TABLE OF CONTENT

MODULE 1
INTRODUCTION TO JAVASCRIPT

1.1 WHAT IS JAVASCRIPT?

JavaScript *is a programming language initially designed to interact with elements of web pages. In web browsers,*

Client-Side Scripting: *JavaScript is mainly used as a client-side scripting language, meaning it runs on the user's browser after the web page is loaded. It enables developers to create interactive web pages by modifying the content and behavior of HTML elements.*

Core Functionality: *JavaScript provides core programming features such as variables, data types, operators, control structures (like conditionals and loops), functions, and error handling. These features allow developers to create algorithms and logic within their applications.*

JavaScript: *Cypress is built on top of JavaScript and utilizes its syntax and features. As a testing framework, Cypress allows you to write test scripts using JavaScript to interact with web elements, simulate user actions, and make assertions about the application's behavior.*

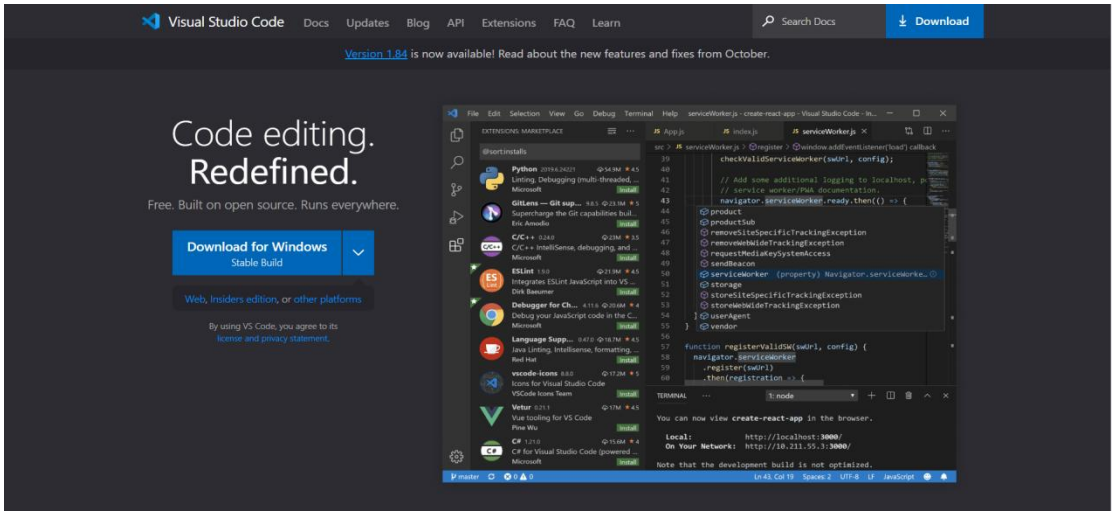
1.2 INSTALL NODE.JS AND SETUP VSCODE EDITOR

To write and run Javascript code, first, let's set up the environment. We need the following components:

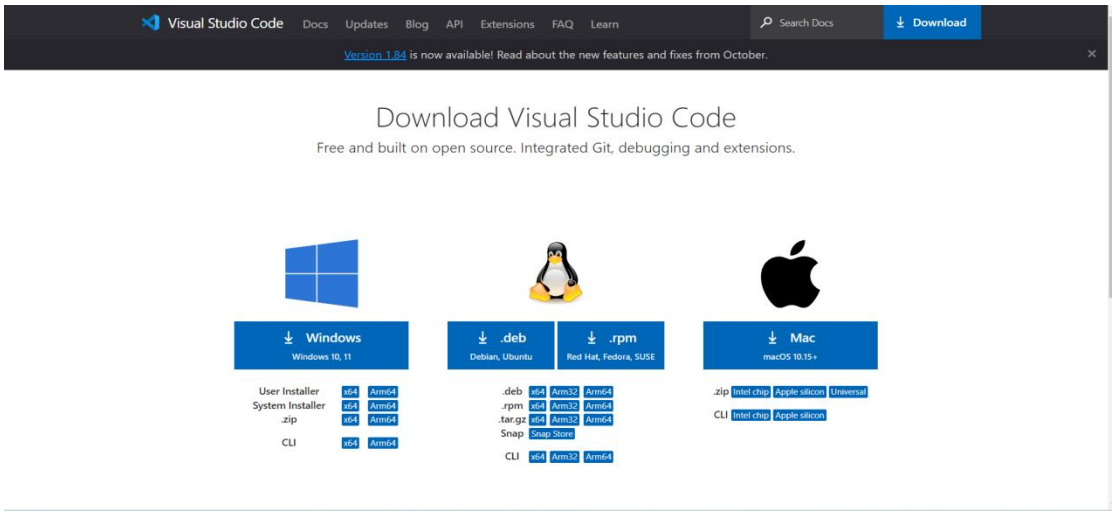
1. Visual Studio Code (VS Code): a code editor.
2. Node.js: a JavaScript runtime environment.

1. install Visual Studio Code (VS Code), please follow these steps:

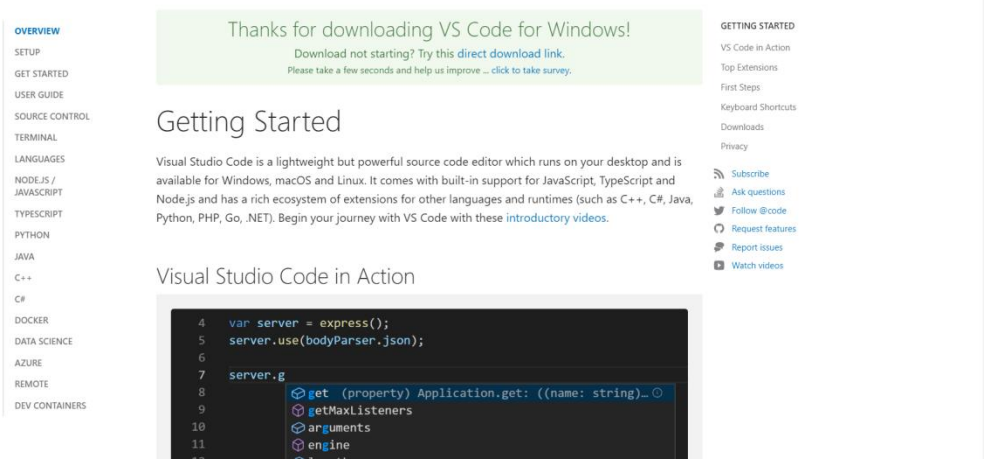
1. Visit the official Visual Studio Code website at https://code.visualstudio.com/.



