CONDITIONAL STATEMENTS

What is a Conditional Statement?

Conditional statements perform different actions based on different conditions. The primary conditional statements in JavaScript are if, else if, else, and switch.

JavaScript if-statement

It is a conditional statement used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statements is executed otherwise not.

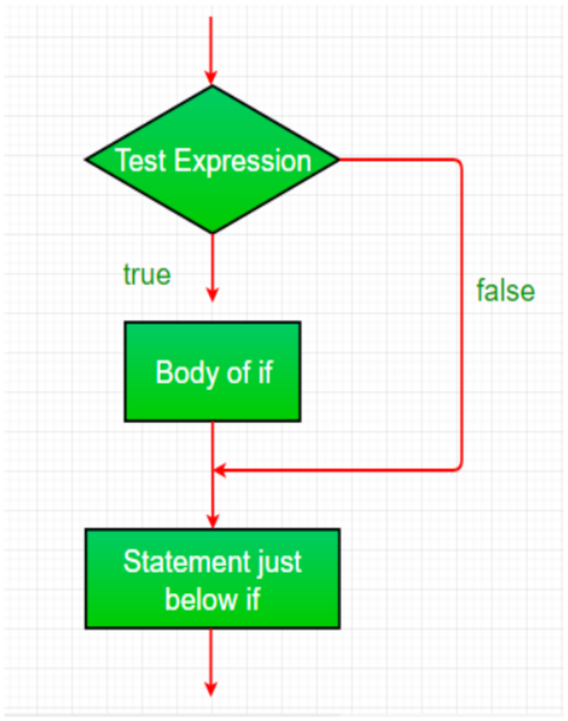
Syntax:

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

The if statement accepts boolean values – if the value is true then it will execute the block of statements under it. If we do not provide the curly braces '{' and '}' after **if(condition)** then by default if statement considers the immediate one statement to be inside its block. For example,

```
if(condition)
   statement1;
   statement2;
// Here if the condition is true, if block
// will consider only statement1 to be inside
```

Flow chart:



Example: Here is a simple example demonstrating the **if** statement.

```
// JavaScript program to illustrate If statement
let age = 19;
if (age > 18)
  console.log("Congratulations, You are eligible to drive");
```

Output:

```
Congratulations, You are eligible to drive
```

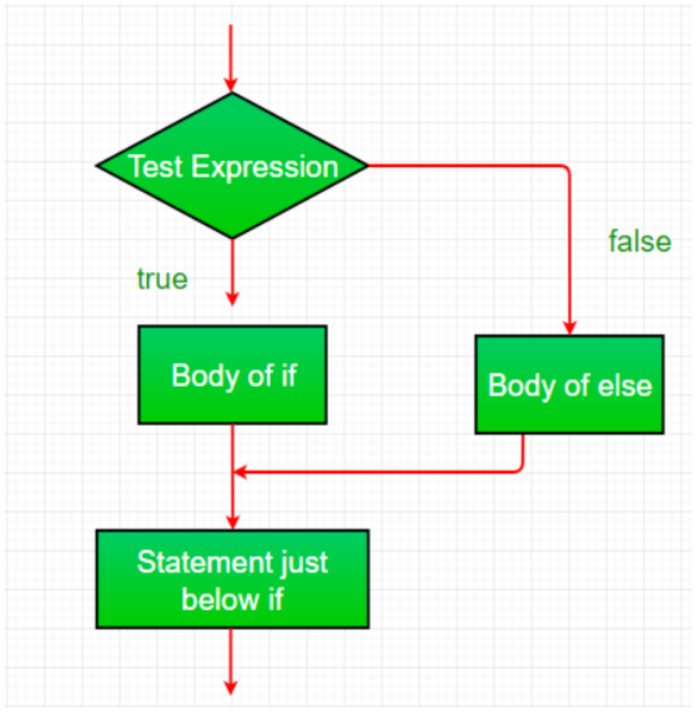
JavaScript if-else statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false? Here comes the else statement. We can use the else statement with the if statement to execute a block of code when the condition is false

Syntax:

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

Flow chart:



Example: This example describes the if-else statement in Javascript.

```
let i = 10;
if (i < 15)
    console.log("i is less than 15");
else
    console.log("I am Not in if");
```

