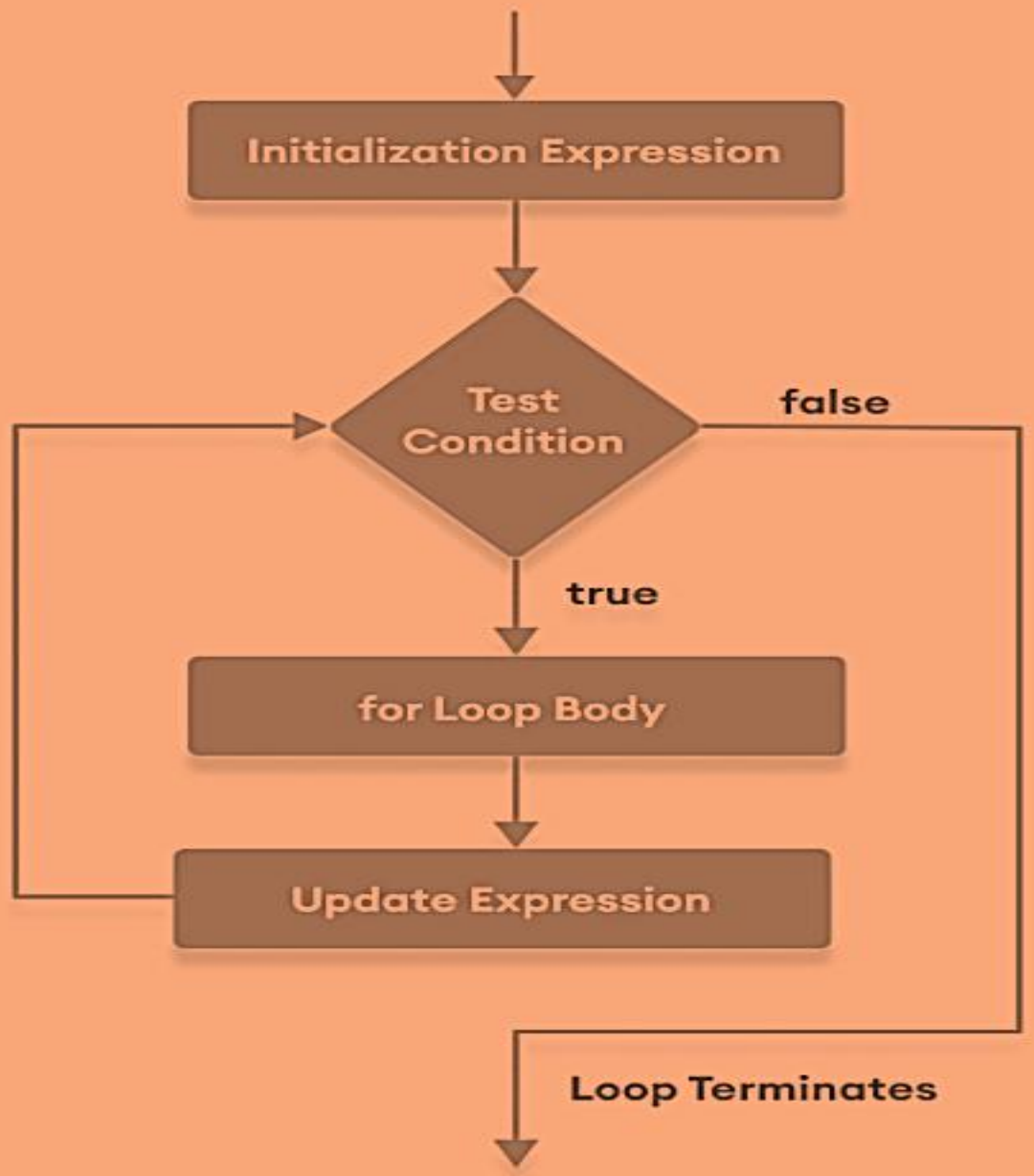
- **JavaScript for loop**
- In programming, loops are used to repeat a block of code.
- For example, if you want to show a message 100 times, then you can use a loop. It's just a simple example; you can achieve much more with loops.
- The syntax of the for loop is:

```
for (initialExpression; condition; updateExpression) {
   // for loop body
}
```