2. Click on the "Download" button for the operating system (Windows, macOS, or Linux) you are using.

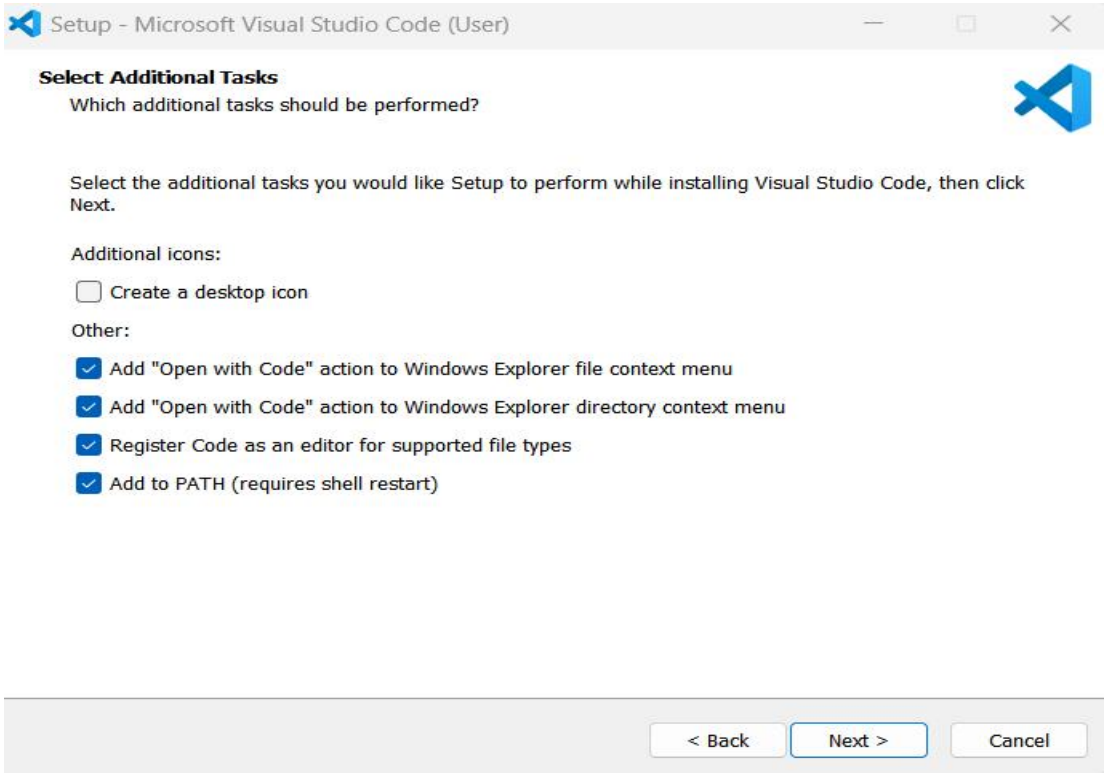


3. The executable file has been downloaded

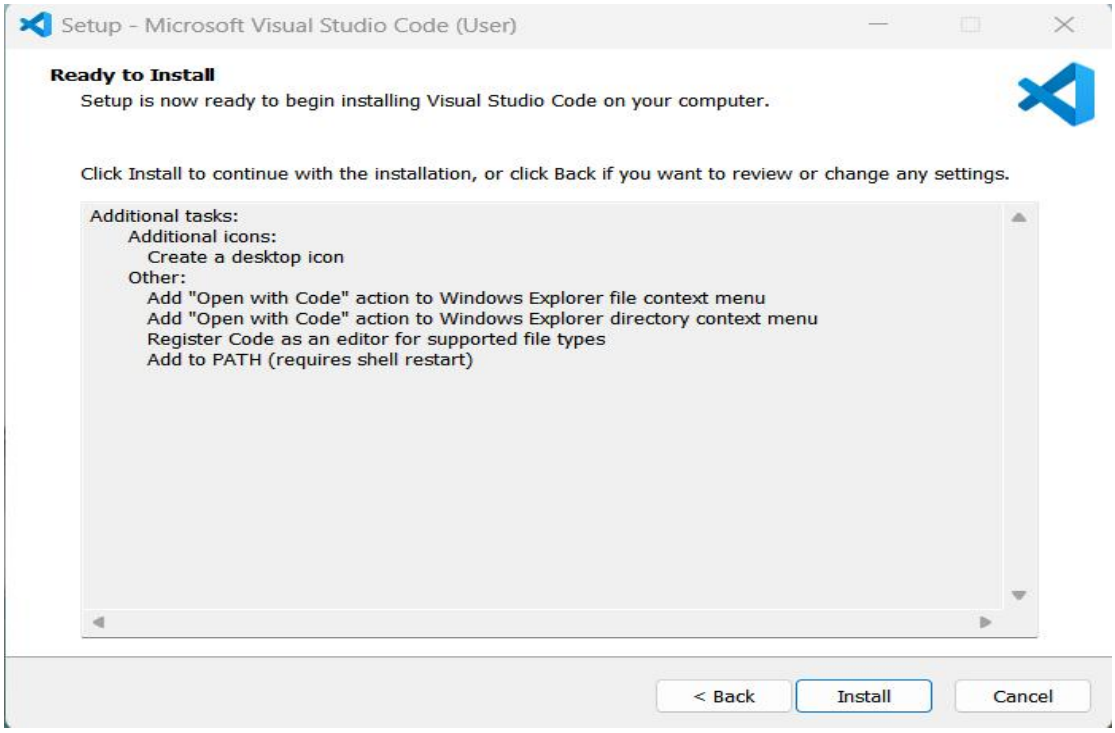


4. Once the download is complete, open the downloaded file from the folder and start the installation process by selecting "I accept the agreement"

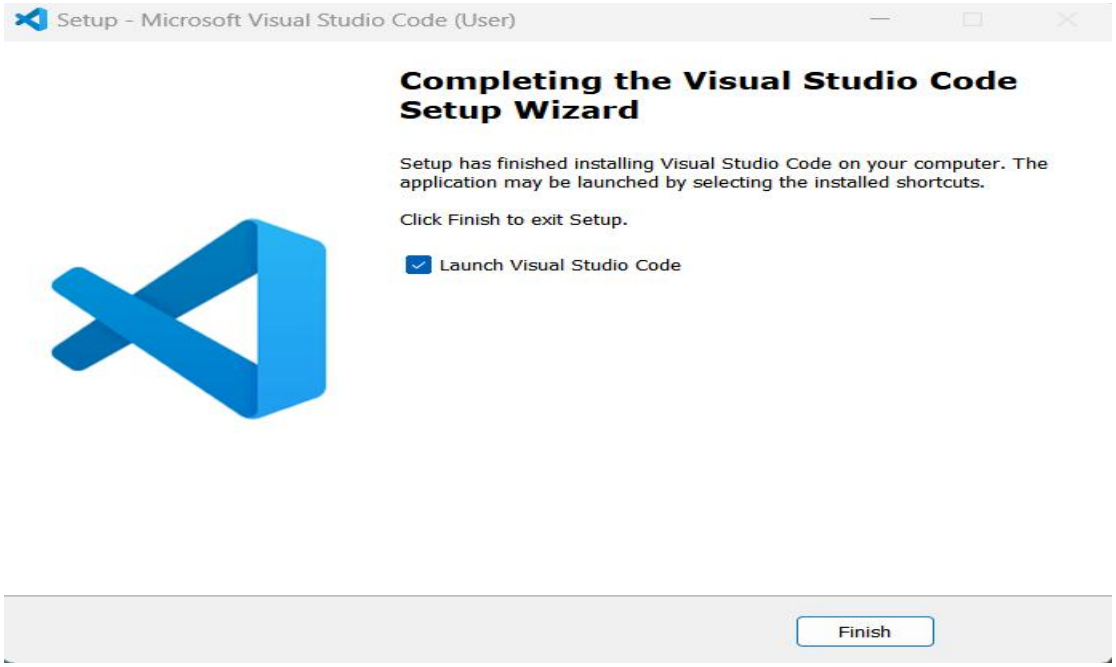
5. Check all option from listed option and click next.