Output

```
i is less than 15
```

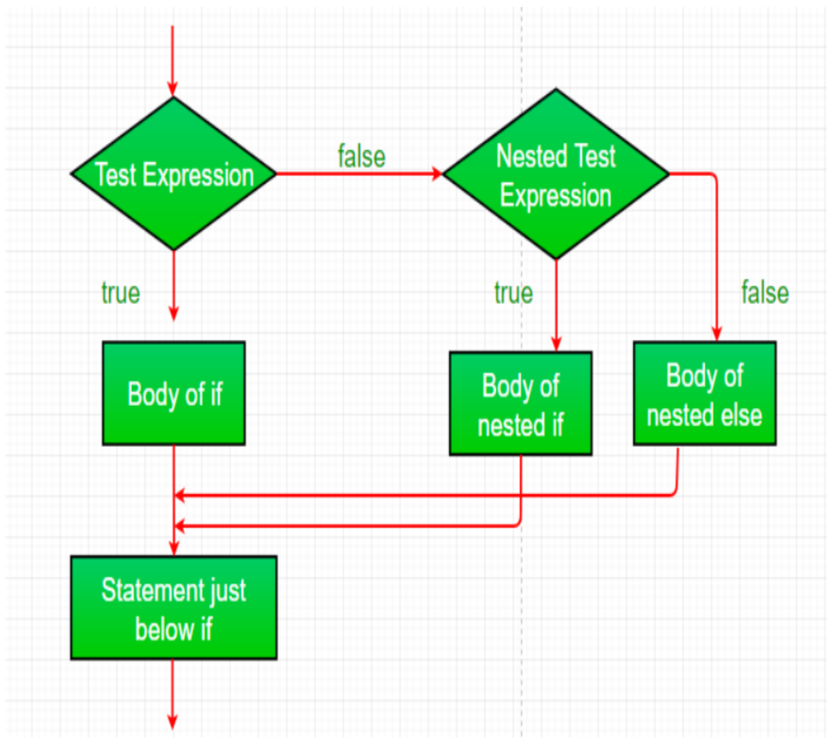
JavaScript nested-if statement

JavaScript allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement. A nested if is an if statement that is the target of another if or else.

Syntax:

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
```

Flow chart:



Example: This example describes the nested-if statement in Javascript.

```
// JavaScript program to illustrate nested-if statement
let i = 10;
if (i == 10) {  // First if statement
   if (i < 15) {
      console.log("i is smaller than 15");
      // Nested - if statement
      // Will only be executed if statement above
      // it is true
      if (i < 12)
         console.log("i is smaller than 12 too");
      else
         console.log("i is greater than 15");
   }
}
```

Output

```
i is smaller than 15

i is smaller than 12 too
```

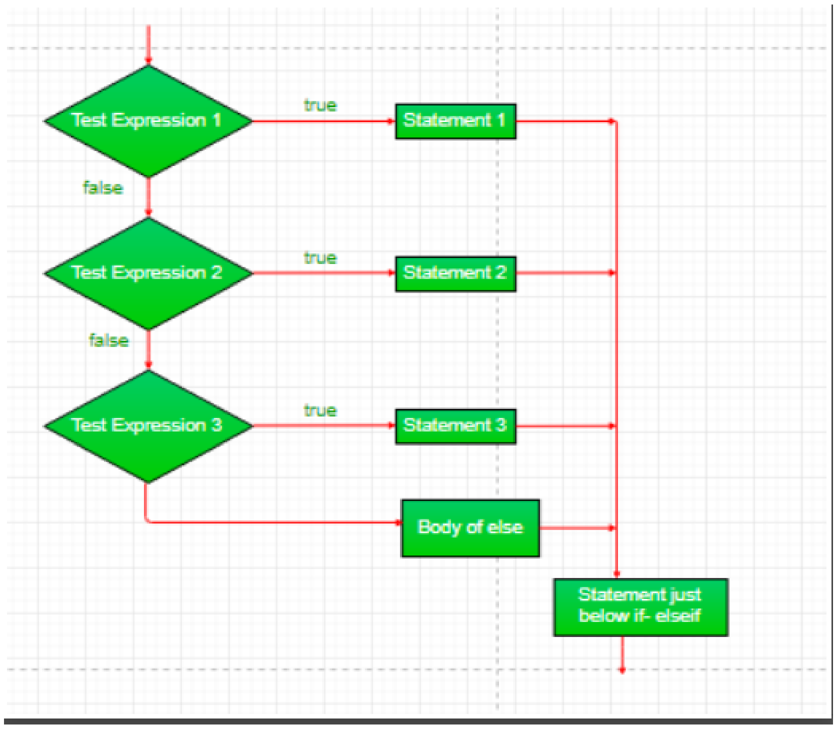
JavaScript if-else-if ladder statement

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition)

    statement;

else if (condition)

    statement;

.

.

else

    statement;
```

Flow chart:



Example: This example describes the if-else-if ladder statement in Javascript.

```
// JavaScript program to illustrate nested-if statement
let i = 20;


if (i == 10)
   console.log("i is 10");
else if (i == 15)
   console.log("i is 15");
else if (i == 20)
   console.log("i is 20");
else
   console.log("i is not present");
```

Output

```
i is 20
```

Example Questions:

1. A company decided to give bonus of 5% to employee if his/her year of service is more than 5 years. Ask user for their salary and year of service and print the net bonus amount.
2. Write a program to calculate the electricity bill (accept number of unit from user) according to the following criteria :

	Unit	Price
a.	First 100 units	no charge
b.	Next 100 units	Rs 5 per unit
c.	After 200 units	Rs 10 per unit

   (For example if input unit is 350 than total bill amount is Rs2000)
3. Write a program to accept a number from 1 to 7 and display the name of the day like 1 for Sunday , 2 for Monday and so on.
4. . Accept any city from the user and display monument of that city.