- Example 1: Display a Text Five Times
- // program to display text 5 times
- const n = 5;
- // looping from i = 1 to 5
- for (let i = 1; i <= n; i++) {
- console.log(`I love JavaScript.`);
- }

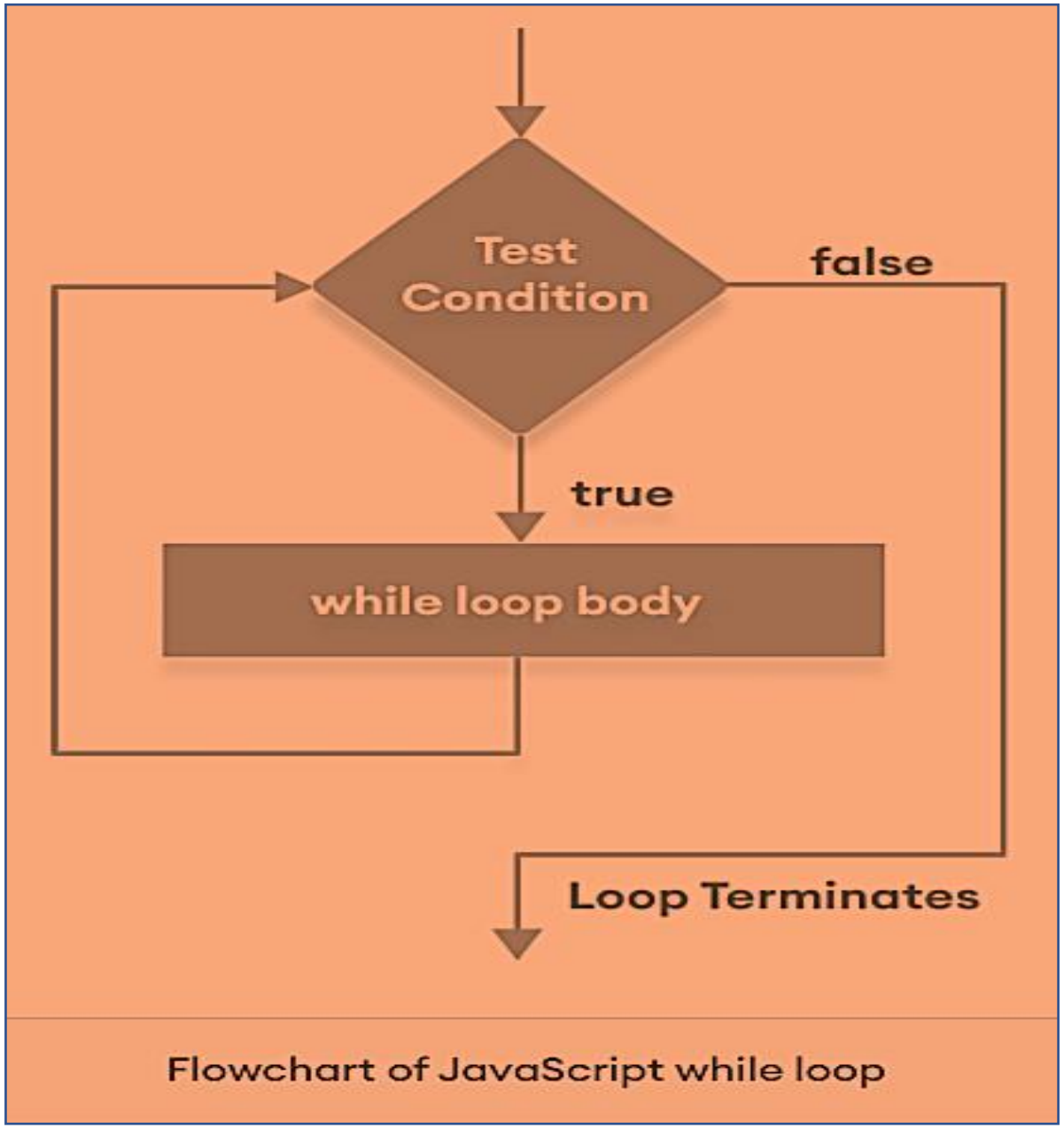


Flowchart of JavaScript for loop

JavaScript while and do...while Loop