6. Continue the installation process by pressing the "Install" option.



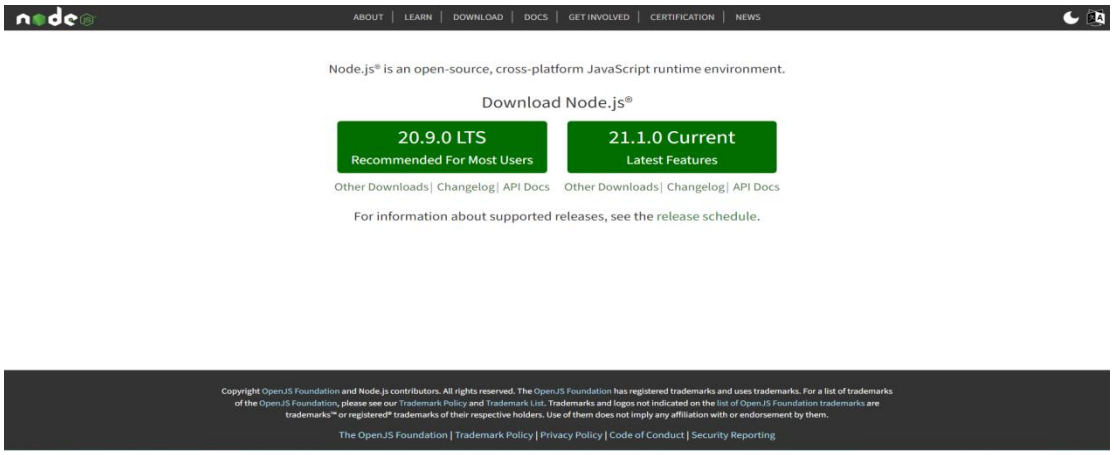
7. Once installtion complete to launch the vsicode check on "**launch visual studio code" option and click finish button**.



Congratulations! You have successfully viscode editor on your computer.

2. Install Node.js please follow these steps:

1. Visit the official Node.js website at https://nodejs.org/.



2. On the homepage, you will see two versions available for download: LTS (Long-Term Support) and Current. It is recommended to choose the LTS version for stability.

Select recommanded for most user opetion.

3. Click on the LTS version to download the installer package for your operating system (Windows, macOS, or Linux).

4. Once the installer package is downloaded, run the installer by double-clicking on the downloaded file.

5. After the installation is complete, open the command prompt or terminal and type "node -v" to verify if Node.js is installed successfully. You should see the version number displayed.

Congratulations! You have successfully installed Node.js on your computer.

1.3 JAVASCRIPT STATEMENTS

A computer program is a list of "**instructions**" to be "executed" by a computer.
In a programming language, these programming instructions are called statements. A JavaScript program is a list of programming statements.

JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.

Semicolons
Semicolons separate JavaScript statements. Add a semicolon at the end of each executable statement:

JavaScript White Space
JavaScript ignores multiple spaces. You can add white space to your script to make it more readable. A good practice is to put spaces around operators (= + - * /):

```
let x = y + z;
```

JavaScript Code Blocks
JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
The purpose of code blocks is to define statements to be executed together.
One place you will find statements grouped together in blocks, is in JavaScript functions:

JavaScript syntax:- is the set of rules, how JavaScript programs are constructed

1.4 JAVASCRIPT FIRST CODE AND OUT PUT

To write first code. Of javascript follow the following step.

Crate folder in descktop and name **Javascript**.
Open this foder by viscode.
Crate **first_js_code.js** file in the Javascript foder.
Type the following code to print hello world text in Terminal.

```
console.log("Hello, World")
```

To run the code.
Open terminal and type the following.

```
node first_js_code.js
```

1.5 JAVASCRIPT COMMENTS

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

Single Line Comments:
Single-line comments are used to comment on a single line of code. They start with // (double forward slash) Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

```
// This is a my first javascript code;
console.log("Hello, World")
```

Multi-line Comments:
Multi-line comments, also known as block comments, are used to comment on multiple lines or to comment out a block of code. They start with **/* and end with */**. Here's an example.

```
/*
This is a multi-line comment.
This is javascrpit cours for class of QA.
Coz for cypress class
*/
console.log("Multi-Line Comment Example")
```

1.6 JAVASCRIPT VARIABLES:

Variables are containers for storing data (storing values).
Variables allow you to store and retrieve data, perform calculations, and control the flow of your program.

Example

```
var age = 24
let name = "CRAFT";
const ID = 314001;
```

In the above example.

Varabile age store numbrical value (data) 24

Varibale name store textual value (data) "Efrem"

Varibel ID store numbrical value (data) 314001

All JavaScript variables must be identified with unique names. These unique names are called **identifiers**. Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.
- By convention, variable names use camelcase.

  example:- yourAge, and myName.

Declare a variable

4 Ways to Declare a JavaScript Variable:

Using var

Using let

Using const

Using nothing

NOTE: will discuss more in the next topic about declarative keywords.

Initialize a variable

Once you have declared a variable, you can initialize it with a value. To initialize a variable, you specify the variable name, followed by an equals sign (=) and a value.

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator use for assign value to variable

```
var age = 24   // for age varibale 24 value assigened
```

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value undefined

1.7 LET ,CONST ,VAR KEYWORD

1. **let:**

The let keyword is used to declare variables that are:-

1. **block-scoped,** meaning they are limited to the block of code in which they are declared (e.g., inside a function, loop, or conditional statement). 2. Variables declared with let **can be reassigned** a new value,

3. Variables declared **cannot be redeclared** within the same scope.

Example:

```
let age = 25;

age = 30; // Valid - Reassignment

let age = 35; // Invalid - Redefinition within the same scope not allowed.
```

2. **const:**

The const keyword is used to declare variables that are

a) block-scoped

b) Variables declared with const **can not be reassigned**.

c) Variables declared **cannot be redeclared** within the same scope.

Example:

```
const PI = 3.14;
PI = 3.14159; // Invalid - Reassignment not allowed
const PI = 3.14159; // Invalid - Redefinition within the same scope
```

3. **var:**

Variables declared with var are function-scoped, meaning they are accessible throughout the entire function in which they are declared, regardless of block boundaries. Additionally, unlike let and const, var variables can be **redeclared** and **reassigned** within the same scope.
Example:

```
var x = 5;
var x = 10; // Valid - Redeclaration within the same scope

x = 15; // Valid - Reassignment

 {
   var y = 20;  // y is accessible here within the block
 }

 console.log(y); // regadless of block y can be access here also.
```

1.8 JAVASCRIPT DATA TYPES

In JavaScript, there are several built-in data types that are used to represent different kinds of values.

1. Number:
The number data type represents numeric values. It includes both integers and floating-point numbers.
Example:

```
// This is the example of numbric data.
var num1 = 5;  // +Ve int value
var num2 = -10; // -Ve int Valid
PI = 3.14  // +Ve float value

console.log(num1);
console.log(num2);
console.log(PI);
```

2. **String:**

The string data type represents sequences of characters enclosed in single quotes (') or double quotes (").

A string that begins with a double quote must end with a double quote. Likewise, a string that begins with a single quote must also end with a single quote:

Example:

```
// This is the example of String data.

var message = "Hello, World";  // duble quates
var greeting = 'hi every one'; // single quote.

console.log(message);
console.log(greeting);
```

3. **Boolean:**

The boolean data type represents logical values indicating either true or false. For example:

```
var isActive= true;
var Status= false;

console.log(isActive);
console.log(Status);
```

4. **undefined**: is a primitive type that has only one value undefined. By default, when a variable is declared but not initialized, it is assigned the value of undefined.

Example:

```
var counter;
console.log(counter);
```

5. **null**: is a primitive data type that also has only one value null.

Example:

```
var counter = null;
console.log(counter);
```

JavaScript defines that null is equal to undefined as follows:

6. NaN: stands for Not a Number. It is a special numeric value that indicates an invalid number. For
example, the division of a string by a number returns NaN.

```
var counter = null;
console.log('a'/2);
```

1.9 JAVASCRIPT OPERATORS

1. Arithmetic Operators:
Arithmetic operators are used to perform mathematical calculations.

- Addition (+): Adds two values together.
- Subtraction (-): Subtracts one value from another.
- Multiplication (*): Multiplies two values.
- Division (/): Divides one value by another.
- Modulus (%): Returns the remainder of a division operation.

