

## UNIT 3 – JAVASCRIPT INTERACTIVITY

### 1. Introduction to Function

- A function is a block of code that performs a specific task.
- Suppose you need to create a program to create a circle and color it. You can create two functions to solve this problem:
  - a function to draw the circle
  - a function to color the circle
- Dividing a complex problem into smaller chunks makes your program easy to understand and reusable.

### 2. Working with Function

#### ➤ 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");  
}
```

### 3. Calling 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!");
}

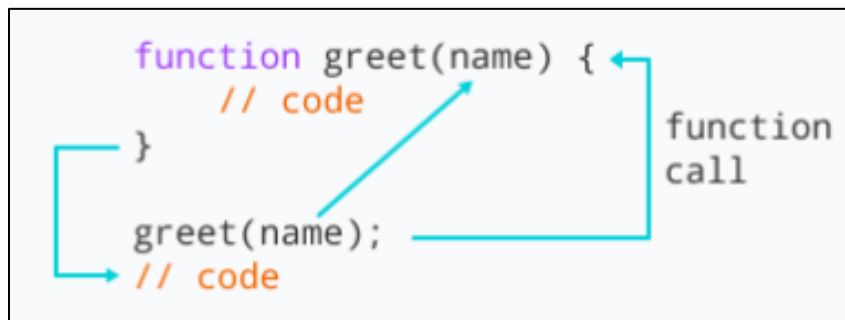
// calling the function
greet();
```

### Output

```
Hello there!
```

### ➤ 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);
```

## Output

```
Enter a name: Simon
Hello Simon :)
```

- In the above program, the `greet` function is declared with a `name` parameter. The user is prompted to enter a `name`. Then when the function is called, an argument is passed into the function.

## Example : Add Two Numbers

```
// program to add two numbers using a function
// declaring a function
function add(a, b) {
    console.log(a + b);
}

// calling functions
add(3,4);
add(2,9);
```

## Output

```
7
11
```

In the above program, the `add` function is used to find the sum of two numbers.

- The function is declared with two parameters `a` and `b`.
- The function is called using its name and passing two arguments `3` and `4` in one and `2` and `9` in another.

## ➤ Function Return

- The `return` statement can be used to return the value to a function call.
- The `return` statement denotes that the function has ended. Any code after `return` is not executed.
- If nothing is returned, the function returns an `undefined` value.

```
function add(num1, num2) {
    // code
    return result;
}

let x = add(a, b);
// code
```

function call

### Example : Sum of Two Numbers

```
// program to add two numbers
// declaring a function
function add(a, b) {
    return a + b;
}

// take input from the user
let number1 = parseFloat(prompt("Enter first number: "));
let number2 = parseFloat(prompt("Enter second number: "));

// calling function
let result = add(number1,number2);

// display the result
console.log("The sum is " + result);
```

### Output

```
Enter first number: 3.4
Enter second number: 4
The sum is 7.4
```

In the above program, the sum of the numbers is returned by the function using the `return` statement. And that value is stored in the `result` variable.

#### ➤ Benefits of Using a Function

- Function makes the code reusable. You can declare it once and use it multiple times.
- Function makes the program easier as each small task is divided into a function.
- Function increases readability.

## 4. Built-in String function

- In JavaScript, strings are used to represent and work with a sequence of characters.
- A string can represent an object as well as the primitive data type.
- JavaScript automatically converts primitive strings to String objects so that it's possible to use String methods and access properties even for primitive strings.
- In this reference page, you will find all String methods and properties available in JavaScript.
- For example, the `toUpperCase()` method returns the string converted to uppercase.

### 1. Javascript String toUpperCase()

The `toUpperCase()` method returns the string converted to uppercase.

### Example

```
const message = "javascript is fun";

// convert message to uppercase
const upperMessage = message.toUpperCase();
console.log(upperMessage);

// Output: JAVASCRIPT IS FUN
```

- **toUpperCase() Syntax**

The syntax of the `toUpperCase()` method is:

```
str.toUpperCase()
```

- **toUpperCase() Parameters**

The `toUpperCase()` method does not take in any parameters.

## 2. Javascript String toLowerCase()

- The `toLowerCase()` method returns the string converted to lowercase.

### Example

```
const message = "JAVASCRIPT IS FUN";

// convert message to lowercase
const lowerMessage = message.toLowerCase();
console.log(lowerMessage);