	City	Monument
a.	Delhi	Red Fort
b.	Agra	Taj Mahal
c.	Jaipur	Jal Mahal

5. Write a program to check whether the last digit of a number(entered by user) is divisible by 3 or not.
6. Write a program to accept the cost price of a bike and display the road tax to be paid according to the following criteria :

	Cost price (in Rs)	Tax
a.	> 100000	15 %
b.	> 50000 and <= 100000	10%
c.	<= 50000	5%

7. Write a program that takes a number as input and checks whether it is:
   a. "Even and divisible by 4"
   b. "Odd and greater than 10"
   c. "Neither of the above"
8. Write a program that checks the total purchase amount and applies discounts:
   a. If the amount is greater than $100, log "You get a 20% discount!".
   b. If the amount is between $50 and $100 (inclusive), log "You get a 10% discount!".
   c. If the amount is less than $50, log "No discount available."
9. Write a program to calculate the monthly charge for a mobile data plan based on data usage:

	Data Usage	Charge
a.	Up to 2Gb	Rs 200
b.	2-10Gb	Rs 200 + Rs 50 per Gb over 2 Gb
c.	Above 10 Gb	Rs 600 + Rs 50 per Gb over 10 Gb

   Example:If the data usage is 15 GB, calculate the total charge.

10. Write a program to calculate the parking fee based on hours parked:

	Hours Parked	Charge
a.	Upto 1 hour	Rs 20
b.	2-5 Hour	Rs 20 + Rs 10 per Hour over 1 hour
c.	Above 5 Hour	Rs 60 + Rs 15 per Hour over 5 Hours

Example: If the parking duration is 8 hours, calculate the total fee

JavaScript switch Statement

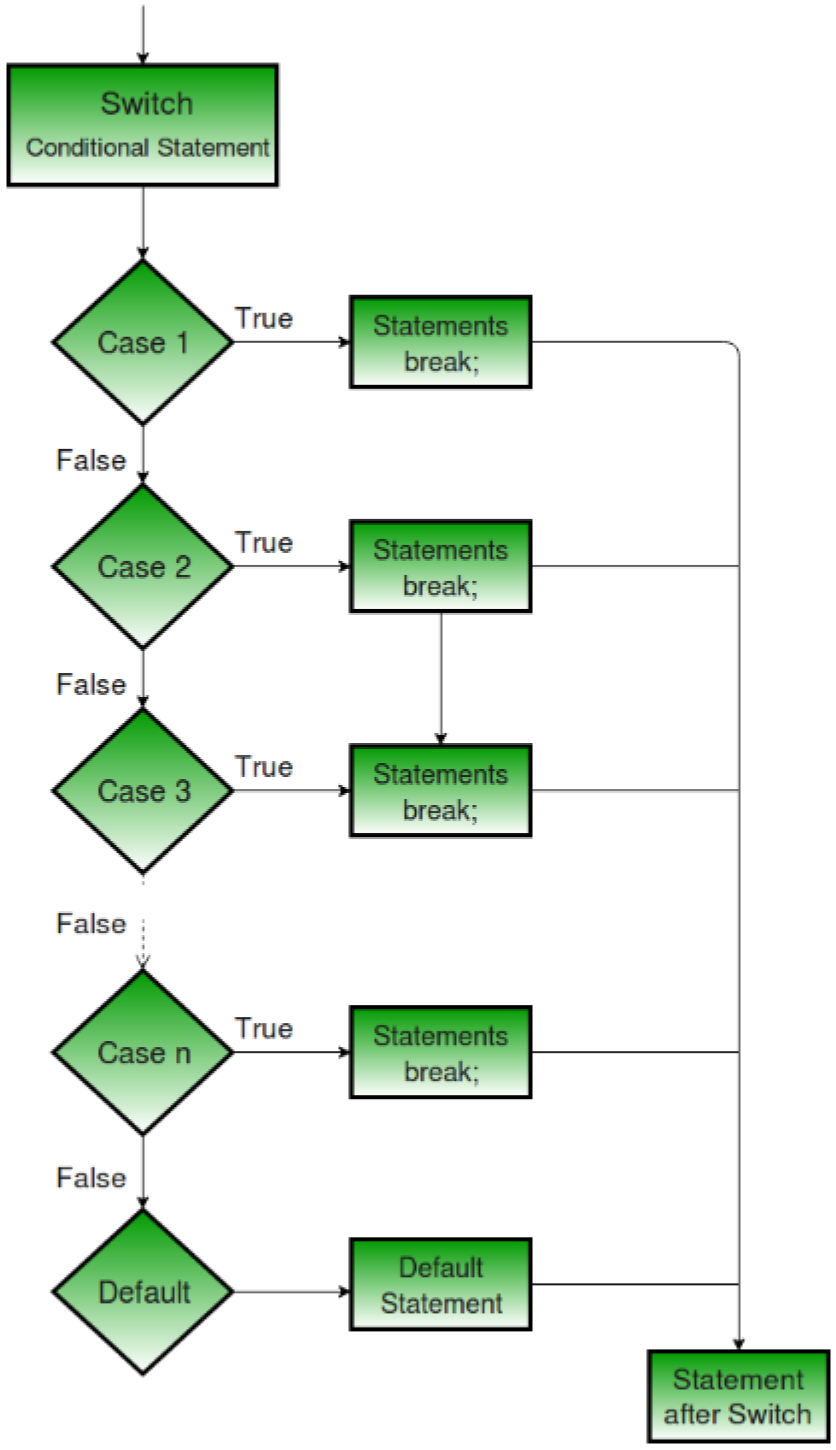
The **JavaScript switch statement** evaluates an expression and executes a block of code based on matching cases. It provides an alternative to long if-else chains, improving readability and maintainability, especially when handling multiple conditional branches.

