**JAVASCRIPT Assignment**

**MODULE: 17 (JAVASCRIPT BASIC AND DOM)**

* **Javascript Introduction**

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

Ans. JavaScript is a versatile and widely-used programming language primarily known for its role in web development. It's a core technology for creating interactive and dynamic elements on websites. Here are some key points about JavaScript:

* It was developed and managed by ECMAScript organization.
* It is most popular, light-weight client-side scripting language.
* It executes faster on browser, compared to other languages.
* It provides many libraries like jQuery, ReactJS, Vue.js, Express JS and frameworks like Angular, Angular JS, Node JS.
* It provides flexibility, code reusability, functionality and events to perform various actions.
* It can run in both <body> and <head> section of HTML files.
* It provides various output methods to print the content and methods to run on browser.

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

Ans. Javascript is different from other programming languages in following ways:

1. **Primary Use:**

* JavaScript: Mainly for web development (frontend and backend with Node.js).
* Python: General-purpose; popular in data science, machine learning, and web development.
* Java: General-purpose; widely used in enterprise applications and Android development.

1. **Runtime Environment**:

* JavaScript: Runs in browsers and server environments (Node.js).
* Python and Java: Run on various platforms; commonly used in desktop, server-side, and CLI applications.

1. **Typing System**:

* JavaScript: Dynamically typed; types are determined at runtime.
* Python: Dynamically typed, with a focus on readability.
* Java: Statically typed; types must be declared explicitly.

1. **Object Orientation**:

* JavaScript: Prototype-based object orientation.
* Python and Java: Class-based object orientation.

1. **Frameworks and Ecosystem**:

* JavaScript: Web-focused frameworks (React, Angular, Vue) and backend (Express).
* Python: Strong in data science (NumPy, pandas), ML (TensorFlow), and web (Django, Flask).
* Java: Enterprise-scale frameworks (Spring, Hibernate).

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

Ans. The <script> tag in HTML is used to include JavaScript code within an HTML document. The following ways can be used to link external Javascript file to an HTML document:

1. **Embedding Inline Javascript:** Placing the Javascript code directly inside the <script> tag

For example:

<script>

alert("Hello, World!");

</script>

1. **Linking to an external Javascript file:** using the src attribute to link an external file, keeping HTML clean and reusable

For example:

<script src="path/to/your-script.js"></script>

1. **Positioning the <script> tag:**

* In <head>: Loads before page content
* Before </body>: Allows the page to load first
* **Variables and Data types**

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

Ans. In JavaScript, **variables** are containers used to store data values, allowing to use and manipulate data throughout your code. Variables can be declared using keywords var, let and const, although they work differently as follows:

1. **var declaration**

* declares a variable with function scope or global scope
* allows re-declaration and re-assignment of values
* for example:

var x = 10;

x = 20; // Allowed

1. **let declaration**

* declares a variable with block scope (only accessible within the block it’s defined in)
* Allows re-assignment but not re-declaration within the same scope
* for example:

let y = 15;

y = 25; // Allowed

1. **const declaration**

* Declares a constant variable with block scope
* Does not allow re-assignment or re-declaration
* for example:

const z = 30;

z = 40; // Error: Assignment to constant variable

1. Explain different datatypes in Javascript? Provide examples for each.

Ans. In JavaScript, data types categorize different types of values. Here’s a breakdown of JavaScript's data types with examples:

1. **String**

* Represents text, written within single (' '), double (" "), or backticks (` `) for template literals.
* for example:

let name = "John";

let greeting = `Hello, ${name}`;

1. **Number**

* Represents both integers and floating-point numbers.
* for example:

let age = 25;

let price = 99.99;

1. **Boolean**

* Represents a logical entity with only two values: true or false.
* for example:

let isActive = true;

let isComplete = false;

1. **Null**

* Represents an intentional absence of any object value.
* for example:

let emptyValue = null;

1. **Object**

* Represents a complex data structure that can store multiple key-value pairs.
* for example:

