Module 8 – JavaScript

Theory Assignment

Q1. What is JavaScript? Explain the role of JavaScript in web development.

Ans : Javascript is scripting language - work as client side/browser based.

JavaScript is a programming language that plays a key role in web development by adding interactivity and dynamic features to web pages.

Q2. How is JavaScript different from other programming languages like Python or Java?

Ans : JavaScript is unique in several ways compared to languages like Python and Java, owing to its primary purpose, features, and runtime environment.

Q3. Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?

Ans : The <script> tag in HTML is used to embed or reference JavaScript code in an HTML document. This enables you to add interactivity, dynamic content, and logic to your web pages.

The <script> tag can be used in two main ways:

1. Inline JavaScript

   - JavaScript code is written directly within the <script> tag.
   - Ex :

     ```
     <script>
       alert('Hello, World!');
     </script>
     ```

2. External JavaScript

   - An external file containing JavaScript code is linked to the HTML document using the src attribute of the <script> tag.
   - Ex :

     ```
     <script src="script.js"></script>
     ```

Q4. What are variables in JavaScript? How do you declare a variable using var, let, and const?

<u>Ans</u> : In JavaScript, **variables** are containers used to store data values. They allow developers to label and reuse data throughout the program.

JavaScript provides three keywords for declaring variables: var, let, and const. Each has different characteristics and use cases.

1. **Var**

   - We can possible Redeclare & reassign

- Ex :

```
var x = 10; // Declare and assign
 x = 20;     // Re-assign value
```

2. const

- We can’t possible Redeclare & reassign
- Ex :

```
const z = 30; // Declare and assign
// z = 40;    // Error: Assignment to constant variable
```

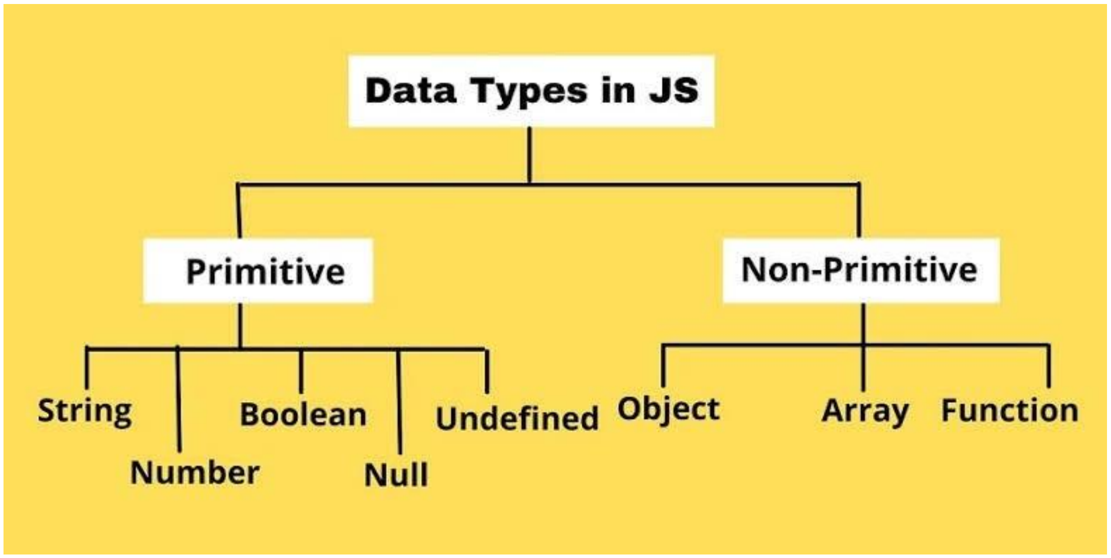
3. let

- We can’t redeclare but reassign possible
- Ex :

```
 let y = 15; // Declare and assign
y = 25;     // Re-assign value
```

Q5. Explain the different data types in JavaScript. Provide examples for each.

Ans :



- **Primitive Data Types**

1. Number

➢ Represents both integers and floating-point numbers

➢ Example:

```
let age = 25;
let temperature = 98.6;
```

2. String

➢ Represents textual data enclosed in quotes: single ('), double ("), or template literals (`).