Syntax:

```
switch (expression) {
    case value1:
        // code block 1;
        break;
    case value2:
         // code block 2;
        break;
   ...
    default:
       // default code block;
}
```

- **Expression** is the value that you want to compare.
- Case value1, case value2, etc., represent the possible values of the expression.
- break statement terminates the switch statement. Without it, execution will continue into the next case.
- Default specifies the code to run if none of the cases match the expression.

Flowchart:


Switch
Conditional Statement
Case 1
True
Statements
break;
False
Case 2
True
Statements
break;
False
Case 3
True
Statements
break;
False
Case n
True
Statements
break;
False
Default
Default
Statement
Statement
after Switch

Example: Here, we will print the day name on day 3.

```
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;
  case 6:
    dayName = "Saturday";
    break;
  case 7:
    dayName = "Sunday";
    break;
  default:
    dayName = "Invalid day";
}
console.log(dayName); // Output: Wednesday
```

Output:

```
Wednesday
```

How Switch Statement Works

- **Evaluation**: The expression inside the switch the statement is evaluated once.
- **Comparison**: The value of the expression is compared with each case label (using strict equality ===).
- **Execution**: If a match is found, the corresponding code block following the matching case the label is executed. If no match is found, the execution jumps to the default case (if present) or continues with the next statement after the switch block.
- **Break Statement**: After executing a code block, the break statement terminates the switch statement, preventing execution from falling through to subsequent cases. If break is omitted, execution will continue to the next case (known as “fall-through”).
- **Default Case**: The default case is optional. If no match is found, the code block under default is executed.

Default Keyword

The default the keyword is used within a switch statement as a fallback option when none of the case expressions match the value being evaluated. It acts similarly to the else statement in an if...else chain, providing a default action to take when no other specific cases match.

Example Questions:

1. Write a program that takes a number (1–7) as input and logs the day of the week:
    - 1: Monday
    - 2: Tuesday
    - 3: Wednesday
    - 4: Thursday
    - 5: Friday
    - 6: Saturday
    - 7:Sunday

      If the input is outside this range, log "Invalid input."
2. Write a program that takes a month number (1–12) and logs the season:
    - December, January, February: "Winter"
    - March, April, May: "Spring"
    - June, July, August: "Summer"
    - September, October, November: "Autumn"

3. Write a program that takes a grade (A, B, C, D, or F) and logs:
    - "Excellent" for A
    - "Good" for B
    - "Average" for C
    - "Poor" for D
    - "Fail" for F

    If the grade is not one of these letters, log "Invalid grade."
4. Write a program that takes two numbers and an operator (+, -, *, /) as input and performs the corresponding operation using a switch statement.
    Example:
    If the input is 10, 20, and "+", log "30".
5. Write a program that takes a browser name (Chrome, Firefox, Safari, Edge, Opera) as input and logs a message:
    - Chrome: "Google's browser"
    - Firefox: "Mozilla's browser"
    - Safari: "Apple's browser"
    - Edge: "Microsoft's browser"
    - Opera: "A popular browser"

    For any other input, log "Browser not recognized."

JavaScript Loops

Loops in JavaScript are used to reduce repetitive tasks by repeatedly executing a block of code as long as a specified condition is true. This makes code more concise and efficient.

Suppose we want to print ‘Hello World’ five times. Instead of manually writing the print statement repeatedly, we can use a loop to automate the task and execute it based on the given condition.