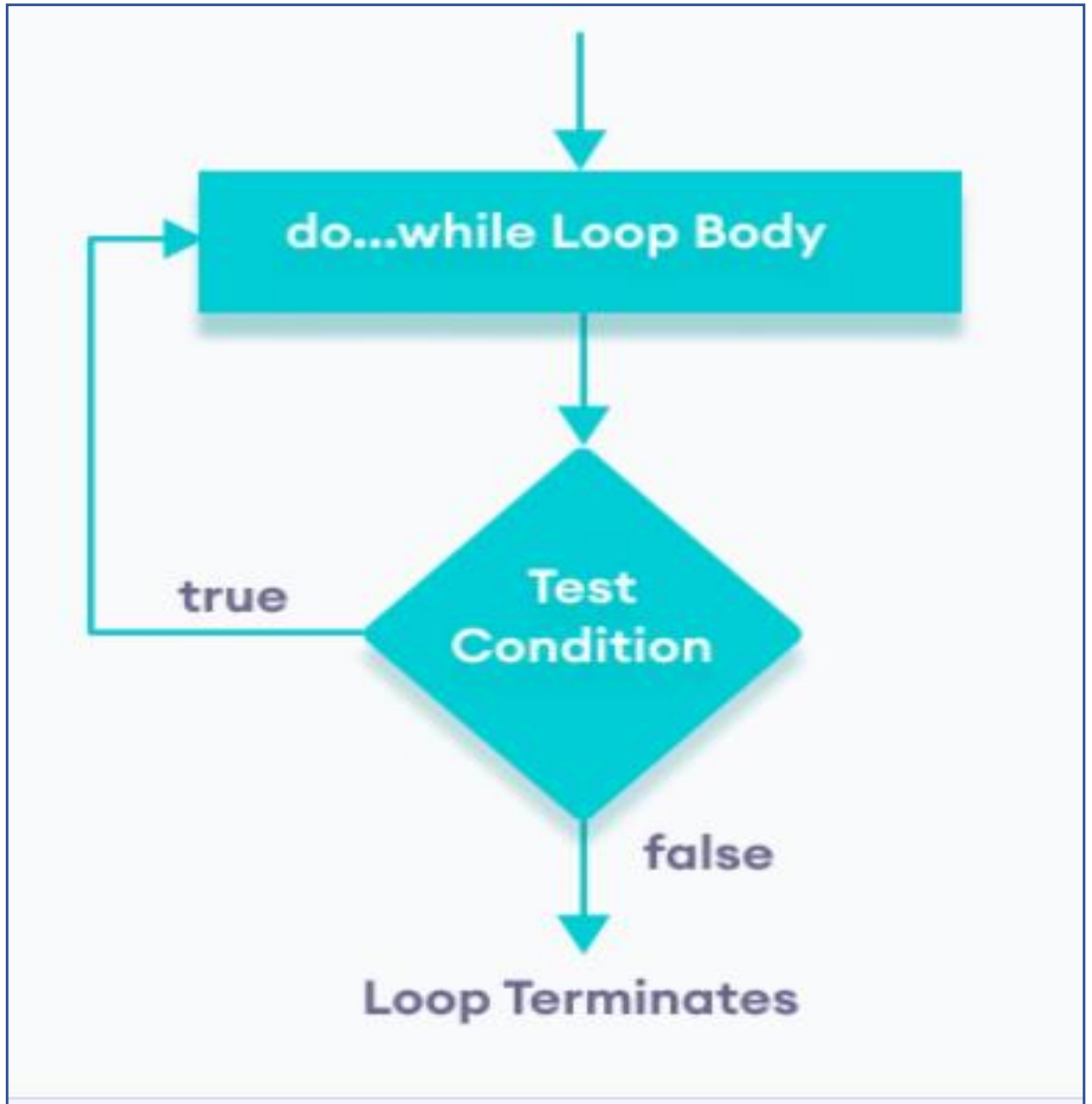
- **JavaScript while Loop**
- The syntax of the while loop is:
- while (condition) {
- // body of loop
- }
- **Example 1: Display Numbers from 1 to 5**
- // program to display numbers from 1 to 5
- // initialize the variable
- let i = 1, n = 5;