- Example:

```
let greeting = "Hello, world!";
let name = `John`;
```

3. Boolean

- Represents logical values: true or false.
- Example:

```
let isLoggedIn = true;
```

4. Undefined

- Represents a variable that has been declared but not assigned a value.
- Example:

```
let x;
console.log(x); // undefined
```

5. Null

- Represents the intentional absence of any value.
- Example:

```
let result = null;
```

- **Non-Primitive Data Types**

1. Object

- Represents collections of key-value pairs.
- Example:

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

2. Array

- A special type of object used for storing ordered collections
- Example:

```
let numbers = [1, 2, 3, 4];
```

3. Function

- JavaScript treats functions as objects that can be passed as arguments or returned by other functions.
- Example:

```
function greet() {
  return "Hello!";
}
```

Q6. What is the difference between undefined and null in JavaScript?

Ans : undefined means a variable has been declared but has not yet been assigned a value, whereas null is an assignment value, meaning that a variable has been declared and given the value of null .

Ex :

```
 let x;
console.log(x); // undefined
let result = null; // Null
```

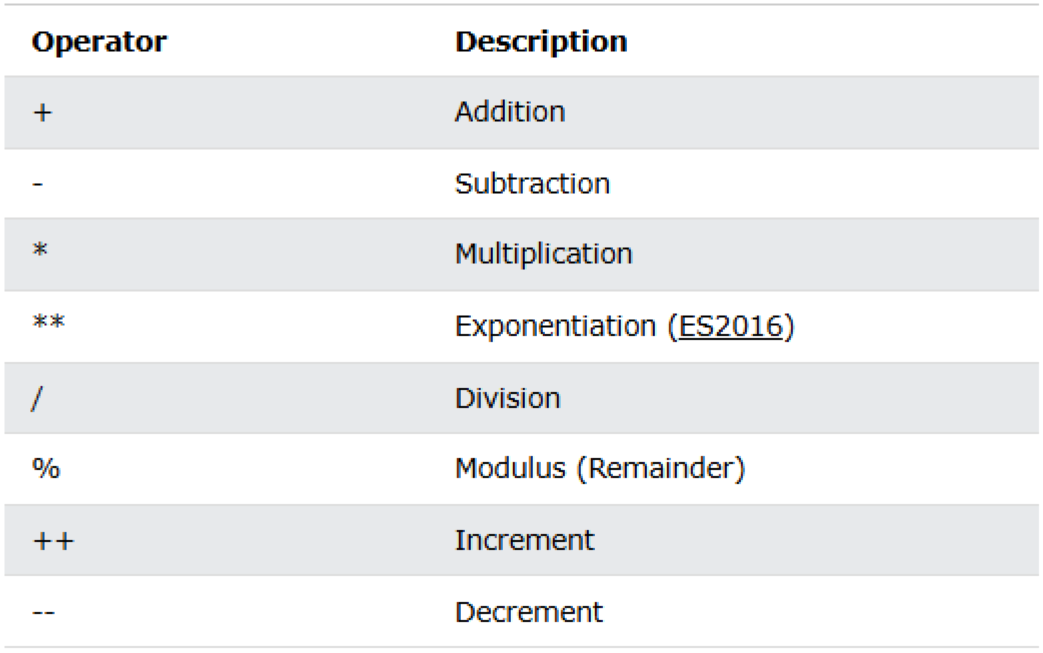
Q7. What are the different types of operators in JavaScript? Explain with examples.

- **Arithmetic operators**
- **Assignment operators**
- **Comparison operators**
- **Logical operators**

Ans : Javascript operators are used to perform different types of mathematical and logical computations.

1. **Arithmetic Operators :**

Used for performing mathematical calculations.