// Output: javascript is fun
```

- **toLowerCase() Syntax**

The syntax of the `toLowerCase()` method is:

```
str.toLowerCase()
```

- **toLowerCase() Parameters**

The `toLowerCase()` method does not take in any parameters.

## 3. JavaScript String charAt()

- The `charAt()` method returns the character at the specified index in a string.

### Example

```
// string declaration
const string = "Hello World!";

// finding character at index 1
let index1 = string.charAt(1);

console.log("Character at index 1 is " + index1);

// Output:
// Character at index 1 is e
```

- **charAt() Syntax**

The syntax of the `charAt()` method is:

```
str.charAt(index)
```

- **charAt() Parameters**

The charAt() method takes in :

- **index** - An integer between **0** and **str.length - 1**. If index cannot be converted to integer or is not provided, the default value **0** is used.

#### 4. Javascript String includes()

- The includes() method checks if one string can be found inside another string.

##### Example

```
const message = "JavaScript is fun";

// check if message includes the string "Java"
let result = message.includes("Java");
console.log(result);

// Output: true
```

- **includes() Syntax**

The syntax of the includes() method is:

```
str.includes(searchString, position)
```

- **includes() Parameters**

The includes() method takes in:

- **searchString** - A string to be searched for within str.
- **position (optional)** - The position within str to begin searching for searchString. By default, it is **0**.

#### 5. Javascript String search()

- The search() method searches for a match between a given string and a regular expression.

##### Example

```
let sentence= "I love JavaScript.";

// pattern that searches the first occurrence of an uppercase character
let regExp = /[A-Z]/;

// searching for a match between regExp and given string
let indexReg = sentence.search(regExp);

console.log(indexReg);

// Output: 0
```

- **search() Syntax**

The syntax of the search() method is:

```
str.search(regex)
```

- **search() Parameters**

The search() method takes a **single** parameter:

- **RegExp** - A regular expression object (Argument is implicitly converted to RegExp if it is a non-RegExp object)

## 6. JavaScript String concat()

The concat() method concatenates given arguments to the given string.

### Example

```
let emptyString = "";

// joint arguments string
let joinedString = emptyString.concat("JavaScript", " is", " fun.");
console.log(joinedString);

// Output: JavaScript is fun.
```

- **concat() Syntax**

The syntax of the concat() method is:

```
str.concat(str1, ..., strN)
```

- **concat() Parameters**

The concat() method takes in an arbitrary number of strings to concatenate to str.

## 7. JavaScript String replace()

- The replace() method returns a new string with the specified string/regex replaced.

### Example

```
const message = "ball bat";

// replace the first b with c
let result = message.replace('b', 'c');
console.log(result);

// Output: call bat
```

- **replace() Syntax**

The syntax of replace() is:

```
str.replace(pattern, replacement)
```

- **replace() Parameter**

The replace() method takes in:

- **pattern** - either a string or a regex that is to be replaced
- **replacement** - the pattern is replaced with this replacement (can be either a string or a function)

## 8. Javascript String substring()

- The substring() method returns a specified part of the string between start and end indexes.

### Example

```
const message = "JavaScript is fun.";

// get the substring starting from index 0 to 10
let result = message.substring(0, 10);
console.log(result);

// Output: JavaScript
```

- **substring() Syntax**

The syntax of the substring() method is:

```
str.substring(indexStart, indexEnd)
```

- **substring() Parameters**

The substring() method takes in:

- **indexStart** - The index of the first character to start including in the returned substring.
- **indexEnd (optional)** - The index before which to stop extraction. (Exclusive) If omitted, it extracts till the end of the string.

## 9. JavaScript String split()

- The split() method divides a string into a list of substrings and returns them as an array.

### Example

```
const message = "JavaScript::is::fun";

// divides the message string at ::
let result = message.split("::");
console.log(result);

// Output: [ 'JavaScript', 'is', 'fun' ]
```

- \* **split() Syntax**

The syntax of split() is:

```
str.split(separator, limit)
```

- \* **split() Parameter**

The split() method takes in:

- **separator (optional)** - The pattern (string or regular expression) describing where each split should occur.
- **limit (optional)** - A non-negative integer limiting the number of pieces to split the given string into.



## 10. JavaScript String length

- The length property returns the number of characters in a string.

### Example

```
// defining a string
let sentence = "I love InfoGalaxy Computers.";
// returns number of characters in the sentence string
let len = sentence.length;
console.log(len);