- // while loop from i = 1 to 5
- while (i <= n) {
- console.log(i);
- i += 1;
- }



- JavaScript do...while Loop
- The syntax of do...while loop is:
- do {
- // body of loop
- } while(condition)
- Example 3: Display Numbers from 1 to 5
- // program to display numbers
- let i = 1;
- const n = 5;

- // do...while loop from 1 to 5
- do {
- console.log(i);
- i++;
- } while(i <= n);



Flowchart of JavaScript do...while loop

JavaScript break Statement

- The break statement is used to terminate the loop immediately when it is encountered.
- The syntax of the break statement is:
- break [label];
- **Working of JavaScript break Statement**
- **Example 1: break with for Loop**
- // program to print the value of i
- for (let i = 1; i <= 5; i++) {
- // break condition
- if (i == 3) {
- break;
- }
- console.log(i);
- }

```
for (init; condition; update) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

Working of JavaScript break Statement

JavaScript continue Statement

- 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.
- The syntax of the continue statement is:
- continue [label];
- **Example 1: Print the Value of i**

```
// program to print the value of i
for (let i = 1; i <= 5; i++) {

    // condition to continue
    if (i == 3) {
        continue;
    }

    console.log(i);
}
```

Output

1
2
4
5

- **Working of JavaScript continue Statement**

```
for (init; condition; update) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```