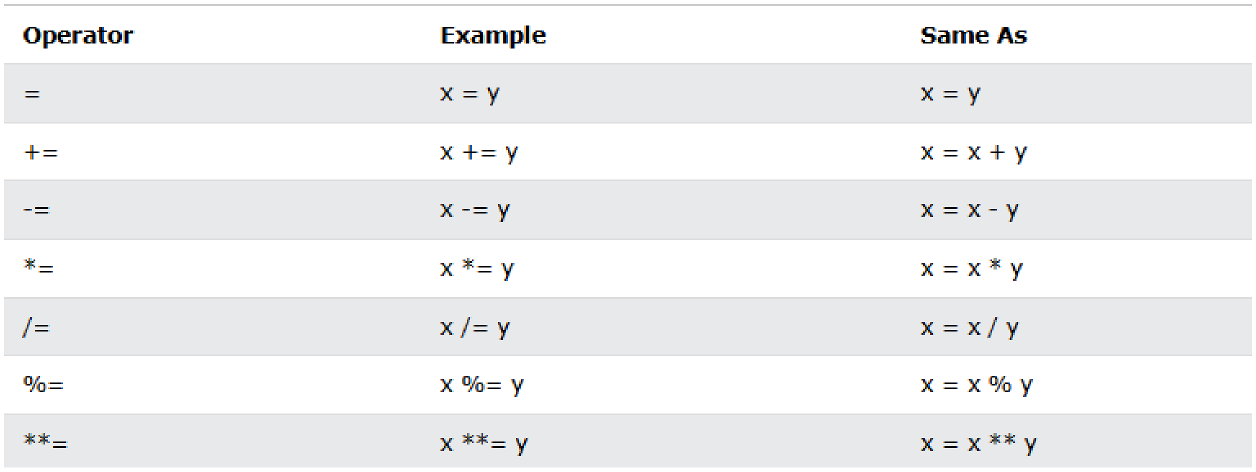
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Ex :

```
let x = 10, y = 3;
console.log(x + y); // 13
console.log(x % y); // 1
```

2. Assignment operators :

Used to assign values to variables.



Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Ex :