Different types of loops available in JavaScript

1. for Loop
2. while Loop
3. do-while Loop
4. for-in Loop
5. for-of Loop

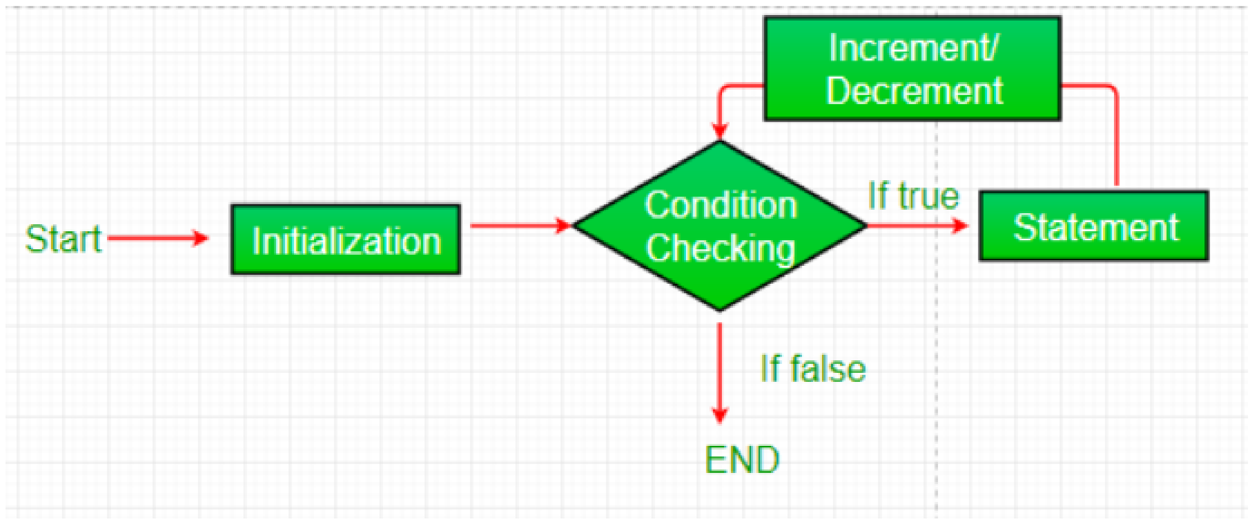
For Loop

The for loop repeats a block of code a specific number of times. It contains initialization, condition, and increment/decrement in one line.

Syntax:

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

Flowchart:



Example:

```
for (let i = 1; i <= 3; i++) {
    console.log("Count:", i);
}
```

Output:

```
Count: 1
Count: 2
Count: 3
```

In this example

- Initializes the counter variable (let i = 1).
- Tests the condition (i <= 3); runs while true.

- Executes the loop body and increments the counter (i++).

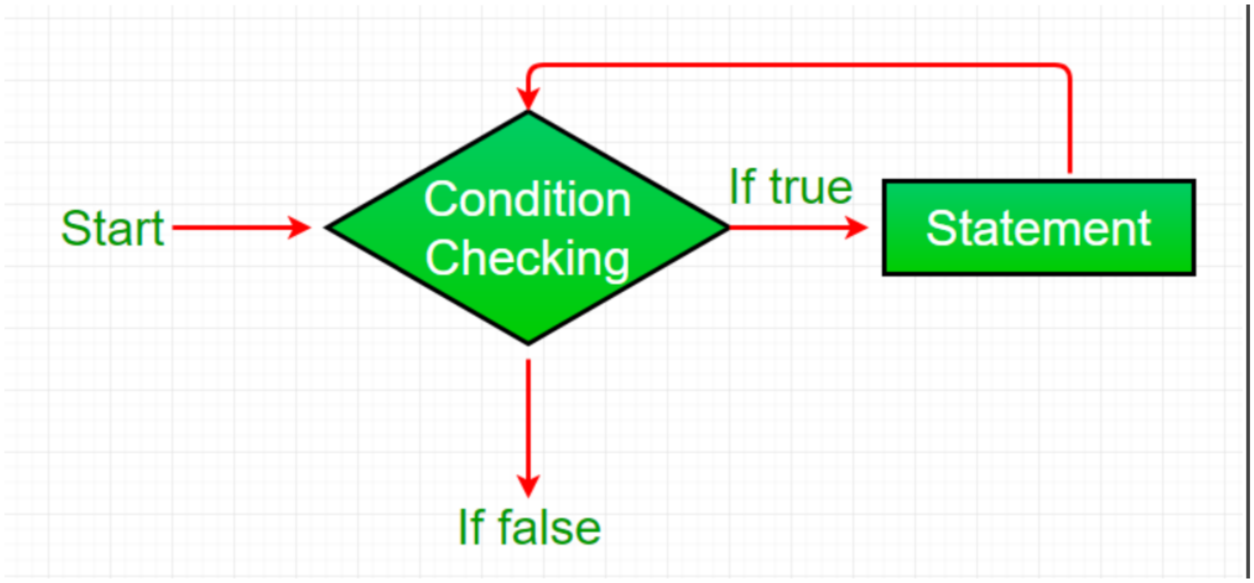
while Loop

The while loop executes as long as the condition is true. It can be thought of as a repeating if statement.