// Output: 28
```

#### \* length Syntax

The syntax of the length property is:

```
str.length
```

Here, `str` is a string.

#### length Parameters

The `length` property does not takes any parameters.

## 11. JavaScript String replace()

- The `replace()` method returns a new string with the specified string/regex replaced.

### Example

```
const message = "ball bat";

// replace the first b with c
let result = message.replace('b', 'c');
console.log(result);

// Output: call bat
```

#### \* replace() Syntax

The syntax of `replace()` is:

```
str.replace(pattern, replacement)
```

#### \* replace() Parameter

The `replace()` method takes in:

- **pattern** - either a string or a regex that is to be replaced
- **replacement** - the pattern is replaced with this replacement (can be either a string or a function)

```

<html>
  <head>
    <script>
      let name1 = "InfoGalaxy";
      document.write("Name1="+name1);
      document.write("<br>Uppercase Name="+name1.toUpperCase());
      document.write("<br>Lowercase Name="+name1.toLowerCase());
      document.write("<br>Char at position 6="+name1.charAt(6));
      document.write("<br>Is name Include Info="+name1.includes("info"));
      document.write("<br>Search for Galaxy="+name1.search("Galaxy"));

      name1 = name1.concat(' ', "Computers");
      document.write("<br>Name1="+name1);

      name1 = name1.replace("Galaxy", "Planet");
      document.write("<br>Name1="+name1);

      let name2 = name1.substring(4,10);
      document.write("<br>Name2="+name2);

      let name3 = name1.split(' ');
      document.write("<br>Name3="+name3);
    </script>
  </head>
  <body>
  </body>
</html>

```

---

Name1=InfoGalaxy  
 Uppercase Name=INFOGALAXY  
 Lowercase Name=infogalaxy  
 Char at position 6=l  
 Is name Include Info=false  
 Search for Galaxy=4  
 Name1=InfoGalaxy Computers  
 Name1=InfoPlanet Computers  
 Name2=Planet  
 Name3=InfoPlanet,Computers

## Output

## 5. Condition Checking-if-else statement

- In computer programming, there may arise situations where you have to run a block of code among more than one alternatives.
- For example, assigning grades A, B or C based on marks obtained by a student.
- In such situations, you can use the JavaScript if...else statement to create a program that can make decisions.
- In JavaScript, there are three forms of the if...else statement.

1. **if** statement
2. **if...else** statement
3. **if...else if...else** statement

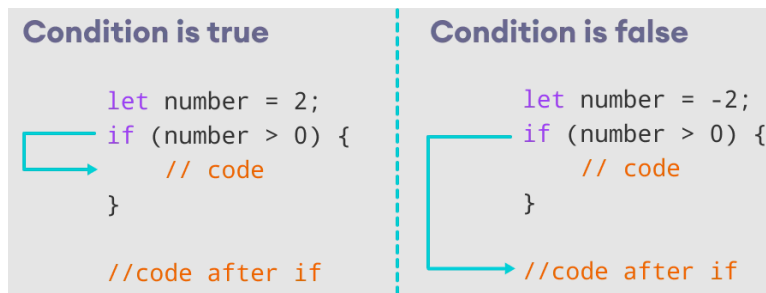
## 5.1 JavaScript if Statement

The syntax of the **if** statement is:

```
if (condition) {
    // the body of if
}
```

The if statement evaluates the condition inside the parenthesis ().

1. If the condition is evaluated to true, the code inside the body of if is executed.
2. If the condition is evaluated to false, the code inside the body of if is skipped.



### Example : if Statement

```
// check if the number is positive
const number = prompt("Enter a number: ");

// check if number is greater than 0
if (number > 0)
{
    // the body of the if statement
    console.log("The number is positive");
}
console.log("The if statement is easy");
```

### Output

```
Enter a number: 2
The number is positive
The if statement is easy
```

## 5.2 JavaScript if...else statement

An **if** statement can have an optional **else** clause. The syntax of the **if...else** statement is:

```
if (condition) {
    // block of code if condition is true
} else {
    // block of code if condition is false
}
```

The **if...else** statement evaluates the **condition** inside the parenthesis.

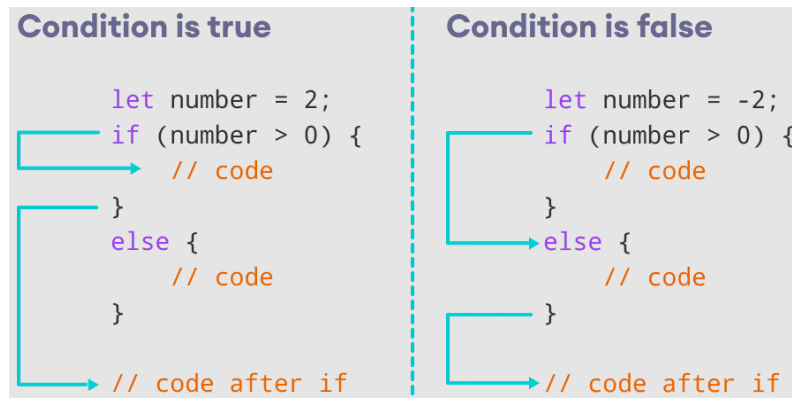
If the condition is evaluated to true,

1. the code inside the body of if is executed

- the code inside the body of else is skipped from execution

If the condition is evaluated to false,

- the code inside the body of else is executed
- the code inside the body of if is skipped from execution



### Example: if...else Statement

```
// check if the number is positive or negative/zero
const number = prompt("Enter a number: ");
// check if number is greater than 0
if (number > 0) {
  console.log("The number is positive");
}
// if number is not greater than 0
else {
  console.log("The number is either a negative number or 0");
}
console.log("The if...else statement is easy");
```

### Output

```
Enter a number: 2
The number is positive
The if...else statement is easy
```

### 5.3 JavaScript if...else if statement

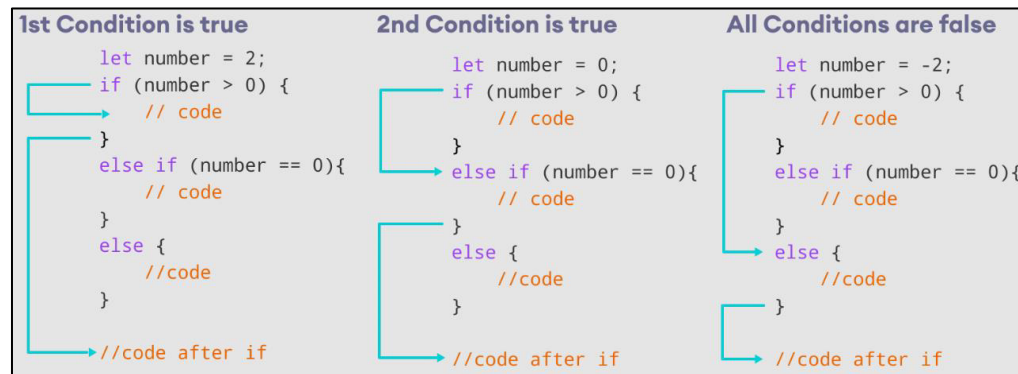
The `if...else` statement is used to execute a block of code among two alternatives. However, if you need to make a choice between more than two alternatives, `if...else if...else` can be used.

The syntax of the `if...else if...else` statement is:

```
if (condition1) {
  // code block 1
} else if (condition2){
  // code block 2
} else {
  // code block 3
}
```

- If **condition1** evaluates to true, the **code block 1** is executed.
- If **condition1** evaluates to false, then **condition2** is evaluated.

- If the **condition2** is true, the **code block 2** is executed.
- If the **condition2** is false, the **code block 3** is executed.



### Example: if...else if Statement

```
// check if the number is positive, negative or zero
const number = prompt("Enter a number: ");

// check if number is greater than 0
if (number > 0) {
    console.log("The number is positive");
}
// check if number is 0
else if (number == 0) {
    console.log("The number is 0");
}
// if number is neither greater than 0, nor zero
else {
    console.log("The number is negative");
}
console.log("The if...else if...else statement is easy");
```

### Output