Example:

```
let x = 5;
let y = 2;

console.log(x + y); // Output: 7 (addition)
console.log(x - y); // Output: 3 (subtraction)
console.log(x * y); // Output: 10 (multiplication)
console.log(x / y); // Output: 2.5 (division)
console.log(x % y); // Output: 1 (modulus)
```

2. Assignment Operators:
Assignment operators are used to assign values to variables.

- Assignment (=): Assigns a value to a variable.
- Addition assignment (+=): Adds a value to the variable and assigns the result.
- Subtraction assignment (-=): Subtracts a value from the variable and assigns the result.
- Multiplication assignment (*=): Multiplies the variable by a value and assigns the result.
- Division assignment (/=): Divides the variable by a value and assigns the result.

- Modulus assignment (%=): Performs modulus operation on the variable and assigns the result.

Example:

```
let a = 10;
let b = 5;

a += b; // Equivalent to: a = a + b;
console.log(a); // Output: 15

b -= 3; // Equivalent to: b = b - 3;
console.log(b); // Output: 2

let c = 2;
c *= 3; // Equivalent to: c = c * 3;
console.log(c); // Output: 6

let d = 10;d /= 5; // Equivalent to: d = d / 5;
console.log(d); // Output: 2

let e = 7;e %= 4; // Equivalent to: e = e % 4;
console.log(e); // Output: 3
```

4. Comparison Operators:

Comparison operators are used to compare values and return a boolean result.

- **Loose Equality (==):**
  The loose equality operator compares two values for equality, allowing for type coercion. It converts the operands to a common type and then compares their values.
- **Strict Equality (===):**
  The strict equality operator compares two values for equality without performing type coercion. It checks both the value and the type of the operands.
- Not **Loose** equal to (!=): Checks if two values are not equal.
- **Not Strict Equality (===): checks if twh data not strict equal.**
- 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.

Example:

```
let p = 5;
let q = 3;
```

```
console.log(p == q);  // Output: false (equal to)
console.log(p === q);  // Output: false (equal to)
console.log(p != q);  // Output: true (not equal to)
console.log(p > q);   // Output: true (greater than)
console.log(p < q);   // Output: false (less than)
console.log(p >= q);  // Output: true (greater than or equal to)
console.log(p <= q);  // Output: false (less than or equal to)
```

3. Logical Operators:

Logical operators are used to combine or manipulate boolean values.

- Logical AND (&&): Returns true if both operands are true.
- Logical OR (||): Returns true if either operand is true.
- Logical NOT (!): Negates a boolean value.

Example:

```
let isTrue = true;
let isFalse = false;

console.log(isTrue && isFalse); // Output: false (logical AND)
console.log(isTrue || isFalse); // Output: true (logical OR)
console.log(!isTrue);           // Output: false (logical NOT)
```

SUMMARY QUESTION.

I. Choose the correct answere from the given multiple choose.

1. What is JavaScript?
a. A markup language
b. A programming language
c. A database management system
d. A styling language

2. JavaScript statements are:
a. Instructions that tell the browser what to do
b. Text displayed on the web page
c. Styling rules for HTML elements
d. Database queries

3. What is the output of the following JavaScript code?

```
console.log("Hello, World!");
```

4. Which of the following is used to write comments in JavaScript?
a. // comment
b. <!-- comment -->
c. /* comment */
d. **comment**

5. How do you declare a variable in JavaScript?
a. var x;
b. int x;
c. let x;
d. const x;

6. Which keyword is used to declare a block-scoped variable in JavaScript?
a. let
b. const
c. var
d. block

7. What are the data types in JavaScript?
a. Number, String, Boolean, Object, Array, Null, Undefined, Symbol
b. Integer, Float, Text, Boolean, Array, Undefined, Symbol
c. Number, Text, Boolean, Object, Array, Null, Undefined, Symbol
d. Integer, String, Boolean, Object, Array, Null, NaN, Symbol

8. What is the operator used for addition in JavaScript?
a. +
b. /
c. *
d. -

9. Which operator is used to assign a value to a variable in JavaScript?
a. =
b. ==
c. ===
d. +=

10. What is the logical AND operator in JavaScript?
a. &&
b. ||
c. !
d. &

11. Which operator is used to check if two values are equal in JavaScript?
a. ==
b. =
c. ===
d. !=

12. What is the output of the following code?

```
console.log(10 > 5);
```

a. true
b. false
c. undefined
d. Error

13. What is the output of the following code?

```
console.log(5 + "5");
```

a. 10
b. 5
c. "55"
d. Error

14. Which keyword is used to declare a constant in JavaScript?
a. var
b. let
c. const
d. constant

15. What is the output of the following code?

```
console.log(typeof "Hello");
```

a. string
b. number
c. boolean
d. undefined

16. How do you increment a variable in JavaScript?
a. x++
b. x += 1
c. x = x + 1
d. All of the above

17. Which operator is used for exponentiation in JavaScript?
a. ^
b. **
c. *
d. /

18. What is the output of the following code?

```
console.log(10 % 3);
```

a. 3
b. 1
c. 0.33333333333
d. Error

19. What is the result of the logical OR operation if one operand is true?
a. true
b. false

c. undefined
d. Error

20. Which operator is used for string concatenation in JavaScript?
a. +
b. -
c. /
d. *

21. What is the output of the following code?

```
console.log(10 >= 5);
```

a. true
b. false
c. undefined
d. Error

22. What is the output of the following code?

```
console.log(5 === "5");
```

a. true
b. false
c. undefined
d. Error

23. How do you declare a multi-line comment in JavaScript?
a. // comment
b. /
c. /* statement */
d. All

MODULE 2:
CONDITIONAL STATEMENT IN JAVASCRIPT

Conditional statements are implemented when you need to perform different actions based on different conditions. And determine whether or not pieces of code can run.

In JavaScript, the conditional statements that we have are

- **if.**
- **if else.**
- **if else if else.**
- **switch statements.**
- **Ternary opertors in javascript.**

2.1 if Statement

The if statement is used when we want a block of code to be run as long as the condition it true.

syntax

```
if ( condition)
{
   // st a t ement
}
```

Example:

```
if (4 * 4 == 16)
  {
    console.log("if the ondition true this will run");
  }
```

2.2 if, else Statement

The if statement executes a block if a condition is true. When the condition is false, it does nothing. But if you want to execute a statement if the condition is false, you can use an if. . .else statement.