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

1. **Array** (a type of object)

* Represents an ordered list of values.
* for example:

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

1. **BigInt**

* Represents large integers beyond the safe limit for Number data type.
* for example:

let bigNum = 1234567890123456789012345678901234567890n;

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **Undefined** | **Null** |
| **Meaning** | Variable declared but not assigned a value | Intentional absence of any object or value |
| **Type** | Undefined (primitive type) | object |
| **Assigned by** | JavaScript automatically | Developer explicitly |
| **Usage** | Default state for unassigned variables | Used to represent "no value" or "empty" |
| **Example** | let x; // x is undefined | let y = null; // y is null |
| **Type check result** | Returns “undefined” | Returns “object” |

* **Javascript Operators**

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

arithematic, assignment, comparison, logical

Ans. In JavaScript, operators are special symbols that perform operations on variables and values. Here are the different types of operators, along with explanations and examples:

**Arithmetic Operators**

* **Purpose**: Perform mathematical calculations.
* **Examples**:
* Addition (+)

let sum = 5 + 3; // 8

* Substraction (-)

let difference = 5 - 3; // 2

* Multiplication (\*)

let product = 5 \* 3; // 15

* Division (/)

let quotient = 6 / 3; // 2

* Modulus (%): Returns remainder of a division

let remainder = 5 % 3; // 2

* Exponentiation (\*\*): Raises a number to the power of another

let power = 2 \*\* 3; // 8

**Assignment Operators**

* **Purpose**: Assign values to variables.
* **Examples**:
* Simple Assignment (=)

let x = 10; // x is assigned 10

* Addition Assignment (+=)

x += 5; // x is now 15

* Substraction Assignment (-=)

x -= 3; // x is now 12

* Multiplication Assignment (\*=)

x \*= 2; // x is now 24

* Division Assignment (/=)

x /= 4; // x is now 6

* Modulus Assignment (%=)

x %= 5; // x is now 1

**Comparison Operators**

* **Purpose**: Compare two values and return a Boolean (true or false).
* **Examples**:
* Equal to (==)

console.log(5 == '5'); // true

* Strict Equal to (===)

console.log(5 == '5'); // false

* Not Equal to (!=)

console.log(5 != '5'); // false

* Strict Not Equal to (!==)

console.log(5 !== '5'); // true

* Greater than (>)

console.log(5 > 3); // true

* Less than (<)

console.log(5 < 3); // false

* Greater than or Equal to (>=)

console.log(5 >= 5); // true

* Less than or Equal to (<=)

console.log(5 <= 4); // false

**Logical Operators**

* **Purpose**: Perform logical operations and return Boolean values.
* **Examples**:
* Logical AND (&&): Returns true if both operands are true.

console.log(true && false); // false

* Logical OR (||): Returns true if at least one operand is true.

console.log(true || false); // true

* Logical NOT (!): Inverts the Boolean value

console.log(!true); // false

1. What is the difference between == and === in Javascript?

Ans. In JavaScript, == (double equals) and === (triple equals) are comparison operators used to compare two values, but they behave differently in terms of type coercion.

* Loose Comparison (==)

checks whether the given values of the variable are same or not, irrespective of their datatypes

console.log(5 == `5`); // true (both are numbers)

* Strict comparison (===)

checks if both the data value and data types of the given variable is same or not

console.log(5 === '5'); // false (number Vs string)

console.log(`5` === '5'); // true (string Vs string)

* **Control Flow(if..else, switch)**

1. What is control flow in Javascript? Explain how if..else statement works with an example.

Ans. Control flow in JavaScript refers to the order in which statements or instructions are executed in a program. It determines the path that the program takes based on certain conditions. Control flow structures include conditional statements (like if..else), loops (for, while), and switches.