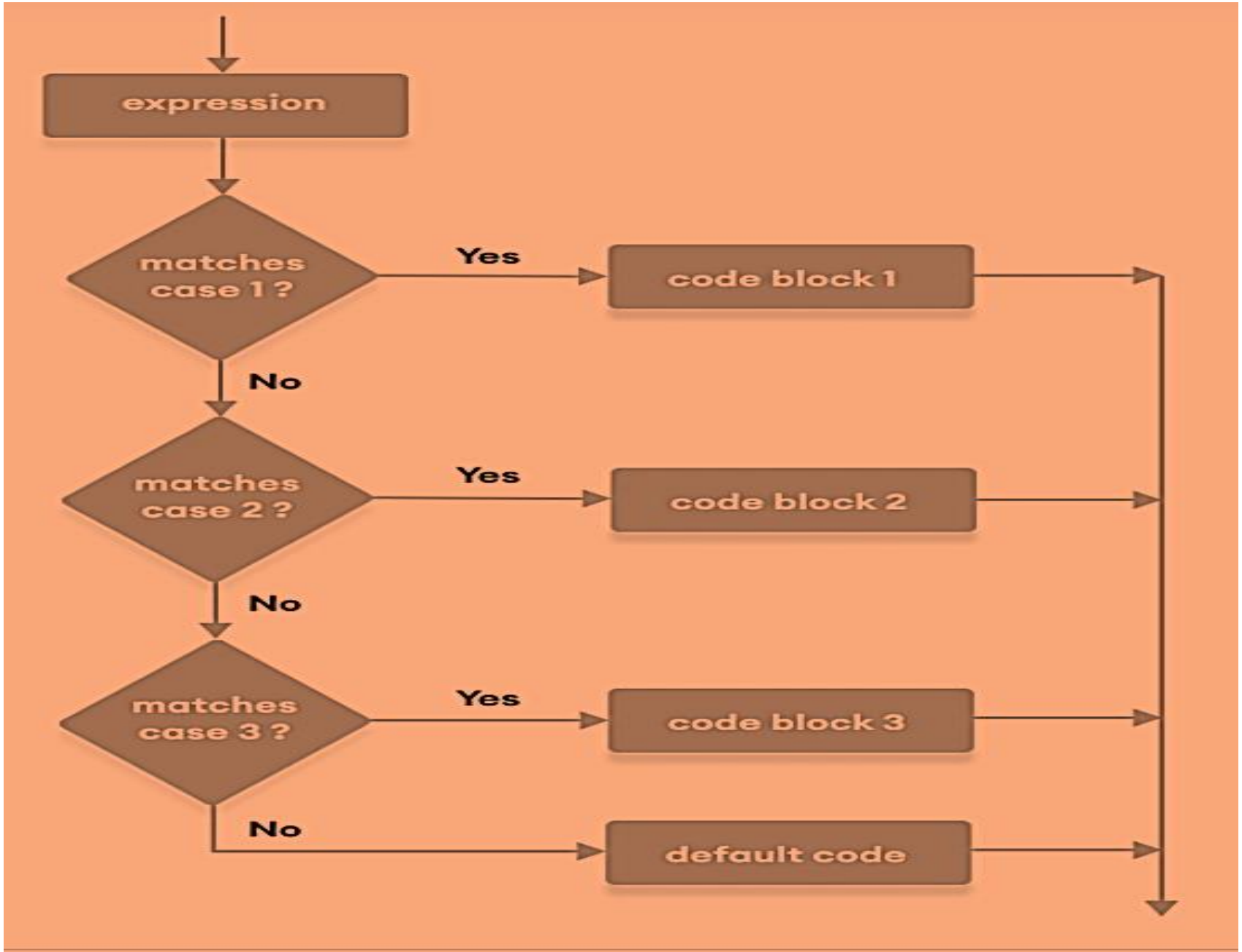
```
while (condition) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```

Working of JavaScript continue Statement

JavaScript Switch Statement

- The JavaScript switch statement is used in decision making.
- The switch statement evaluates an expression and executes the corresponding body that matches the expression's result.
- The syntax of the switch statement is:
- switch(variable/expression) {

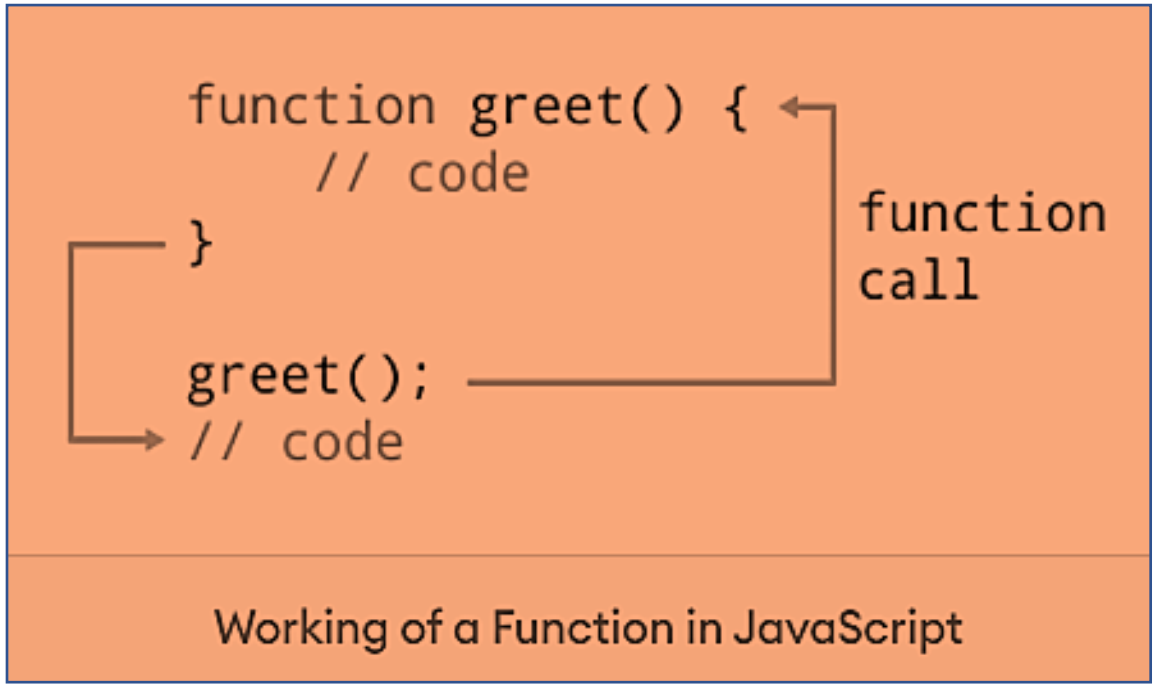
```
  case value1:
    // body of case 1
    break;
  case value2:
    // body of case 2
    break;
  case valueN:
    // body of case N
    break;
  default:
    // body of default
}
```



Flowchart of JavaScript switch statement

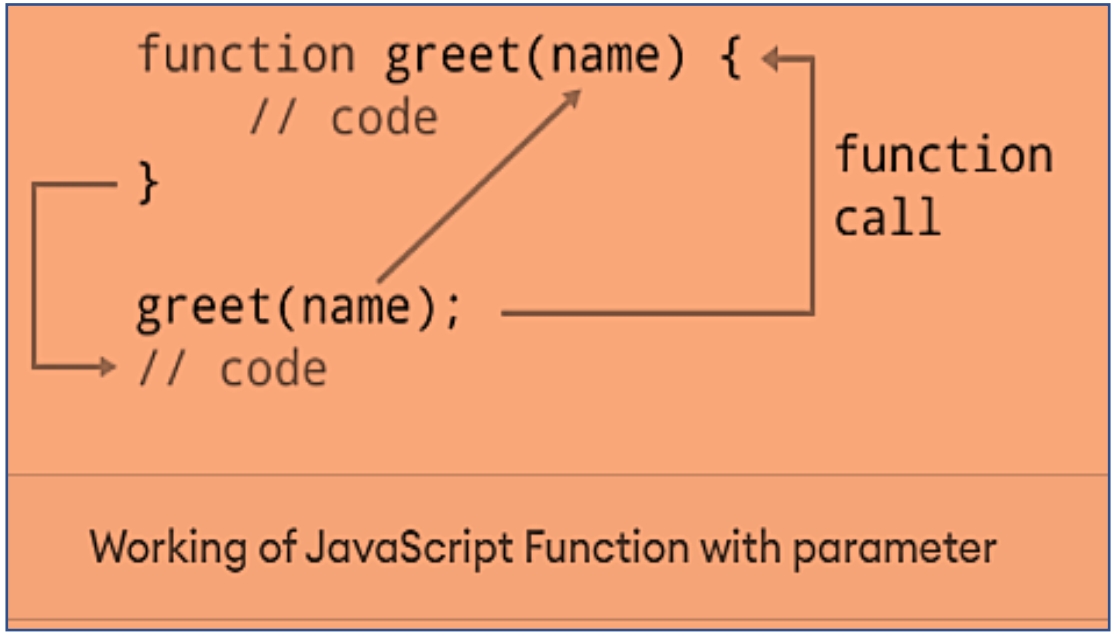
- **Declaring a Function**
- The syntax to declare a function is:
- function nameOfFunction () {
- // function body
- }
- A function is declared using the **function** keyword.
- The basic rules of naming a function are similar to naming a variable.
- It is better to write a descriptive name for your function. For example, if a function is used to add two numbers, you could name the function add or addNumbers.
- The body of function is written within {……}.
- For example,
- // declaring a function named greet()
- function greet() {
- console.log("Hello there");
- }

- **Calling a Function**
- In the above program, we have declared a function named greet(). To use that function, we need to call it.
- Here's how you can call the above greet() function.
- // function call
- greet();
- **Example 1: Display a Text**
- // program to print a text
- // declaring a function
- function greet() {
- console.log("Hello there!");
- }
- // calling the function
- greet();



Working of a Function in JavaScript

- **Function Parameters**
- A function can also be declared with parameters. A parameter is a value that is passed when declaring a function.