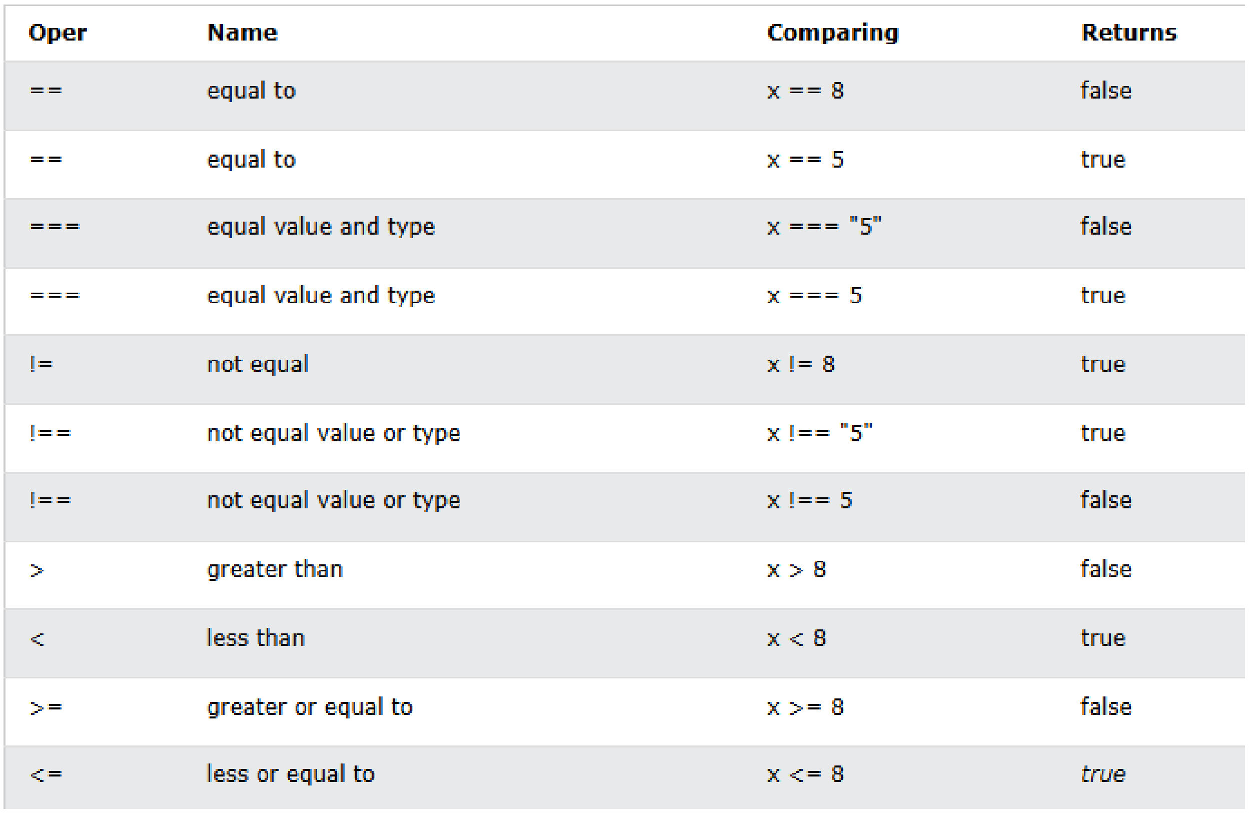
```
let a = 10;
a += 5; // a = 15
```

3. Comparison operators :

Used to compare two values, returning true or false.

Ex :

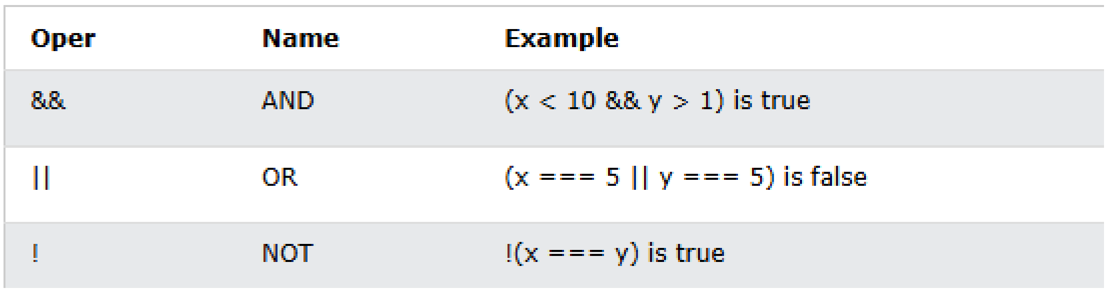
```
console.log(10 > 5); // true
console.log(10 === '10'); // false
```



Oper	Name	Comparing	Returns
==	equal to	x == 8	false
==	equal to	x == 5	true
===	equal value and type	x === "5"	false
===	equal value and type	x === 5	true
!=	not equal	x != 8	true
!==	not equal value or type	x !== "5"	true
!==	not equal value or type	x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater or equal to	x >= 8	false
<=	less or equal to	x <= 8	*true*

4. Logical operators :

Used for logical operations, often with Boolean values.



Oper	Name	Example
&&	AND	(x < 10 && y > 1) is true
||	OR	(x === 5 || y === 5) is false
!	NOT	!(x === y) is true

Ex:

```
 let a = true, b = false;
console.log(a && b); // false
console.log(a || b); // true
console.log(!a); // false
```

Q8. What is the difference between == and === in JavaScript?

Ans :

- double equals (==) operator :- It can be check only value.

  Ex:

  ```
   let a=35 //number
  let b="35" //string
  //return true
  ```

- triple equals (===) operator :- it can be check value and datatype both.

  Ex:

  ```
   let a=35 //number
  let b="35" //string
  //return false
  ```

Q9. What is control flow in JavaScript? Explain how if-else statements work with an example.

Ans : Control flow is the order in which statements are executed in a program. The default control flow is for statements to be read and executed in order from left-to-right, top-to-bottom in a program file.

if-else Statements:

- if condition is true it will be execute if block code otherwise it can be execute else block code.

Syntax :

```
 if (condition {
     // Executes this block if
      // condition is true
} else {
      // Executes this block if
     // condition is false
}
```

Ex :

```
 let age = 18;
if (age >= 18) {
 console.log("You are eligible to vote.");
} else {
  console.log("You are not eligible to vote.");
}
```

Output :

```
  You are eligible to vote.
```

Q10. Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

Ans : The JavaScript switch statement evaluates an expression and executes a block of code based on matching cases. It provides an alternative to long if-else chains, improving readability and maintainability, especially when handling multiple conditional branches.