**If..Else Statement**

The if..else statement is a fundamental control structure that allows to execute different blocks of code based on the evaluation of a condition. It works in following manner:

* **If Block:** If the condition is true, the code inside the if block executes.
* **Else Block:** If the condition is false, the code inside the else block executes (if it exists).
* **Else If Block:** It is used to test multiple conditions.
* For example:

let score = 85;

if (score >= 90) {

console.log("Grade: A");

} else if (score >= 80) {

console.log("Grade: B");

} else if (score >= 70) {

console.log("Grade: C");

} else {

console.log("Grade: D");

}

**Explanation of the Example**

* The variable score is set to 85.
* The if statement checks if score is greater than or equal to 90. This condition is false, so it moves to the next condition.
* The else if statement checks if score is greater than or equal to 80. This condition is true, so the code inside this block executes, and it logs "Grade: B" to the console.
* The remaining else if and else blocks are skipped since one condition has already been satisfied.

1. Describe how switch statement works in Javascript. When should you use a switch statement instead of if…else?

Ans. The switch statement in JavaScript is a control flow structure that allows to execute different blocks of code based on the value of a specific expression. It is particularly useful for situations where there are multiple potential values for a single variable or expression. It works in the following way:

* **Expression Evaluation:** The switch statement evaluates an expression and compares its value against the values specified in the case clauses.
* **Case Clauses:** Each case defines a value to compare with the expression. If there is a match, the corresponding block of code executes.
* **Break Statement:** The break statement is used to exit the switch block. If omitted, execution continues to the next case (this is known as "fall-through").
* **Default Clause:** The default clause can be used to specify a block of code that runs if none of the case values match the expression.
* For example:

let fruit = "banana";

switch (fruit) {

case "apple":

console.log("You chose an apple.");

break;

case "banana":

console.log("You chose a banana.");

break;

case "orange":

console.log("You chose an orange.");

break;

default:

console.log("Fruit not found.");

}

**Explanation of the Example**

* The variable fruit is set to "banana".
* The switch statement evaluates fruit and compares it with the values in each case.
* When it reaches the case "banana":, it finds a match and executes the code inside that block, logging "You chose a banana." to the console.
* The break statement prevents further execution of subsequent case blocks.
* If fruit had not matched any case, the default block would execute.

**When to Use a Switch Statement Instead of If…Else**

* **Multiple Conditions:** Use switch when there are multiple specific values to compare against a single variable or expression, as it can be cleaner and more readable than using multiple if..else if statements.
* **Better Readability:** When dealing with many cases, switch can improve readability by neatly organizing the conditions.
* **Exact Matches:** When need arises to check for exact matches (like strings or numbers), switch is a good choice.
* **Loops (for, while, do..while)**

1. Explain different types of loops in Javascript. Provide a basic example for each.

Ans. In JavaScript, there are several looping structures that allows to execute a block of code repeatedly. Here are the main types of loops:

1. **For loop**

The for loop is used when the number of iterations is known. It consists of three parts: initialization, condition, and increment/decrement.

*eg.* for (let i = 0; i < 5; i++) {

console.log(i); // Output: 0, 1, 2, 3, 4

}

1. **While loop**

The while loop continues to execute as long as a specified condition is true. It checks the condition before each iteration.

*eg.* let i = 0;

while (i < 5) {

console.log(i); // Output: 0, 1, 2, 3, 4

i++; }

1. **Do…while loop**

The do...while loop is similar to the while loop, but it guarantees that the block of code is executed at least once, as the condition is checked after the execution.

*eg.* let i = 0;

do {

console.log(i); // Outputs: 0, 1, 2, 3, 4

i++;

} while (i < 5);

1. **For...of loop**

The for...of loop is used to iterate over iterable objects (like arrays, strings, and more). It provides a simpler way to loop through elements.

*eg.* const array = [1, 2, 3, 4, 5];

for (const value of array) {

console.log(value); // Outputs: 1, 2, 3, 4, 5

}

1. **For…in loop**

The for...in loop is used to iterate over the properties of an object. It is not recommended for arrays because it can lead to unexpected results.