The else statement to specify a block of code to be executed if the condition is false.
Multiple else block is NOT al lowed.

syntax

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

Example:

```
let age = 20;

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


   // output

  You are an adult
```

2.3 if, else if, else Statement

The if-else if-else statement in JavaScript allows you to create multiple branching conditions based on different cases. It allows your program to evaluate multiple conditions sequentially and execute the corresponding block of code when a condition is 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 none of the conditions are true
   }
```

Example:

```
let score = 85;

if (score >= 90)
   {
    console.log("Excellent");
   }
else if (score >= 80)
  {
    console.log("Good");
   }
else if (score >= 70)
  {
   console.log("Average");
  }
else
  {
   console.log("Below Average");
  }
```

2.4 switch Statement

The switch statement evaluates an expression, compares its result with case values, and executes the statement associated with the matching case value.

Syntax:

```
switch (expression)
  {
   case value1:
      // Code to be executed if expression matches value1
      break;
    case value2:
      // Code to be executed if expression matches value2
      break;
    case value3:
      // Code to be executed if expression matches value3
      break;
    default:
     // Code to be executed if expression doesn't match any of the cases
       break;
    }
```

How switch work.

First, evaluate the expression inside the parentheses after the switch keyword.

Second, compare the result of the expression with the value1, value2, … in the case branches from
top to bottom. The switch statement uses the strict comparison (===).

Third, execute the statement in the case branch where the result of the expression equals the value that follows the case keyword. then The break statement exits the switch statement.

If the result of the expression does not strictly equal to any value, the switch statement will execute the statement in the default branch.

That the switch statement will stop comparing the expression ' s result with the remaining case values as long as it finds a match.

The switch statement is like the if…else…if statement. But it has more readable syntax

Example:

```
let day = 3;

let dayName;
switch (day)
 {
  case 1:
     dayName = "Monday";
     break;
  case 2:
     dayName = "Tuesday";
     break;
  case 3:
     dayName = "Wednesday";
     break;
  case 4:
     dayName = "Thursday";
     break;
  case 5:
     dayName = "Friday";
     break;
  default:
    dayName = "Invalid day";
    break;
  }

 console.log(dayName);
```

```
// Output: "Wednesday"
```

2.5 Ternary Operator

The ternary operator in JavaScript is a shorthand way to write conditional expressions. It allows you to evaluate a condition and return one of two values based on the result of the condition.
The ternary operator is also known as the conditional operator.

Syntax:

```
condition ? expression1 : expression2
```

Here's a breakdown of the syntax:

- The condition is the expression that is evaluated. It can be any expression that resolves to a boolean value (true or false).
- If the condition is true, expression1 is evaluated and returned as the result.
- If the condition is false, expression2 is evaluated and returned as the result.

Example:

```
let age = 20;

let result = (age >= 18) ? "You are an adult" : "You are a minor";

console.log(result); // Output: "You are an adult"
```

Summary question

1. Which statement is used to perform conditional execution in JavaScript?
a) if statement
b) for statement
c) while statement
d) switch statement

2. In JavaScript, what is the syntax for an if statement?
a) if { }
b) if ( )
c) if [ ]
d) if < >

3. What is the purpose of an else statement?
a) To define a new condition
b) To execute a block of code when the if condition is false
c) To exit a loop
d) To perform a switch case

4. Which of the following is true about if-else if-else statements?
a) You can have only one else if statement
b) You can have multiple else if statements
c) You cannot have an else statement
d) You cannot have any conditions in else if statements

5. What is the purpose of a switch statement?
a) To perform arithmetic calculations
b) To define a new function
c) To evaluate multiple conditions and execute different blocks of code
d) To create loops

6. In a switch statement, what happens if none of the case values match the expression value?
a) It throws an error
b) It executes the default case block
c) It executes the first case block
d) It exits the switch statement

7. Which operator is used to write a ternary operator in JavaScript?
a) ?
b) :

c) =
d) ||

8. What is the purpose of a ternary operator?
a) To perform arithmetic operations
b) To assign a value to a variable
c) To evaluate a condition and choose between two values
d) To create loops

9. Which of the following is the correct syntax for a ternary operator?
a) if ? else :
b) if : else ?
c) ? if : else
d) condition ? expression1 : expression2

10. What is the result of the following ternary expression? true ? "Yes" : "No"
a) true
b) false
c) Yes
d) No

11. In an if-else statement, how many blocks of code can be executed?
a) Only one block
b) Two blocks
c) Three blocks
d) Any number of blocks

12. Which of the following is true about the placement of else if statements in an if-else if-else statement?
a) They can only be placed at the beginning
b) They can only be placed at the end
c) They can be placed anywhere between if and else statements
d) They are not allowed in if-else if-else statements

13. What is the output of the following code?

```
let num = 5;
```

```
let result = (num > 10) ? "Greater than 10" : "Less than or equal to 10";
console.log(result);
```

a) Greater than 10

b) Less than or equal to 10

c) 5

d) Error

14. Which statement is used to exit a switch statement?

a) exit

b) break

c) continue

d) return

15. What is the output of the following code?

```
let day = "Monday";
let message = "";
switch (day)
{
  case "Monday":
     message = "Start of the week";
     break;
  case "Friday":
     message = "End of the week";
     break;
  default:
     message = "Some other day";
}

console.log(message);
```

a) Monday

b) Friday

c) Start of the week

d) Some other day

16. In a switch statement, can you have multiple case blocks with the same value?

a) Yes

b) No

c) Only if they are adjacent
d) Only if the values are strings

17. What is the output of the following code?

```
 let x = 10;
 let result = (x > 5) ? "Greater than 5" : (x > 0) ? "Greater than 0" :
"Less than or equal to 0";

 console.log(result);
```

a) Greater than 5
b) Greater than 0
c) Less than or equal to 0
d) 10

18. In an if statement, what happens if the condition evaluates to false and there is no else statement?
a) The program exits
b) The program continues to the next line of code after the if statement
c) The program throws an error
d) The program enters an infinite loop

19. Which of the following is the correct syntax for an if-else if-else statement?
a) if { } else if { } else { }
b) if ( ) else if ( ) else ( )
c) if [ ] else if [ ] else [ ]
d) if < > else if < > else < >

20. What is the result of the following ternary expression? false ? "Yes" : "No"
a) true
b) false
c) Yes
d) No

MODULE 3
ITERATION IN JAVASCRIPT

Loop (Iteration) statement in JavaScript

Loops offer a quick and easy way to do something repeatedly. In JavaScript, there are many different kinds of loops, but they all essentially do the same thing: The various loop mechanisms offer different ways to determine the start and end points of the loop.

1. for loop
2. while loop
3. do while loop

1. JavaScript for loop

Loops are handy, if you want to run the same code over and over again, each time with a different value.

The for statement creates a loop with 3 optional expressions:

syntax:

```
for (initial i zer; condition; iterator)
{
// statements
}
```

1) **ini t ializer**: The for statement executes the initial i zer only once the loop starts. Typical ly, you declare and initial i ze a local loop variable in the initial i zer.

2) **condi t ion**: The condition is a boolean expression that determines whether the for should execute the next iteration. The for statement evaluates the condition before each iteration. If the condition is true, it executes the next iteration. Otherwise, it' l l end the loop.

3) **i terator**: The for statement executes the iterator after each iteration.

Example.

```
for (let i = 0; i < 5; i++)
    {
        console.log(i);
    }
```

In this example, the for loop will iterate five times. Here's how it works:

1. Initialization: let i = 0 - This sets the initial value of the loop counter i to 0.
2. Condition: i < 5 - This specifies the condition that needs to be true for the loop to continue executing. As long as i is less than 5, the loop will continue.
3. Body: console.log(i) - This is the code that will be executed in each iteration of the loop. In this case, it will log the current value of i to the console.
4. Increment: i++ - This statement is executed at the end of each iteration and increments the value of i by 1.

2. Javascript while loop

The JavaScript whi le statement creates a loop that executes a block as long as a condition evaluates to true.

syntax:

```
whi le (expression)
{
// statement
}
```

- The while statement evaluates the expression **before each iterat ion** of the loop.
- If the expression **evaluates to true,** the while **statement executes** the statement. Otherwise, the whi le loop exits.
- Because the whi le loop evaluates the expression before each iteration, it is known as a **pretest loop.**
- If the expression evaluates to **false** before the **loop enters**, the whi le loop wi l l never execute.