- **Example 2: Function with Parameters**
- // program to print the text
- // declaring a function
- function greet(name) {
- console.log("Hello " + name + ":)");
- }
- // variable name can be different
- let name = prompt("Enter a name: ");
- // calling function
- greet(name);



Working of JavaScript Function with parameter

- Function add(a,b){
- Console.log(a+b)
- }

// Calling a function

Add(3,4);

Add(7,9);

JavaScript 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:

1. Global Scope
2. Local Scope

- **Global Scope**
- A variable declared at the top of a program or outside of a function is considered a global scope variable.
- Let's see an example of a global scope variable.

- // program to print a text
- let a = "hello";
- function greet () {
- console.log(a);
- }
- greet(); // hello
- In the above program, variable a is declared at the top of a program and is a global variable.
- It means the variable a can be used anywhere in the program.

- The value of a global variable can be changed inside a function. For example,
- // program to show the change in global variable
- let a = "hello";
- function greet() {
- a = 3;
- }
- // before the function call
- console.log(a);// hello
- //after the function call
- greet();
- console.log(a); // 3

- In JavaScript, a variable can also be used without declaring it.
- If a variable is used without declaring it, that variable automatically becomes a global variable. For example,
- function greet() {
- a = "hello“
- }
- greet();
- console.log(a); // hello
- In the above program, variable a is a global variable.
- If the variable was declared using let a = "hello", the program would throw an error.
- Note: In JavaScript, there is "**strict mode";** in which a variable cannot be used without declaring it.

- A variable can also have a local scope, i.e it can only be accessed within a function.
- **Example 1: Local Scope Variable**