*eg.* const obj = { a: 1, b: 2, c: 3 };

for (const key in obj) {

console.log(key, obj[key]); // Outputs: a 1, b 2, c 3

}

1. **Array.prototype.foreach ()**

While not a traditional loop, the forEach() method is an array method that executes a provided function once for each array element.

*eg.* const array = [1, 2, 3, 4, 5];

array.forEach(value => {

console.log(value); // Outputs: 1, 2, 3, 4, 5

});

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

Ans.

|  |  |  |
| --- | --- | --- |
| **Feature** | **While loop** | **Do..while loop** |
| **Condition evaluation** | |  | | --- | |  |   Evaluated **before** loop execution | |  | | --- | |  |   Evaluated **after** loop execution |
| **Execution Guarantee** | May not execute at all if condition is false | Executes **at least once** |
| **Syntax** | while (condition) { ... } | do { ... } while (condition); |
| **Use Case** | Used when the number of iterations is not known beforehand and you want to check the condition first | Ideal when the loop must run at least once (e.g., for user input validation) |
| **Example** | let count = 0;  while (count < 3) {  console.log("Count is:", count);  count++;} | let number = 0;  do {  console.log("Number is:", number);  number++;  } while (number < 3); |
| **Output** | Count is: 0 Count is: 1 Count is: 2 | Number is: 0 Number is: 1 Number is: 2 |

* **Functions**

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

Ans. Functions in JavaScript are reusable blocks of code designed to perform a specific task. They allow you to encapsulate code, making it modular, easier to read, and reusable across your application. Functions can take inputs (known as parameters) and can return outputs (values).

* Function Declaration:

function add(a, b) {

return a + b;

}

* Calling a Function

const sum = add(5, 3); // Calling the function with arguments 5 and 3

console.log(sum); // Output: 8

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

Ans. Below is the differences between function declaration and function expression:

**Function Declarations:**

* **Syntax:** Defined using function functionName(parameters) { ... }.
* **Hoisting:** Hoisted to the top of their containing scope, allowing them to be called before their definition.
* **Usage:** Can be invoked anywhere in the scope (before or after the declaration).
* **Named Functions:** Must always be named.
* **Context Binding:** Has its own this context based on how it is called.

**Function Expressions:**

* **Syntax:** Defined using const functionName = function(parameters) { ... };.
* **Hoisting:** Not hoisted like declarations; can only be called after they are defined.
* **Usage:** Can only be invoked after the expression has been evaluated.
* **Anonymous or Named:** Can be anonymous (without a name) or named.
* **Context Binding:** this context depends on how the function is called, which can lead to different behavior.

1. Discuss the concept of parameters and return values in function.

Ans. Parameters and return values are fundamental concepts in functions that enhance their functionality and flexibility.

**Parameters**

* **Definition:** Parameters are variables listed in a function's definition that act as placeholders for the values (arguments) that will be passed to the function when it is called.
* **Purpose:** They allow functions to accept input, enabling them to operate on different data and perform various tasks based on the provided arguments.
* **Multiple Parameters:** A function can have multiple parameters, separated by commas. Each parameter can have a default value if needed.
* For example:

function multiply(a, b) {

return a \* b; // 'a' and 'b' are parameters

}

**Return Values**

* **Definition:** A return value is the value that a function produces when it is executed. It is sent back to the part of the program that called the function.
* **Purpose:** Return values allow functions to output data, enabling further processing or decision-making based on the result.
* **Return Statement:** The return statement is used to specify the value to be returned. Once a function reaches a return statement, it stops executing and sends the specified value back.
* For example:

function add(a, b) {

return a + b; // Returns the sum of 'a' and 'b'

}

const sum = add(4, 6); // sum will be assigned the value 10

console.log(sum); // Output: 10

* **Arrays**

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

Ans. An array in JavaScript is a special type of object that allows you to store a collection of values in a single variable. Arrays can hold multiple items, including strings, numbers, objects, and even other arrays, making them versatile for organizing and managing data.

