element in the document (like <div>, <p>, or <body>) is represented as a **node** in the DOM tree, allowing JavaScript to interact with and manipulate the content, structure, and styles of the document.
* **How does JavaScriptinteract with the DOM-**
* JavaScript interacts with the DOM using the **document** object, which provides methods and properties to:

1. Access elements in the DOM.
2. Modify content, structure, and attributes of elements.
3. Add or remove elements dynamically.
4. Handle events triggered by user actions.
5. **Explain the methods getElementById (), getElementsByClassName (), and querySelector () used to select elements from the DOM.**

* JavaScript provides several methods to select elements from the DOM, enabling developers to manipulate and interact with them. Here, we’ll explore three commonly used methods: getElementById(), getElementsByClassName(), and querySelector().
* **1. getElementById ()**
* Selects **a single element** by its unique id attribute.
* Returns the first matching element or null if no element is found.
* **Syntax - document.getElementById("id");**
* **Example –**

<!DOCTYPE html>

<html>

<body>

<h1 id="header">Welcome</h1>

<script>

let element = document.getElementById("header");

console.log(element.textContent); // Output: "Welcome"

</script>

</body>

</html>

* **2. getElementsByClassName ()**
* Selects **all elements** with a given class name.
* Returns an **HTMLCollection** (a live collection of elements).
* Access individual elements using an index (e.g., collection [0]).
* **Syntax - document.getElementsByClassName("className");**
* **Example –**

<!DOCTYPE html>

<html>

<body>

<p class="text">First paragraph</p>

<p class="text">Second paragraph</p>

<script>

let elements = document.getElementsByClassName("text");

console.log(elements[0].textContent); // Output: "First paragraph"

console.log(elements[1].textContent); // Output: "Second paragraph"

</script>

</body>

</html>

* **3. querySelector ()**
* Selects the **first element** that matches a specified CSS selector.
* Returns null if no matching element is found.
* **Syntax - document.querySelector("selector");**
* **Example –**

<!DOCTYPE html>

<html>

<body>

<div class="container">

<p>First paragraph</p>

<p>Second paragraph</p>

</div>

<script>

let element = document.querySelector(".container p");

console.log(element.textContent); // Output: "First paragraph"

</script>

</body>

</html>

* **JavaScript Timing Events (setTimeout, setInterval)**
  1. **Explain the setTimeout () and setInterval () functions in JavaScript. How are they used for timing events?**
* Both setTimeout() and setInterval() are used to schedule events to occur after a certain delay or at regular intervals. These functions allow you to manage timing in JavaScript, enabling you to create delays or repeat tasks.
* **1. setTimeout()**
* The setTimeout() function is used to execute a **single** function or code block after a specified delay.
* Returns a **timer ID** that can be used to **cancel** the timeout using clearTimeout().
* Syntax - setTimeout(function, delay);
* Example –

setTimeout(function() {

alert("This message appears after 3 seconds!");

}, 3000); // 3000 milliseconds = 3 seconds

* **Cancelling a Timeout:**
* You can cancel a timeout using clearTimeout() with the timer ID returned by setTimeout().
* Example –

let timerId = setTimeout(function() {

console.log("This won't run!");

}, 5000);

clearTimeout(timerId); // Cancels the timeout before it executes

* **2. setInterval()**
* The setInterval() function is used to execute a **repeated** function or code block at **regular intervals**.
* It runs continuously at the specified interval (in milliseconds) until stopped.
* Returns a **timer ID** that can be used to **stop** the interval using clearInterval().
* Syntax - setInterval(function, interval);
* Example -

setInterval (function () {

console.log ("This message appears every 2 seconds.");

}, 2000); // 2000 milliseconds = 2 seconds

* 1. **Provide an example of how to use setTimeout ()to delay an action by 2 seconds.**
* When the button is clicked, the setTimeout() function is triggered.
* The alert message will appear after 2 seconds (2000 milliseconds).

<!DOCTYPE html>

<html>

<head>

<title>setTimeout Example</title>

</head>

<body>

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

<script>

document.getElementById("myButton").addEventListener("click", function() {

setTimeout(function() {

alert("This message appears after 2 seconds!");

}, 2000); // 2000 milliseconds = 2 seconds

});

</script>

</body>

</html>

* **JavaScript Error Handling**

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

* **1. Try Block**
* The try block is used to wrap code that might throw an error. If an error occurs inside the try block, it will be caught by the catch block (if present).
* Syntax –

try {

// Code that may throw an error

}

* **2. Catch Block**
* The catch block is used to handle errors that occur in the try block. It only executes if an error is thrown inside the try block. The error object is passed to catch to provide details about the error.
* Syntax –

catch (error) {

// Code to handle the error

}

* **3. Finally Block**
* The finally block is optional and runs **after** the try and catch blocks, regardless of whether an error occurred or not. It is useful for cleaning up resources, such as closing files or connections.
* Syntax –

finally {

// Code that runs after the try and catch blocks

}

* Example –

try {

let number = 10;

let result = number / 0; // This won't throw an error, but let's simulate one.

// Simulating an error

if (result === Infinity) {

throw new Error ("Division by zero is not allowed!");

}

console.log ("Result:", result);

} catch (error) {

console.error ("Error:", error.message); // Catches and logs the error

} finally {

console.log ("This will run no matter what, for clean up or final actions.");

}

Output - Error: Division by zero is not allowed!

This will run no matter what, for clean up or final actions.

1. **Why is error handling important in JavaScript applications?**

* Error handling is crucial in JavaScript applications for several reasons. It ensures the reliability, maintainability, and user-friendliness of applications.