Syntax :

```
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
```

```
                    // code block
                }
```

This is how it works :

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Ex :

```
    let trafficLight = "green";
let message = ""


switch (trafficLight) {
  case "red":
    message = "Stop immediately.";
    break;
  case "yellow":
    message = "Prepare to stop.";
    break;
  case "green":
    message = "Proceed or continue driving.";
    break;
```

```
   default:
       message = "Invalid traffic light color.";
}

console.log(message)

// Output: Proceed or continue driving.
```

Q11. Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.

Ans :-

For loop :- For loop is used to for reapeating task for fixed number of iterations.

Syntax:-

```
for (initialization; condition; increment/decrement) {
// Code to execute
}
```

Ex:-

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

While loop :- It can be used to reapeating task as long as condition Holds True.

Syntax :-

```
        while (condition) {
        // Code to execute
        }
```

Ex:-

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

do-while loop :- A do...while loop in JavaScript is similar to a while loop, but it can perform task at least one time and then check condition and print holds true.

Syntax :-

```
       do {
          // Code to execute
       } while (condition);
```

Ex:-

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

Q12. What is the difference between a while loop and a do-while loop?

<u>Ans</u> :-

While Loop	**do...while Loop**
The condition is checked before executing the loop body.	The condition is checked after executing the loop body.
The loop body may never execute if the condition is initially false.	The loop body will always execute at least once, even if the condition is false.
Used when you want to repeat a block of code only if the condition is true from the beginning.	Used when the block of code needs to execute at least once before checking the condition.

Q13. What are functions in JavaScript? Explain the syntax for declaring and calling a function.

<u>Ans</u> :- In JavaScript, a function is a reusable block of code that performs a specific task.

Syntax :-

```
    function functionName(parameters) {
        // Code to execute
        return value; // (Optional) Return a value
    }
```

Ex :-

```
    function sum(x, y) {
        return x + y;
    }
    console.log(sum(6, 9));
   // Output :- 15
```

Q14. What is the difference between a function declaration and a function expression?

<u>Ans</u> :-

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

Q15. Discuss the concept of parameters and return values in functions.

Ans :- A return value is a result of the function's execution. It can be returned to the block of code that called the function and then used as needed. Parameters are the input for a function that are necessary for the it to be executed and produce a result.

Q16. What is an array in JavaScript? How do you declare and initialize an array?

Ans :- Array is a collection of data.

Declaring and Initializing an Array :

```
  // Declaring and initializing an array
 let fruits = ['apple', 'banana', 'cherry'];

 // Accessing elements
console.log(fruits[0]); // Output: 'apple'
console.log(fruits[2]); // Output: 'cherry'
```

Q17. Explain the methods push(), pop(), shift(), and unshift() used in arrays.

<u>**Ans**</u> :-

1. **push() :**

    - Adds one or more elements to the **end** of the array.

    Ex:

    ```
    let fruits = ['apple', 'banana'];
    fruits.push('cherry'); // Adds 'cherry' to the end
    console.log(fruits); // Output: ['apple', 'banana', 'cherry']
    ```

2. **pop() :**

- Removes the **last** element from the array.

Ex:

```
let fruits = ['apple', 'banana', 'cherry'];
let removed = fruits.pop(); // Removes 'cherry'
console.log(fruits); // Output: ['apple', 'banana']
```

3. shift() :

- Removes the **first** element from the array.

Ex:

```
let fruits = ['apple', 'banana', 'cherry'];
let removed = fruits.shift(); // Removes 'apple'
console.log(fruits); // Output: ['banana', 'cherry']
```

4. unshift() :

- Adds one or more elements to the **beginning** of the array.

Ex :

```
let fruits = ['banana', 'cherry'];
fruits.unshift('apple'); // Adds 'apple' to the beginning
console.log(fruits); // Output: ['apple', 'banana', 'cherry']
```

Q18. What is an object in JavaScript? How are objects different from arrays?

<u>Ans</u> :- An object in JavaScript is a data structure used to store related data collections. It stores data as key-value pairs.

- Arrays are used when we need to collection a list of elements of the same data type or structure. On the other hand, objects are used when grouping multiple sets of data that belong together using labels, where each property or key has its own value of any type.

Q19. Explain how to access and update object properties using dot notation and bracket notation.