Syntax

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

Flowchart:



Example:

```
let i = 0;
while (i < 3) {
    console.log("Number:", i);
    i++;
```

Output:

```
Number: 0
Number: 1
Number: 2
```

In this example

- Initializes the variable (let i = 0).
- Runs the loop body while the condition (i < 3) is true.
- Increments the counter after each iteration (i++).

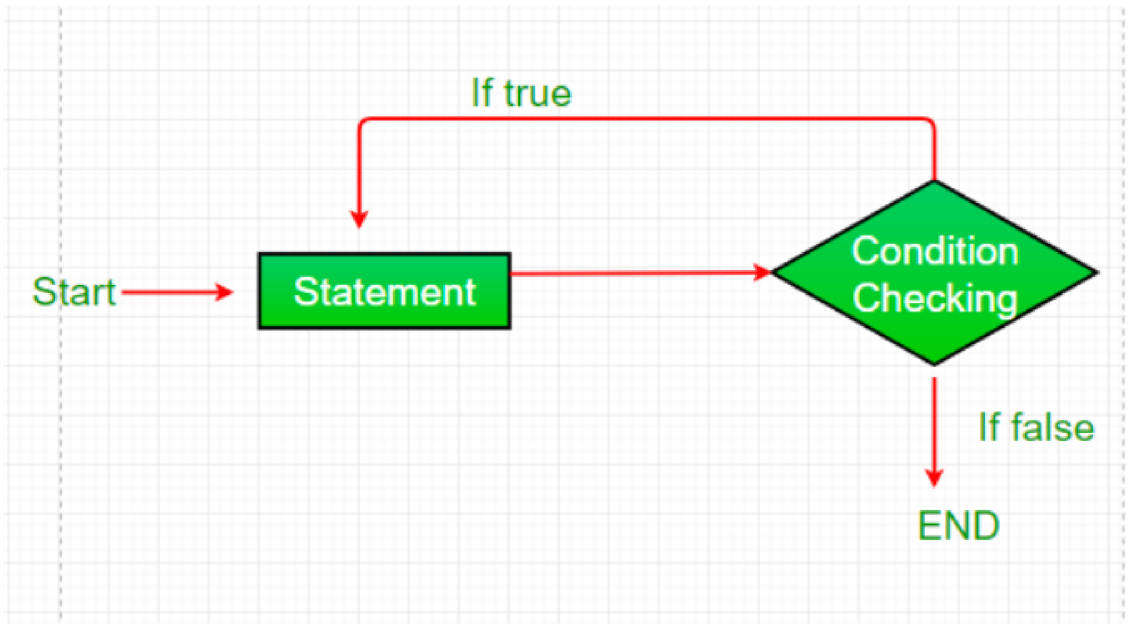
do-while Loop

The do-while loop is similar to while loop except it executes the code block at least once before checking the condition.

Syntax

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

Flowchart



Example:

```
let i = 0;

do {

    console.log("Iteration:", i);

    i++;

} while (i < 3);
```

Output:

```
Iteration: 0

Iteration: 1

Iteration: 2
```

JavaScript for-in Loop

The for…in loop is used to iterate over the properties of an object. It only iterate over keys of an object which have their enumerable property set to “true”.

Syntax:

```
for (let key in object) {
   // Code to execute
}
```

Example:

```
const obj = { name: "Ashish", age: 25 };
for (let key in obj) {
   console.log(key, ":", obj[key]);
}
```

Output:

```
name : Ashish
age : 25
```

JavaScript for-of Loop

The for…of loop is used to iterate over iterable objects like arrays, strings, or sets. It directly iterate the value and has more concise syntax than for loop.

Syntax:

```
for (let value of iterable) {
   // Code to execute
}
```

Example

```
let a = [1, 2, 3, 4, 5];
for (let val of a) {
   console.log(val);
}
```

Output:

```
1

2

3

4

5
```

Tasks:

1. Write a for loop to print numbers from 1 to 10 in the console.
2. Use a for loop to print all odd numbers between 1 and 20.
3. Given const colors = ['red', 'green', 'blue'], use a for loop to print each color.
4. Write a while loop to calculate the sum of numbers from 1 to 50.
5. Use a do...while loop to print numbers from 10 down to 1.
6. Given const numbers = [1, 2, 3, 4, 5], use a loop to print the elements in reverse order.
7. Use a for loop to calculate the total of all numbers in const items = [10, 20, 30, 40].
8. Write a loop to find the largest number in const nums = [15, 34, 7, 23, 56].
9. Given const person = { name: 'John', age: 25, city: 'New York' }, use a for...in loop to print all the keys and their values.
10. Write a loop to create a new array containing only the even numbers from const arr = [12, 7, 19, 28, 33, 40].
11. Use a nested loop to print a multiplication table for numbers 1 to 5.
12. Write a loop to create a new array without duplicates from const nums = [4, 5, 4, 6, 7, 6,8].
13. Given const nested = [[1, 2], [3, 4], [5, 6]], use a loop to flatten it into [1, 2, 3, 4, 5, 6].
14. Write a loop to create a new array without duplicates from const nums = [4, 5, 4, 6, 7, 6, 8].
15. Given const sales = { January: 100, February: 200, March: 300 }, write a loop to calculate the total sales
16.