**Declaring and Initializing an Array**

There are two common ways to declare and initialize an array in JavaScript:

1. **Using Array Literal Syntax:**

* using square brackets [] and list the elements inside.

// Declaration and initialization of an array with numbers

const numbers = [1, 2, 3, 4, 5];

// Declaration and initialization of an array with mixed data types

const mixedArray = [1, "two", true, null, { key: "value" }];

1. **Using the Array Constructor:**

* using the Array constructor.

// Creating an array using the Array constructor

const colors = new Array("red", "green", "blue");

// Creating an empty array

const emptyArray = new Array();

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

Ans The methods push(), pop(), shift(), and unshift() are commonly used array manipulation methods in JavaScript. Each of these methods allows you to modify the contents of an array in different ways. Here’s a detailed explanation of each method:

1. **push()**

* **Purpose:** Adds one or more elements to the end of an array.
* **Return Value:** Returns the new length of the array after the elements have been added.
* **For example:**

const fruits = ["apple", "banana"];

const newLength = fruits.push("cherry", "date");

console.log(fruits); // Output: ["apple", "banana", "cherry", "date"]

console.log(newLength); // Output: 4

1. **pop()**

* **Purpose:** Removes the last element from an array.
* **Return Value:** Returns the removed element. If the array is empty, it returns undefined.
* **For example:**

const fruits = ["apple", "banana", "cherry"];

const lastFruit = fruits.pop();

console.log(fruits); // Output: ["apple", "banana"]

console.log(lastFruit); // Output: "cherry"

1. **shift()**

* **Purpose:** Removes the first element from an array.
* **Return Value:** Returns the removed element. If the array is empty, it returns undefined.
* **For example:**

const fruits = ["apple", "banana", "cherry"];

const firstFruit = fruits.shift();

console.log(fruits); // Output: ["banana", "cherry"]

console.log(firstFruit); // Output: "apple"

1. **unshift()**

* **Purpose:** Adds one or more elements to the beginning of an array.
* **Return Value:** Returns the new length of the array after the elements have been added.
* **For example:**

const fruits = ["banana", "cherry"];

const newLength = fruits.unshift("apple");

console.log(fruits); // Output: ["apple", "banana", "cherry"]

console.log(newLength); // Output: 3

* **Objects**

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

Ans. An object in JavaScript is a complex data type that allows you to store collections of key-value pairs. Objects are fundamental to JavaScript and are widely used to represent real-world entities and their attributes or behaviors.

They can be declared and initialized either by using object literal syntax or the new Object() constructor.

* Using object syntax:

const person = {

name: "Alice",

age: 30,

isStudent: false

};

* Using object constructor

const person = new Object();

person.name = "Alice";

person.age = 30;

person.isStudent = false;

* Differences between Objects and Arrays

|  |  |  |
| --- | --- | --- |
| **Feature** | **Objects** | **Arrays** |
| **Structure** | Consist of key-value pairs. | Consist of ordered lists of elements. |
| **Key/Index type** | Keys are strings (or Symbols). | Indices are numbers (starting from 0). |
| **Purpose** | Used to represent complex entities with properties and methods. | Used to store ordered collections of data. |
| **Accessing elements** | Accessed using keys (e.g., object.key). | Accessed using indices (e.g., array[0]). |
| **Iteration** | Typically iterated using for...in or Object.keys(). | Typically iterated using loops (e.g., for or forEach). |
| **Order** | Properties are unordered (not guaranteed). | Elements are ordered and maintain a sequence. |

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

Ans. In JavaScript, you can access and update object properties using two primary notations: dot notation and bracket notation. Each has specific use cases and syntax.