<u>Ans</u> :-

<u>Accessing Object Properties :-</u>

Dot Notation :

- Syntax: object.property
- Used when the property name is a valid identifier (e.g., no spaces, special characters, or numbers at the start).

Ex :-

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

```
console.log(person.name); // Output: "Alice"
console.log(person.age);  // Output: 30
```

Bracket Notation :

- Syntax: object["property"]
- Used when :
    - The property name is dynamic (e.g., stored in a variable).
    - The property name contains special characters, spaces, or starts with a number.

Ex :-

```
const person = {
"full name": "Alice Smith",
 age: 30
 };

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

 // Dynamic property access
const prop = "age";
console.log(person[prop]); // Output: 30
```

Updating Object Properties :-

Dot Notation :

- Assign a new value using the = operator.

Ex :

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

person.name = "Bob"; // Updating property
console.log(person.name); // Output: "Bob"
```

Bracket Notation :

- Assign a new value similarly, but with the property name in quotes or a variable.

Ex :

```
 const person = {
 "full name": "Alice Smith",
  age: 30
 };

 person["full name"] = "Bob Brown"; // Updating property
console.log(person["full name"]); // Output: "Bob Brown"
```

```
 // Using a dynamic property
const prop = "age";
person[prop] = 35; // Updating property
console.log(person.age); // Output: 35
```

Q20. What are JavaScript events? Explain the role of event listeners.

<u>Ans</u> :- A JavaScript event is a specific action that occurs within a web page or application, such as clicking on an element, moving the mouse, pressing a key, or loading a page.

JavaScript's event listener function allows you to create custom responses to events like mouse clicks, keyboard clicks, and window resizing. The programming paradigm of waiting and responding to real-time events is called event handling.

Q20. How does the addEventListener() method work in JavaScript? Provide an example.

<u>Ans</u> :- addEventListener is a built-in function in JavaScript that allows you to attach an event handler to a specified element, such as a button or a link. The addEventListener function takes two arguments: the type of the event you want to listen for (e.g. "click" or "keydown") and the function that should be called when the event is detected.

Ex :-

```
let button = document.querySelector('#my-button');
button.addEventListener('click', function() {
console.log('Button was clicked');
});
```

Q21. What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

Ans :- The Document Object Model (DOM) is programming interface that allows javascript to interact with and minipulate HTML and XML document.

JavaScript interacts with the DOM (Document Object Model) by selecting HTML elements using methods like getElementById, getElementsByClassName, or querySelector, and then manipulating their properties or contents using methods like innerHTML, textContent, or setAttribute

Ex :-

```
document.getElementById("demo").innerHTML = "Hello World!";
Or
const element = document.getElementById("intro");
```

Q22. Explain the methods getElementById(), getElementsByClassName(),and querySelector() used to select elements from the DOM.

Ans :-

1. **getElementById() :**

- Selects a single element with a specific id attribute.
- Returns the first element (unique) with the matching id.
- Best for selecting a single, unique element.

  Syntax :

```
const element = document.getElementById('idName');
```

2. **getElementsByClassName() :**

- Selects all elements with a specific class attribute.
- Returns a live HTMLCollection (similar to an array) of matching elements.
- Best for selecting multiple elements with the same class.

  Syntax :

```
const elements = document.getElementsByClassName('className');
```

3. **querySelector() :**

- Selects the first element that matches a CSS selector (e.g., #id, .class, or element type).
- Returns a single element or null if no match is found.
- More flexible than getElementById() and getElementsByClassName() because it supports CSS-style selectors.

  Syntax :

  ```
  const element = document.querySelector('selector');
  ```

Q23. Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?

Ans :- These are two important JavaScript functions used to execute code after a delay or repeatedly at specified intervals. They are part of the Web APIs provided by the browser.

1. **setTimeout() :**

   - The setTimeout() function allows you to execute a piece of code once after a specified delay in milliseconds

   Syntax :

   ```
   const timeoutId = setTimeout(function, delay, arg1, arg2, ...);
   ```

   - function: The function to execute.

- delay: The time in milliseconds to wait before executing the function.
- arg1, arg2, ...: Optional arguments to pass to the function.

Ex : Display a message After 3 seconds