Functions in JavaScript

Functions in JavaScript are reusable blocks of code designed to perform specific tasks. They allow you to organize, reuse, and modularize code. It can take inputs, perform actions, and return outputs.

Functions in JavaScript can be broadly categorized into two main types:

1. Built-in Functions
2. User-defined Functions

Built-in Functions

These are predefined functions provided by JavaScript to perform common tasks. They come with the language and are ready to use without needing to define them.

Math functions

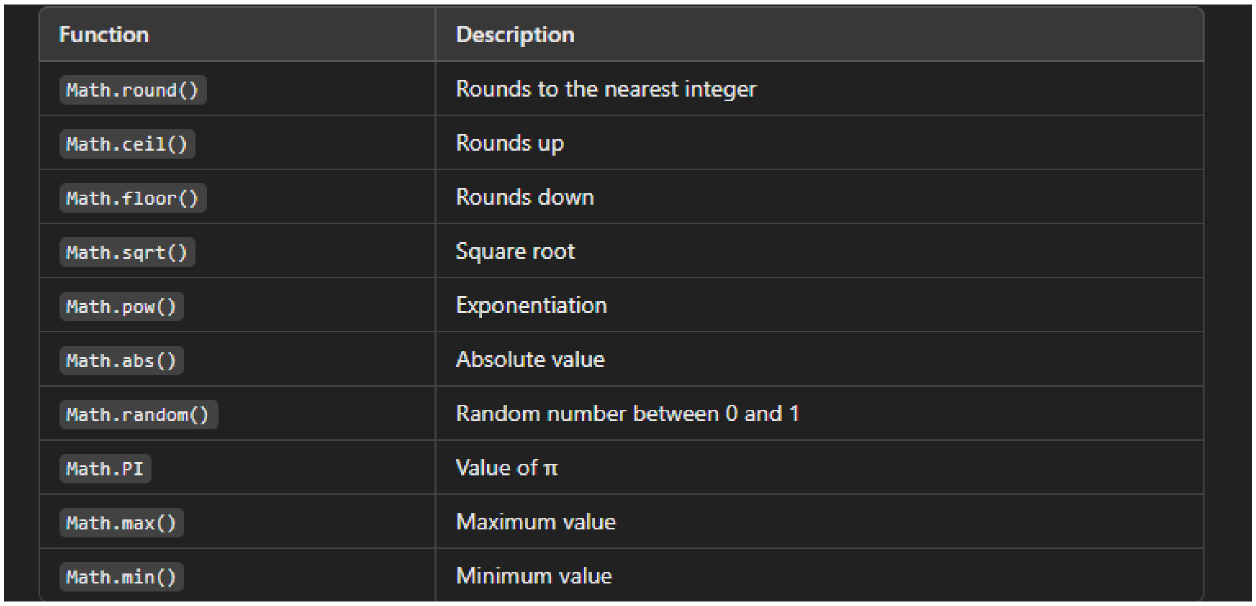
JavaScript provides a Math object with various functions to perform mathematical operations. Below are some commonly used **Math functions** with examples:

Math.max(): Returns the largest of given numbers.

```
console.log(Math.max(10, 20, 5, 30)); // Output: 30
```

Math.min(): Returns the smallest value from a list of numbers.

```
console.log(Math.min(10, 20, 5, 30)); // Output: 5
```



Function	Description
`Math.round()`	Rounds to the nearest integer
`Math.ceil()`	Rounds up
`Math.floor()`	Rounds down
`Math.sqrt()`	Square root
`Math.pow()`	Exponentiation
`Math.abs()`	Absolute value
`Math.random()`	Random number between 0 and 1
`Math.PI`	Value of π
`Math.max()`	Maximum value
`Math.min()`	Minimum value

User-defined Functions

These are functions created by the programmer to perform specific tasks. They allow you to encapsulate logic and reuse it wherever needed.

Different ways of writing functions in JavaScript

JavaScript provides different ways to define functions, each with its own syntax and use case

1. Function Declaration
2. Anonymous function
3. Arrow Function

Function Declaration

A Function Declaration is the traditional way to define a function. It starts with the function keyword, followed by the function name and any parameters the function needs.

Syntax:

```
function name(arguments)

{

  javascript statements

}
```

A Function definition or function statement starts with the function keyword and continues with the following.

- Function's name.
- A list of function arguments contained in parenthesis and separated by commas.
- Statements are enclosed in curly brackets.

Example:

```
function welcome() {

   console.log("welcome to Js");

}


// Function calling

welcome();
```

Output:

```
welcome to Js
```

JavaScript Function Parameters

Function parameters are variables defined in the function declaration that receive values (arguments) when the function is called. They play a key role in making functions reusable and dynamic.