```
Enter a number: 0
The number is 0
The if...else if...else statement is easy
```

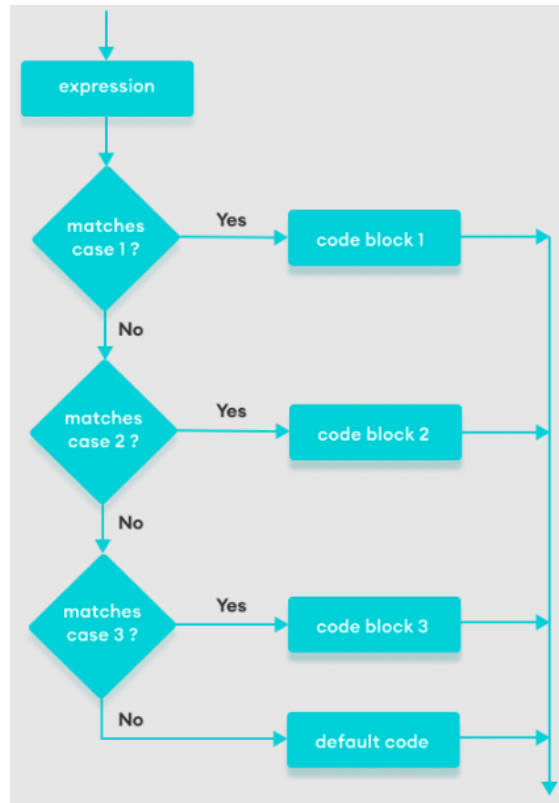
## 6. Switch Case Statement

- 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
}
```

The switch statement evaluates a variable/expression inside parentheses ().

- If the result of the expression is equal to value1, its body is executed.
- If the result of the expression is equal to value2, its body is executed.
- This process goes on. If there is no matching case, the default body executes.



### Example : Simple Program Using switch Statement

```
// program using switch statement
let a = 2;
switch (a) {
  case 1:
    a = 'one';
    break;
  case 2:
    a = 'two';
    break;
  default:
    a = 'not found';
    break;
}
console.log(`The value is ${a}`);
```

### Output

```
The value is two.
```

## 7. Looping Statements – for Loop While Loop

In programming, loops are used to repeat a block of code.

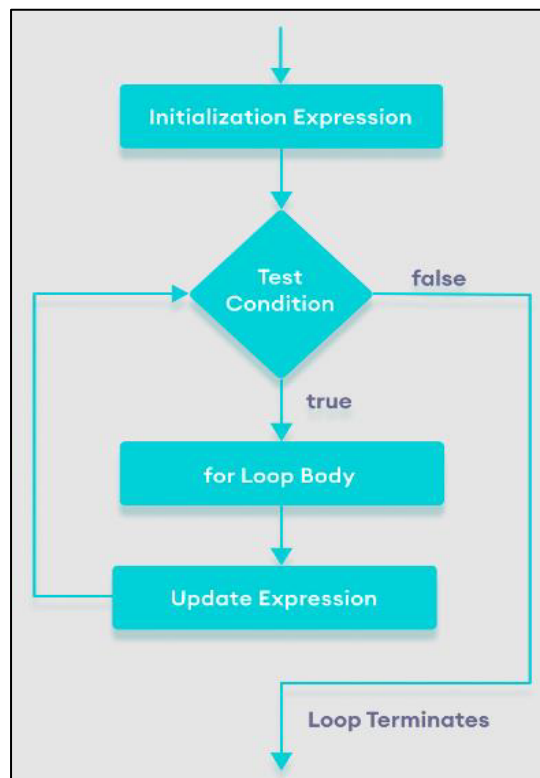
### 7.1 JavaScript for loop

- For 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  
}
```

1. The **initialExpression** initializes and/or declares variables and executes only once.
2. The **condition** is evaluated.
  - If the condition is false, the for loop is terminated.
  - If the condition is true, the block of code inside of the for loop is executed.
3. The **updateExpression** updates the value of **initialExpression** when the condition is true.
4. The **condition** is evaluated again. This process continues until the condition is false.



### Example : 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.`);
}
```

### Output

```
I love JavaScript.
I love JavaScript.
I love JavaScript.
I love JavaScript.
I love JavaScript.
```

### Example: Display Numbers from 1 to 5

```
// program to display numbers from 1 to 5
const n = 5;

// looping from i = 1 to 5
// in each iteration, i is increased by 1
for (let i = 1; i <= n; i++) {
    console.log(i);    // printing the value of i
}
```

### Output

```
1
2
3
4
5
```

## 7.2 JavaScript while Loop

While 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.