```
setTimeout(() => {
console.log("Hello after 3 seconds!");
 }, 3000);
```

- Stopping a Timeout with clearTimeout():

If you need to cancel a timeout before it executes, use clearTimeout()

Ex :-

```
console.log("Start");

const timeoutId = setTimeout(() => {
console.log("Executed after 2 seconds");
}, 2000);

// Cancelling the timeout
clearTimeout(timeoutId); // The scheduled code won't execute.
```

2. setInterval() :

- The setInterval() function allows you to execute a piece of code repeatedly at a specified time interval in milliseconds.

Syntax :-

```
const intervalId = setInterval(function, delay, arg1, arg2, ...);
```

- function: The function to execute repeatedly.
- delay: The time interval in milliseconds between successive executions.
- arg1, arg2, ...: Optional arguments to pass to the function.

Ex :- Print a Message Every 5 Seconds

```
setInterval(() => {
console.log("Checking for updates...");
}, 5000); // Every 5 seconds
```

- Stopping an Interval with clearInterval():
- To stop an interval, use clearInterval().

Q24. Provide an example of how to use setTimeout() to delay an action by 2 seconds.

<u>Ans</u> :-

```
 console.log("Action will be delayed...");

 setTimeout(() => {
 console.log("This message is displayed after 2 seconds!");
 }, 2000);

console.log("Waiting...");
```

Q25. What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.

Ans :- Error handling in JavaScript is a process that helps manage unexpected conditions and errors that occur during runtime. It's an important part of writing robust, reliable, and user-friendly applications.

- The try statement defines the code block to run (to try).
- The catch statement defines a code block to handle any error.
- The finally statement defines a code block to run regardless of the result.

Syntax :