```
// program showing local scope of a variable
let a = "hello";

function greet() {
    let b = "World"
    console.log(a + b);
}

greet();
console.log(a + b); // error
```

Output

```
helloWorld
Uncaught ReferenceError: b is not defined
```

- **let is Block Scoped**
- The let keyword is block-scoped (variable can be accessed only in the immediate block). **Example 2: block-scoped Variable**

```
// program showing block-scoped concept
// global variable
let a = 'Hello';

function greet() {

    // local variable
    let b = 'World';

    console.log(a + ' ' + b);

    if (b == 'World') {

        // block-scoped variable
        let c = 'hello';

        console.log(a + ' ' + b + ' ' + c);
    }

    // variable c cannot be accessed here
    console.log(a + ' ' + b + ' ' + c);
}

greet();
```

Output

```
Hello World
Hello World hello
Uncaught ReferenceError: c is not defined
```

JavaScript Hoisting

- **Hoisting** in JavaScript is a behavior in which a function or a variable can be used before declaration. For example,

```
// using test before declaring
console.log(test);   // undefined
var test;
```

- The above program works and the output will be **undefined.**
- **Variable Hoisting**
- In terms of variables and constants, keyword **var** is hoisted and **let** and **const** does not allow hoisting**.** For example,

```
// program to display value
a = 5;
console.log(a);
var a; // 5
```

- In the above example, variable a is used before declaring it. And the program works and displays the output 5

- However in JavaScript, initializations are not hoisted. For example,

```
// program to display value
console.log(a);
var a = 5;
```

- Output

Undefined

- Only the declaration is moved to the memory in the compile phase.
- Hence, the value of variable **a** is undefined because **a** is printed without initializing it.

- Also, when the variable is used inside the function, the variable is hoisted only to the top of the function. For example,
- // program to display value

```
var a = 4;
function greet() {
    b = 'hello';
    console.log(b); // hello
    var b;
}
greet(); // hello
console.log(b);
```

- **Output**

```
hello
Uncaught ReferenceError: b is not defined
```

Note: In hoisting, the variable declaration is only accessible to the immediate scope.

- **Function Hoisting**
- A function can be called before declaring it. For example,

```
// program to print the text
greet();
function greet() {
   console.log('Hi, there.');
}
```

- **Output**

```
Hi, there
```

- In the above program, the function greet is called before declaring it and the program shows the output. **This is due to hoisting**

- However, when a function is used as an **expression**, an error occurs because only declarations are hoisted. For example;
- **// program to print the text**