```
let i = 0;while (i < 5)
  {
    console.log(i);  i++;
  }
```

In this example, the while loop will iterate as long as the condition i < 5 is true. Here's how it works:

1. Initialization: let i = 0 - This sets the initial value of the loop counter i to 0.

2. Condition: i < 5 - This specifies the condition that needs to be true for the loop to continue executing. As long as i is less than 5, the loop will continue.
3. Body: console.log(i) - This is the code that will be executed in each iteration of the loop. In this case, it will log the current value of i to the console.
4. Increment: i++ - This statement is executed at the end of each iteration and increments the value of i by 1.

3. JavaScript do...while Loop

The do. . .whi le loop statement creates a loop that executes a block unti l a condition evaluates to false.

syntax:

```
do {
    statement;
} while(expression);
```

- The do-whi le loop always executes the statement at least once before evaluating the expression. Because the do. . .whi le loop evaluates expression after each iteration, it' s often referred to as a ***post - test loop***.
- Inside the loop body, you need to make changes to some variables to ensure that the expression is false after some iterations. Otherwise, you ' l l have an indefinite loop.

Example.

```
let i = 0;
do {
   console.log(i);
   i++;
  } while (i < 5);
```

In this example, the while loop will iterate as long as the condition i < 5 is true. Here's how it works:

1. Initialization: let i = 0 - This sets the initial value of the loop counter i to 0.
2. Condition: i < 5 - This specifies the condition that needs to be true for the loop to continue executing. As long as i is less than 5, the loop will continue.

3. Body: console.log(i) - This is the code that will be executed in each iteration of the loop. In this case, it will log the current value of i to the console.
4. Increment: i++ - This statement is executed at the end of each iteration and increments the value of i by 1.

Break and Continue

1. The **break statement** in JavaScript is used to terminate the execution of a loop prematurely. When a break statement is encountered within a loop, the loop is immediately exited, and the program continues with the next statement after the loop.

Example

```
for (let i = 0; i < 10; i++)
   {
      if (i === 5)
        {
          break;
        }
       console.log(i)
   }
```

when i is equal to 5, the break statement terminates the loop. Hence, the output doesn't include values greater than or equal to 5.

Output

0

1

2

3

4

2. The **continue** statement: is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

Example.

```
For (let i = 0; i < 5; i++)
 {
   if (i === 3)
   {
     continue;
```

```
  }
  console.log(i);
}
```

This means :- When i is equal to 3, the continue statement skips the third iteration. Then, i becomes 4 and the test condition and continue statement is evaluated again.
Hence, 4 and 5 are printed in the next two iterations.
Output
1
2
4
5

Summary Question

1.Which of the following statements creates a loop with three optional expressions?
a) if statement
b) switch statement
c) for loop
d) while loop

2.Which loop mechanism is known as a pretest loop?
a) for loop
b) while loop
c) do...while loop
d) switch statement

3.What is the purpose of the iterator in a for loop?
a) It executes the loop condition.
b) It executes the loop body.
c) It increments the loop counter.
d) It terminates the loop prematurely.

4.Which loop mechanism always executes the loop body at least once before evaluating the condition?
a) for loop
b) while loop

c) do...while loop
d) switch statement

5.What does the continue statement do in a loop?
a) It terminates the loop prematurely.
b) It skips the current iteration and moves to the next iteration.
c) It evaluates the loop condition.
d) It executes the loop body.

6.Which loop mechanism evaluates the loop condition before each iteration?
a) for loop
b) while loop
c) do...while loop
d) switch statement

7.What happens when a break statement is encountered within a loop?
a) The loop is terminated prematurely.
b) The loop body is skipped for the current iteration.
c) The loop condition is evaluated.
d) The loop counter is incremented.

8.Which loop mechanism is known as a post-test loop?
a) for loop
b) while loop
c) do...while loop
d) switch statement

9.In a for loop, where is the iterator executed?
a) Before each iteration
b) After each iteration
c) Before the loop starts
d) After the loop ends

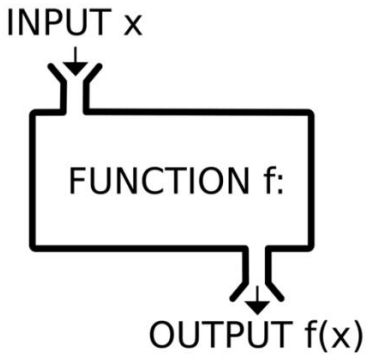
10. Which loop mechanism offers a quick and easy way to do something repeatedly in JavaScript?
a) if statement
b) switch statement
c) for loop
d) while loop

MODULE 4:
JAVASCRIPT FUNCTIONS

4.1 WHAT IS FUNCTIONS

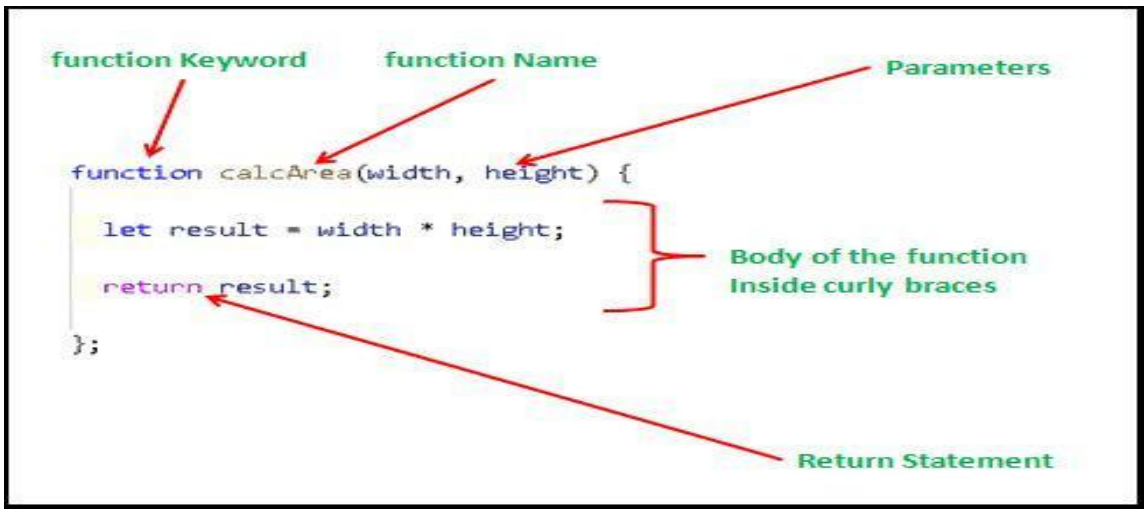
A JavaScript function is a block of code designed to perform a particular task. When developing an application, we often need to perform the same action in many places. To avoid repeating the same code all over places, we can use a function to wrap that code and reuse it.

JavaScript provides many built-in functions such as parseInt() and parseFloat()
A JavaScript function is executed when invokes it (calls it).
Functions also make code readable & maintainable.



4.2 DECLARE A FUNCTION

To declare a function, you use the function keyword, followed by the function name, a list of parameters, and the function body as follows and return value or espersione .



Syntax:

```
function functionName(parameter1, parameter2, ...)
  {
     // Code to be executed  // ...
     return value;// Optional return statement
  }
```

Example.