```
 try {
    tryCode - Code block to run
 }
```

catch(err) {

catchCode - Code block to handle errors

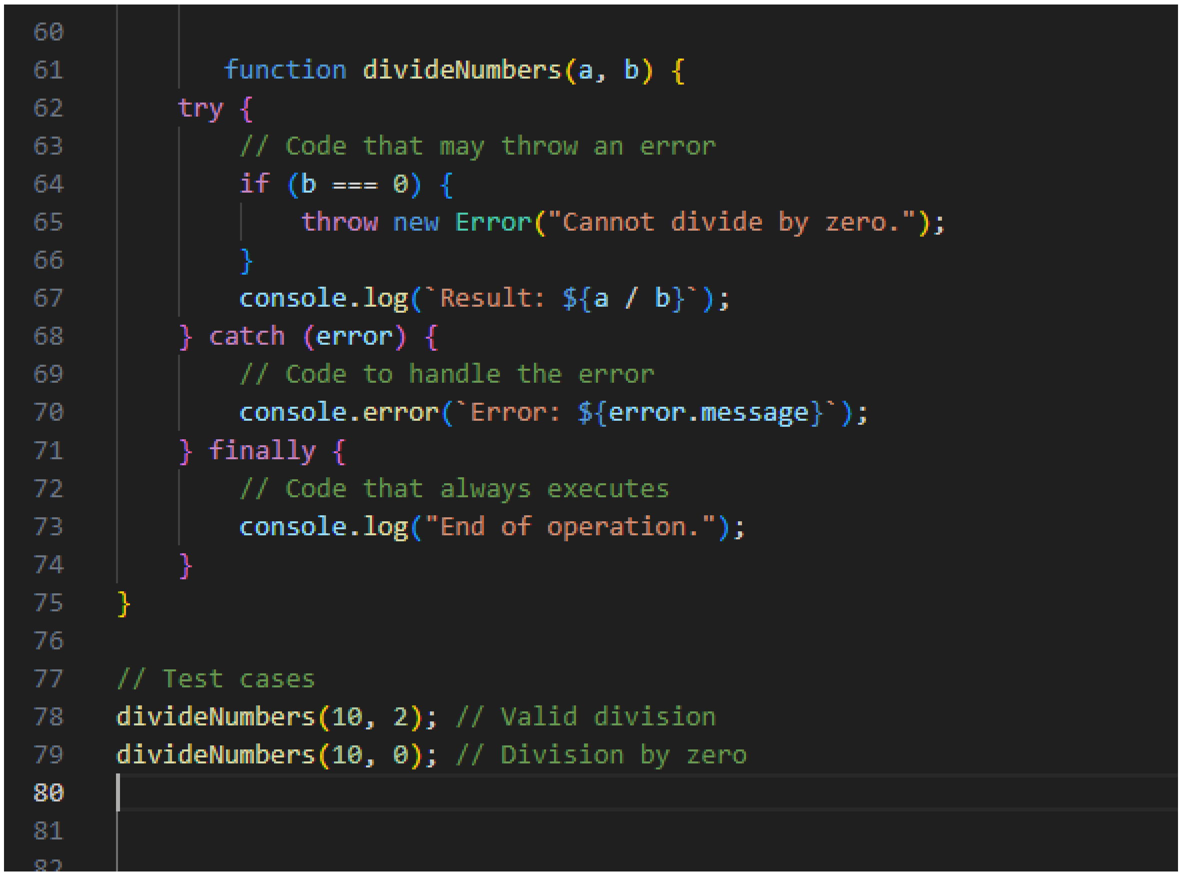
}

finally {

finallyCode - Code block to be executed regardless of the try result

}

Ex :

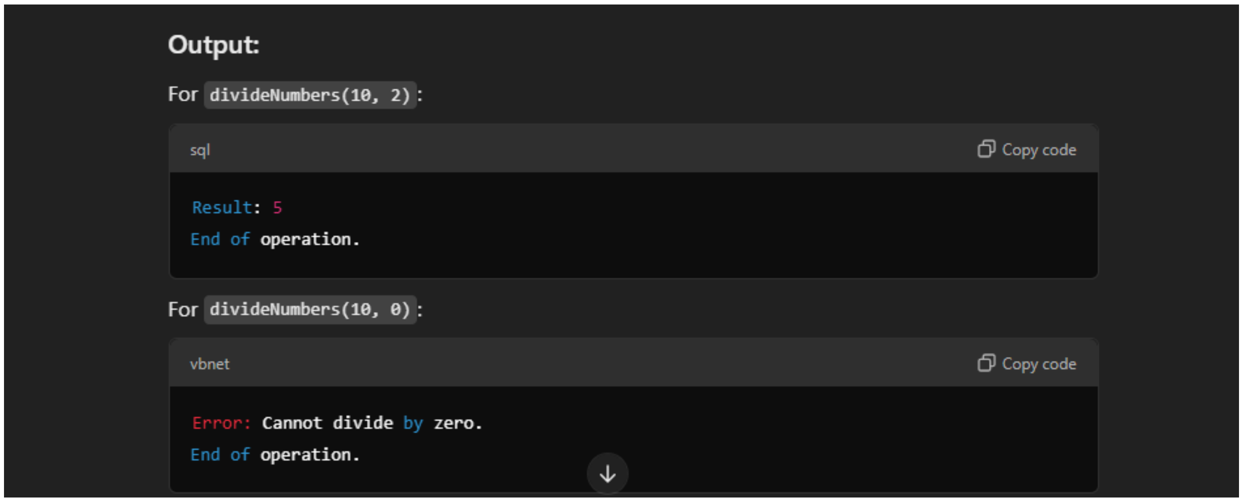


```javascript

    function divideNumbers(a, b) {
  try {
    // Code that may throw an error
    if (b === 0) {
      throw new Error("Cannot divide by zero.");
    }
    console.log(`Result: ${a / b}`);
  } catch (error) {
    // Code to handle the error
    console.error(`Error: ${error.message}`);
  } finally {
    // Code that always executes
    console.log("End of operation.");
  }
}

// Test cases
divideNumbers(10, 2); // Valid division
divideNumbers(10, 0); // Division by zero
```

Output :



Output:

For `divideNumbers(10, 2)`:

sql Copy code
```
Result: 5
End of operation.
```

For `divideNumbers(10, 0)`:

vbnet Copy code
```
Error: Cannot divide by zero.
End of operation.
```

Q26. Why is error handling important in JavaScript applications?

Ans :-

- It makes your app reliable.
- It prevents crashes.
- It helps developers fix and improve the app.
- It makes users happy because they don't see confusing errors.4
- Maintain a smooth user experience.
- Aid in debugging and monitoring.

.