```
greet();
let greet = function() {
   console.log('Hi, there.');
}
```

- **Output**

Uncaught ReferenceError: greet is not defined

- If var was used in the above program, the error would be:

Uncaught TypeError: greet is not a function

JavaScript Objects

- JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.
- If you are familiar with other programming languages, JavaScript objects are a bit different. You do not need to create classes in order to create objects.
- Here is an example of a JavaScript object.
- // object
- const student = {
- firstName: 'ram',
- class: 10
- };
- **JavaScript Object Declaration**
- The syntax to declare an object is:

```
const object_name = {
   key1: value1,
   key2: value2
}
```

- Each member of an object is a key: value pair separated by commas and enclosed in curly braces {}…. For example,
- // object creation

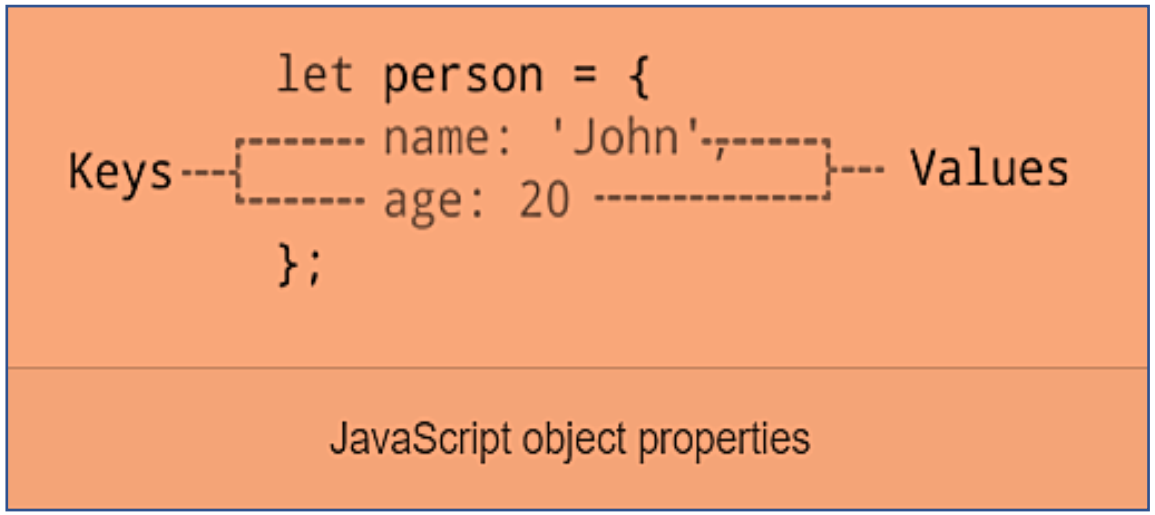
```
const person = {
   name: 'John',
   age: 20
};
console.log(typeof person); // object
```

- You can also define an object in a single line.
- const person = { name: 'John', age: 20 };
- In the above example, **name and age are keys**, and **John and 20** are values respectively.

- **JavaScript Object Properties**
- In JavaScript, "key: value" pairs are called **properties**. For example,

```
let person = {
    name: 'John',
    age: 20
};
```

- Here, name: 'John' and age: 20 are properties.



- **Accessing Object Properties**
- You can access the **value** of a property by using its **key**.

1. Using dot Notation

- **objectName.key**
- For example,

```
const person = {
    name: 'John',
    age: 20,
};
// accessing property
console.log(person.name); // John
```

2. Using bracket Notation

- objectName["propertyName"]
- For example,

```
const person = {
    name: 'John',
    age: 20,
};
// accessing property
console.log(person["name"]); // John
```

- **JavaScript Nested Objects**

```
// nested object
const student = {
    name: 'John',
    age: 20,
    marks: {
        science: 70,
        math: 75
    }
}

// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}

// accessing property of marks object
console.log(student.marks.science); // 70
```

- In the above example, an object **student** contains an object value in the **marks** property.

- **JavaScript Object Methods**

```
const person = {
    name: 'Sam',
    age: 30,
    // using function as a value
    greet: function() { console.log('hello') }
}

person.greet(); // hello
```

- Here, a function is used as a value for the greet key.
- That's why we need to use person.greet() instead of person.greet to call the function inside the object.