- Values are assigned to parameters in the order they are passed.
- You can assign default values to parameters if no arguments are provided.
- Allows capturing an indefinite number of arguments into an array.
- Primitive types are passed by value, whereas objects are passed by reference.

Example:

```
function greet(name) {
   console.log(`Hello, ${name}!`);
}
greet("Meeta")
```

Output:

```
Hello, Meeta!
```

- **Parameter:** name in the function definition.
- **Argument:** “Meeta” passed when calling the function.

Types of Parameters in JavaScript

1. **Required Parameters**
   These are the basic parameters expected by the function. If not provided, they will be undefined.

   Example:

```
function add(a, b) {
   return a + b;
}
console.log(add(5, 3));
console.log(add(5));
// Output
8
NaN
```

2. **Default Parameters**
   Introduced in ES6, default parameters allow you to assign a default value to a parameter if no argument is passed or if the argument is undefined.

   Example:

```
function mul(a, b = 1) {
   return a * b;
}
console.log(mul(5));
console.log(mul(5, 2));
// Output
5
10
```

3. **Rest Parameters**
   Rest parameters allow a function to accept an indefinite number of arguments as an array. Use the … syntax to capture all additional arguments

   Example:

```
function sum(...numbers) {
   return numbers.reduce((total, num) => total + num, 0);
}
console.log(sum(1, 2, 3, 4));

// Output
10
```

4. **Destructured Parameters**

   You can destructure arrays or objects passed as arguments into individual variables.

   Example :

```
function displayUser({ name, age }) {
   return `${name} is ${age} years old.`;
}
const user = { name: "Meeta", age: 25 };
console.log(displayUser(user));
// Output
Meeta is 25 years old.
```

5. **Passing Functions as Parameters (Higher-Order Functions)**
   Functions in JavaScript can accept other functions as parameters, making it easy to create reusable code.

   Example:

```
function executeTask(task, callback) {
   console.log(`Task: ${task}`);
   callback();
}
executeTask("Clean the room", () => {
   console.log("Task Completed!");
});

// Output
Task: Clean the room
Task Completed!
```

Anonymous Functions:

An anonymous function is simply a function that does **not have a name**. Unlike named functions, which are declared with a name for easy reference, anonymous functions are usually created for specific tasks and are often assigned to variables or used as arguments for other functions.

In JavaScript, you normally use the **function keyword followed by a name** to declare a function. However, in an anonymous function, the name is **omitted**. These functions are often used in situations where you don't need to reuse the function outside its immediate context.

Syntax:

```
function() {
   // Function Body
 }
```

Example :

```
const greet = function () {

    console.log("Welcome to anonymous function!");

};

greet();  // output - Welcome to anonymous function!
```

Arrow function:

An arrow function is a shorter syntax for writing functions in JavaScript. Introduced in ES6, arrow functions allow for a more concise and readable code, especially in cases of small functions. Unlike regular functions, arrow functions don't have their own this, but instead, inherit it from the surrounding context.

- Arrow functions are written with the => symbol, which makes them compact.
- They don't have their own this. They inherit this from the surrounding context.
- For functions with a single expression, the return is implicit, making the code more concise.
- Arrow functions do not have access to the arguments object, which is available in regular functions.

Syntax:

```
let function_name = (argument1, argument2 ,..) => expression
```

Example:

```
const add = (a, b) => a + b;

console.log(add(5, 3)); //output  8
```

Return In Javascript:

The return statement in JavaScript is used to exit from a function and optionally send a value back to the function caller. Once a return statement is executed, the function stops running further code.

Key Points:

1. **Exit Function:** It terminates the execution of the current function.
2. **Return Value:** It allows the function to return a result to the caller.

If no return statement is provided, the function will return undefined by default.

Example:

```
function add(a, b) {

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

 }

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

Recursive Function :

A **recursive function** is a function that calls itself during its execution. It is commonly used to solve problems that can be broken down into smaller, similar subproblems, such as calculating factorials, Fibonacci sequences, or traversing data structures like trees.

Tasks :

1. Write a function addNumbers(a, b) that takes two numbers as parameters and returns their sum.
2. Write a function isEven(num) that takes a number as a parameter and returns true if the number is even and false if it is odd.
3. Create a function celsiusToFahrenheit(celsius) that converts Celsius to Fahrenheit using the formula:
   $F=(C\times9/5)+32$
4. Write a function greet(name) that takes a name as input and returns a greeting message.
5. Write a function findMax(a, b) that takes two numbers and returns the larger of the two.
6. Write a function factorial(n) that takes a number as input and returns its factorial.
7. Write a function countVowels(str) that takes a string and returns the number of vowels in it.
8. Create a function isPalindrome(str) that checks if a string is a palindrome (reads the same backward as forward).
9. Write a function sumArray(arr) that takes an array of numbers and returns the sum of all elements.
10. Write a function hasProperty(obj, prop) that checks if an object has a specific property and returns true or false.