1. **Dot Notation**

* **Syntax:** object.property
* **Usage:** Dot notation is straightforward and commonly used when the property name is known and follows JavaScript's naming rules (no spaces, hyphens, or numbers at the start).
* For example:

const person = { name: "Alice", age: 25 };

// Accessing a property

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

// Updating a property

person.age = 30;

console.log(person.age); // Output: 30

1. **Bracket Notation**

* **Syntax:** object["property"]
* **Usage:** Bracket notation is more flexible, allowing access to properties with names that include spaces, special characters, or are stored in a variable.
* For example:

const person = { "first name": "Alice", age: 25 };

// Accessing a property with special characters or spaces

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

// Updating a property using a variable

const prop = "age";

person[prop] = 30;

console.log(person.age); // Output: 30

* **Javascript Events**

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

Ans. JavaScript events are actions or occurrences that happen within the browser, such as a user clicking a button, pressing a key, scrolling a page, or loading a document. These events are central to interactive web pages, allowing JavaScript to respond dynamically to user interactions or browser actions.

**Key Concepts of JavaScript Events**

* **Event Types:** Common events include:
  + **Click events:** When a user clicks on an element (click).
  + **Keyboard events:** When a key is pressed or released (keydown, keyup).
  + **Mouse events:** When a mouse enters, leaves, or moves over an element (mouseover, mouseout, mousemove).
  + **Form events:** When a form or its elements are focused, submitted, or changed (focus, submit, change).
  + **Window events:** When the page loads or resizes (load, resize).

**Event Listeners**

Event listeners play a critical role in handling these events by "listening" for specified events on selected elements and executing a function when the event occurs.

* **Purpose:** Event listeners allow JavaScript code to respond to user actions and other events by running specific code (often called an event handler function).
* **Syntax:** element.addEventListener(event, handlerFunction), where:
  + event is the type of event to listen for (e.g., "click", "keydown").
  + handlerFunction is the function that will execute when the event occurs.
* For example:

const button = document.querySelector("#myButton");

// Add an event listener to the button for a "click" event

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

console.log("Button was clicked!");

});

In this example, an event listener is added to a button element with the ID myButton. When the button is clicked, the specified function runs, and "Button was clicked!" is logged to the console.

1. How does addEventListener() method work in Javascript? Provide an example.

Ans. The addEventListener() method in JavaScript is used to attach an event listener to a DOM element, allowing JavaScript code to respond to various events (such as clicks, key presses, mouse movements, etc.). This method is essential for making web pages interactive, as it lets you specify an event to listen for and a function to execute when that event occurs.

**How addEventListener() Works**

* **Syntax:** element.addEventListener(event, listener, options)
  + **event:** The type of event to listen for (e.g., "click", "keydown", "mouseover").
  + **listener:** The function that will be called when the event occurs (also known as the event handler).
  + **options:** (Optional) An object that specifies additional options, such as:
    - **once:** If true, the listener will be executed at most once and then automatically removed.
    - **capture:** If true, sets the event to be captured during the capturing phase.
    - **passive:** If true, indicates that the event listener will not call preventDefault(), improving performance for scroll events.
* For example:

// Select the button element

const button = document.querySelector("#myButton");

// Define the event handler function

function handleClick() {

console.log("Button was clicked!");

}

// Attach the event listener to the button for the "click" event

button.addEventListener("click", handleClick);

In above example:

* The button variable selects a DOM element with the ID myButton.
* The handleClick function is defined to log a message to the console.
* addEventListener() attaches handleClick to the button for the "click" event, so each time the button is clicked, the message "Button was clicked!" appears in the console.
* **DOM Manipulation**

1. What is DOM(Document Object Model) in Javascript? How does Javascript interact with the DOM?
2. Explain the methods getElementById(), getElementByClassName() and querySelector() used to select elements from the DOM.

* **Javascript Timing Events (setTimeout, setInterval)**

1. Explain setTimeout() and setInterval() functions in Javascript. How are they used for timing events?
2. Provide an example of how to setTimeout() to dealy an action by 2 seconds.

* **Javascript Error Handling**

1. What is Error handling in Javascript? Explain the try, catch and finally blocks with an example.
2. Why is error handling important in Javascript applications?