```
function addNumbers(num1, num2)
  {
      let sum = num1 + num2;
      return sum;
   }
console.log(addNumbers(5, 3)); // Output: 8
```

1. function: The keyword used to declare a function.
2. functionName: The name of the function, which is used to call the function later in the program.
3. parameters: Optional input values that the function can accept. They are defined inside the parentheses and separated by commas. These parameters act as placeholders for the actual values that will be passed into the function when it is called.
4. Function body: The set of instructions that the function executes. It is enclosed within curly braces {}.
5. return statement: An optional statement that specifies the value to be returned by the function. When the return statement is encountered, the function stops executing and the specified value is returned to the caller. If there is no return statement, the function returns undefined by default.

4.3 CALLING A FUNCTION

To use a function, you need to call it. Calling a function is also known as invoking a function.
To call a function, you use its name followed by arguments enclosing in parentheses.
When calling a function, JavaScript executes the code inside the function body

```
functionName(argument1, argument2, ...);
```

Let's take the example of the addNumbers function from the previous response:
javascript

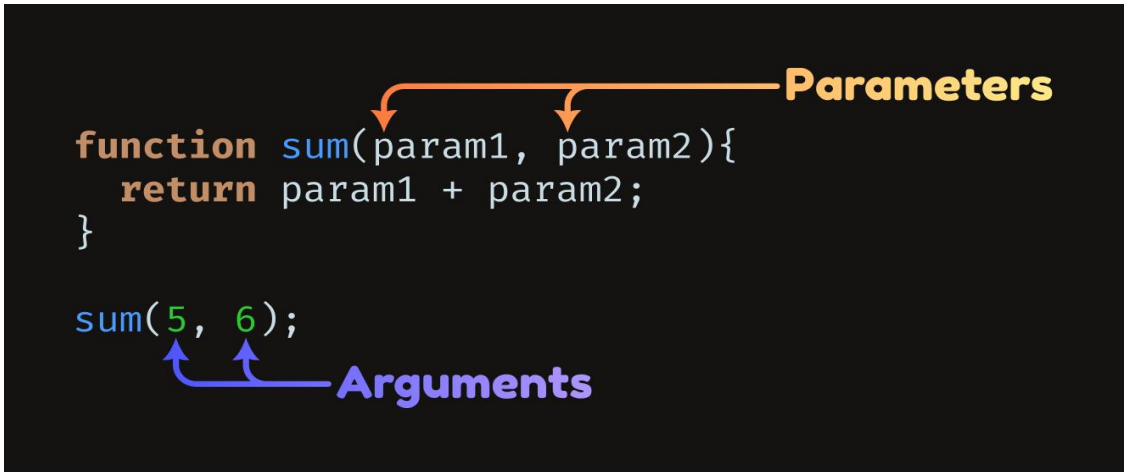
```
function addNumbers(num1, num2)
  {
    let sum = num1 + num2;
    return sum;
  }
```

To call the addNumbers function and get the result, you would do the following:
javascript

```
let result = addNumbers(5, 3);
console.log(result); // Output: 8
```

4.4 PARAMETERS VS. ARGUMENTS

The terms parameters and arguments are often used interchangeably. However, they are essentially different.
When declaring a function, you specify the parameters, when calling a function, you pass the arguments that are corresponding to the parameters.



Example

```
function addNumbers(num1, num2) //
```

In the above addNumber function num1, num2 are parameters.

```
addNumbers(5, 3);
```

n the above addNumber 5,3 are Arguments

4.5 RETURNING A VALUE

Returning a value from a function means that the function produces a result or outcome that can be used or accessed after the function call.
The return statement is used to specify the value to be returned by a function.
Every function in JavaScript implicitly returns undefined unless you explicitly specify a return value.
To specify a return value for a function, you use the return statement followed by an expression or a value, like this:

syntax.

```
function functionName(parameter1, parameter2, ...)
{
  // Code to be executed  // ...
  return value;
}
```

Example.

```
function multiply(num1, num2)
 {
    let product = num1 * num2;
    return product;
 }
```

In the above code, the multiply function takes two parameters num1 and num2. It multiplies them together and assigns the result to the product variable. Then, the return statement is used to return the value of product as the output of the function.

```
let result = multiply(5, 3);
console.log(result); // Output: 15
```

4.6 FUNCT ION & VARIABLE SCOPE.

Scope refers to the availability of variables and functions in certain parts of the code.
In JavaScript, a variable has two types of scope:

- Global Scope
- Local Scope

Global Scope

A variable declared at the top of a program or outside of a function is considered a global scope variable. can be used anywhere in the program.

The value of a global variable can be changed inside a function

Local Scope

A variable can also have a local scope, i.e it can only be accessed within a function. can be accessed only inside the function.

Example.

```
globalVariable = "I am a global variable"; // Global scope variablevar

function myFunction()
 {
  var localVariable = "I am a local variable"; // Local scope variable
  console.log(globalVariable); // Accessible: Outputs "I am a global variable"
  console.log(localVariable); // Accessible: Outputs "I am a local variable"

 }
console.log(globalVariable); // Accessible: Outputs "I am a global variable"
console.log(localVariable); // Not accessible: Throws a ReferenceError
```

4.7 ANONYMOUS FUNCTIONS

In JavaScript, an anonymous function is a function that does not have a name. It is often used when we need to define a function on the spot without assigning it to a variable or giving it a name.

Syntax.

```
// Anonymous function
function(parameter1, parameter2, ...)
    {
        // Function body
        // …
    }
```

Anonymous functions can be used in various ways, such as:

1. Assigning to a variable:

```
 var greeting = function(name)
  {
     console.log("Hello, " + name + "!");
 };

 greeting("Alice"); // Output: Hello, Alice!
```

2. As a callback function:

```
function performOperation(a, b, operation)
  {
     var result = operation(a, b);
     console.log("Result: " + result);
  }
 performOperation(5, 3, function(x, y)
      {
        return x + y;
      }); // Output: Result: 8
```

3. Immediately invoked:

```
 ( function()
    {
       console.log("I am an immediately invoked anonymous function!");
    }
 )();
```

4.8 ARROW FUNCTIONS

Arrow functions, also known as fat arrow functions, are a concise syntax for defining functions in JavaScript. They provide a shorter and more readable way to write functions compared to traditional function expressions. Arrow functions were introduced in ES6 (ECMAScript 2015).

```
(parameter1, parameter2, ...) => {

    // Function body  // …

}
```

Example.

```
// Example 1: Basic arrow functionconst
add = (a, b) => a + b;
console.log(add(2, 3)); // Output: 5//

Example 2: Arrow function with implicit return
const multiply = (a, b) => a * b;
console.log(multiply(4, 5)); // Output: 20//

Example 3: Arrow function with no parameters
const sayHello = () => {  console.log("Hello!");};
sayHello(); // Output: Hello!
```

SUMMARY QUESTION

1. What is a JavaScript function?
A. block of code used for styling web pages.
B. data type used to store values.
C. block of code designed to perform a specific task.
D. method used to manipulate arrays.

2. How do you declare a function in JavaScript?
A. Using the function keyword followed by the function name, parameters, and function body.
B. Using the let keyword followed by the function name and function body.
C. Using the function keyword followed by the function name and function body.
D. Using the const keyword followed by the function name, parameters, and function body.

3. What is the purpose of parameters in a function?
A. Parameters are used to specify the return value of a function.
B. Parameters are used to define the name of a function.
C. Parameters are used to pass values into a function.
D. Parameters are used to specify the data type of a function.

4. How do you call or invoke a function in JavaScript?
A. By using the call keyword followed by the function name.
B. By using the execute keyword followed by the function name.
C. By using the invoke keyword followed by the function name.
D. By using the function name followed by parentheses and arguments.

5. What is the difference between parameters and arguments?
A. Parameters are used when declaring a function, while arguments are passed when calling a function.
B. Parameters and arguments are two different names for the same concept.
C. Parameters are used when calling a function, while arguments are passed when declaring a function.
D. Parameters and arguments are interchangeable terms in JavaScript.

6. How do you return a value from a function in JavaScript?
A. By using the output statement followed by the value.
B. By using the return keyword followed by the value.
C. By using the result keyword followed by the value.
D. By using the output keyword followed by the value.

7. What is the difference between global scope and local scope in JavaScript?
A. Global scope refers to variables accessible only within a function, while local scope refers to variables accessible anywhere in the program.
B. Global scope refers to variables accessible anywhere in the program, while local scope refers to variables accessible only within a function.
C. Global scope refers to variables accessible only within an object, while local scope refers to variables accessible anywhere in the program.
D. Global scope refers to the built-in JavaScript functions, while local scope refers to user-defined functions.

8. What is an anonymous function in JavaScript?
A. A function with no parameters.
B. A function that does not have a name.
C. A function that returns undefined by default.
D. A function that can only be used as a callback.

9. Which version of ECMAScript introduced arrow functions?
A. ECMAScript 2016
B. ECMAScript 2015
C. ECMAScript 2014
D. ECMAScript 2017

10. What is a benefit of using arrow functions in JavaScript?
A. Arrow functions have their own this value.
B. Arrow functions provide a longer syntax compared to regular function expressions.
C. Arrow functions are not suitable for use as callback functions.
D. Arrow functions offer a more concise and readable syntax for function definitions.

MODULE 5:
JAVASCRIPT ARRAY

5.1 WHAT IS JS ARRAY

In JavaScript, an array is an ordered list of values. Each value is called an element specified by an index:

variable can hold only one value. We cannot assign multiple values to a single variable. JavaScript array is a special type of variable, which can store multiple values using a special syntax.

JavaScript array can store a mixed data formats in a single array.
JavaScript arrays are dynamic, which means that they grow or shrink as needed.

5.2 CREATEING JS ARRAY

```
let myArray = [1, 2, 3, 4, 5];
```

The array literal form uses the square brackets [] to wrap a comma-separated list of elements.

You can also create an array, and then provide the elements:

```
let myArray = [];
myArray.push("apple");
myArray.push("banana");
myArray.push("orange");
```

It is not required to store the same type of values in an array. It can store values of different types as well.

```
let mixedArray = [ 1, "apple", true, 25.76 ];
```

5.3 ACCESSING ARRAY ELEMENTS

Array elements (values) can be accessed using an **index**. Specify an index in square brackets with the array name to access the element at a particular index.

like arrayName[index].

```
let myArray = [10, 20, 30, 40, 50];

console.log(myArray[0]); // Output: 10
console.log(myArray[1]); // Output: 20
console.log(myArray[2]); // Output: 30
console.log(myArray[3]); // Output: 40
console.log(myArray[4]); // Output: 50
```

we can use the **arrayName.at(pos)** method to get the element from the specified index. This is the same as arr[index] except that the **.at()** returns an element from the last element if the specified index is negative.

```
let myArray = [10, 20, 30, 40, 50];

console.log(myArray.at(0)); // Output: 10
console.log(myArray.at(1)); // Output: 20
console.log(myArray.at(2)); // Output: 30
console.log(myArray.at(3)); // Output: 40
console.log(myArray.at(4)); // Output: 50
```

How get the array size?
length property of an array returns the number of elements.

```
let myArray = [10, 20, 30, 40, 50];
console.log(myArray.length); // Output: 5
```

5.4 ITERATEAN ARRAY

To iterate over the elements of a JavaScript array, you can use various methods such as a

1. for loop,
2. forEach(),
3. for...of loop,
4. for....in loop

Each method offers different ways to loop through the array and perform operations on its elements.

1. Using a for loop:

```
let myArray = [10, 20, 30, 40, 50];
for (let i = 0; i < myArray.length; i++)
  {
    console.log(myArray[i]);
  }
```

2. Using the forEach() method:

```
 let myArray = [10, 20, 30, 40, 50];

  myArray.forEach(function(element)

    {
     console.log(element);
    });
```

3. Using a for...of loop:

```
let myArray = [10, 20, 30, 40, 50];

for (let element of myArray)
    {
       console.log(element);
    }
```

4. Using a for...in loop:

```
 let myArray = [10, 20, 30, 40, 50];
 for (let index in myArray)
    {
      console.log(index); // Outputs: 0, 1, 2, 3, 4
    }
```

5.5 ARRAY METHODS

JavaScript provides a variety of built-in methods that can be used to manipulate and work with arrays. Here are some commonly used array methods:

1. push(): Adds one or more elements to the end of an array and returns the new length of the array.

```
let myArray = [1, 2, 3];
myArray.push(4, 5);
console.log(myArray); // Output: [1, 2, 3, 4, 5]
```

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

```
let myArray = [1, 2, 3, 4, 5];
let removedElement = myArray.pop();
console.log(removedElement); // Output: 5
console.log(myArray); // Output: [1, 2, 3, 4]
```

3. shift(): Removes the first element from an array and returns that element. It also shifts the remaining elements down to a lower index.

```
let myArray = [1, 2, 3, 4, 5];
let removedElement = myArray.shift();
console.log(removedElement); // Output: 1
console.log(myArray); // Output: [2, 3, 4, 5]
```

4. unshift(): Adds one or more elements to the beginning of an array and returns the new length of the array.

```
let myArray = [1, 2, 3];
myArray.unshift(-1, 0);
console.log(myArray); // Output: [-1, 0, 1, 2, 3]
```

5. concat(): Combines two or more arrays and returns a new array without modifying the original arrays.

```
let array1 = [1, 2, 3];
let array2 = [4, 5];
let combinedArray = array1.concat(array2);
console.log(combinedArray); // Output: [1, 2, 3, 4, 5]
```

6. slice(): Returns a new array containing a portion of the original array, specified by start and end indices. The original array is not modified.

```
let myArray = [1, 2, 3, 4, 5];
let slicedArray = myArray.slice(2, 4);
console.log(slicedArray);
// Output: [3, 4]
```

```
let myArray = [1, 2, 3, 4, 5];
let removedElements = myArray.splice(2, 2, 'a', 'b');
console.log(removedElements); // Output: [3, 4]
console.log(myArray); // Output: [1, 2, 'a', 'b', 5]
```

SUMMARY QUESTION

1. What is a JavaScript array?
a) A type of variable that can store multiple values
b) A function used to create arrays
c) A method for accessing array elements
d) A data type that stores only one value

2. How do you create an array in JavaScript?
a) Using the array() function
b) Using the new Array() syntax
c) Using square brackets [] with a comma-separated list of elements
d) Using curly braces {} with a comma-separated list of elements

3. How do you access array elements in JavaScript?
a) Using the get() method
b) Using the access() function
c) Using square brackets [] with the index number
d) Using the element() method

4. How can you get the length of an array in JavaScript?
a) Using the length() function
b) Using the getSize() method
c) Using the count() function
d) Accessing the length property of the array

5. Which method is used to add elements to the end of an array?
a) append()
b) push()
c) add()
d) insert()

6. Which method is used to remove the last element from an array?
a) removeLast()

b) pop()
c) delete()
d) extract()

7. What does the concat() method do?
a) Concatenates two strings
b) Combines two or more arrays into a new array
c) Deletes elements from an array
d) Splits a string into an array

8. How can you iterate over the elements of an array in JavaScript?
a) Using a for loop
b) Using the iterate() method
c) Using the loop() function
d) Using the iterateArray() method

9. Which method is used to remove and insert elements in an array at a specific position?
a) replace()
b) update()
c) splice()
d) modify()

10. Which method returns a new array containing a portion of the original array?
a) cut()
b) extract()
c) slice()
d) split()