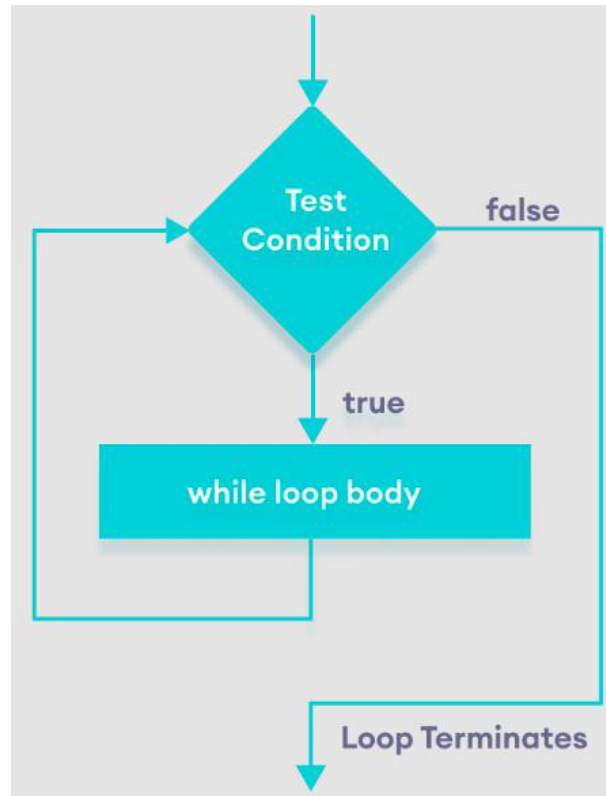
**The while syntax of the while loop is:**

```
while (condition) {
    // body of loop
}
```

1. A while loop evaluates the **condition** inside the parenthesis ().



2. If the **condition** evaluates to true, the code inside the while loop is executed.
3. The **condition** is evaluated again.
4. This process continues until the **condition** is false.
5. When the **condition** evaluates to false, the loop stops.



### Example: 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;
}
```

### Output

```
1
2
3
4
5
```

### Example: Sum of Positive Numbers Only

```
let sum = 0;

// take input from the user
let number = parseInt(prompt('Enter a number: '));

while(number >= 0) {

    // add all positive numbers
    sum += number;

    // take input again if the number is positive
    number = parseInt(prompt('Enter a number: '));
}

// display the sum
console.log(`The sum is ${sum}.`);
```

### Output

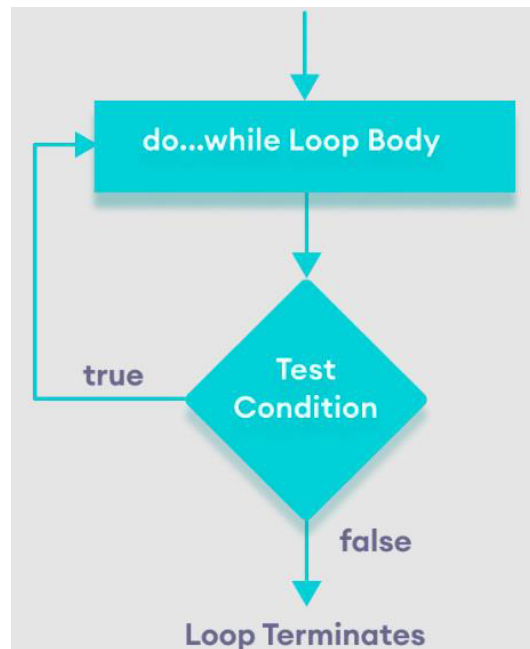
```
Enter a number: 2
Enter a number: 5
Enter a number: 7
Enter a number: 0
Enter a number: -3
The sum is 14.
```

### 7.3 JavaScript do...while Loop

The syntax of do...while loop is:

```
do {
    // body of loop
} while(condition)
```

1. The body of the loop is executed at first. Then the **condition** is evaluated.
2. If the **condition** evaluates to true, the body of the loop inside the do statement is executed again.
3. The **condition** is evaluated once again.
4. If the **condition** evaluates to true, the body of the loop inside the do statement is executed again.
5. This process continues until the **condition** evaluates to false. Then the loop stops.



### Example: 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);
```

### Output

```
1
2
3
4
5
```

### Example: Sum of Positive Numbers

```
let sum = 0;
let number = 0;

do {
  sum += number;
  number = parseInt(prompt('Enter a number: '));
} while(number >= 0)

console.log(`The sum is ${sum}.`);
```

### Output 1

```
Enter a number: 2
Enter a number: 4
Enter a number: -500
The sum is 6.
```

### Output 2

```
Enter a number: -80
The sum is 0.
```

## 8. Break and Continue

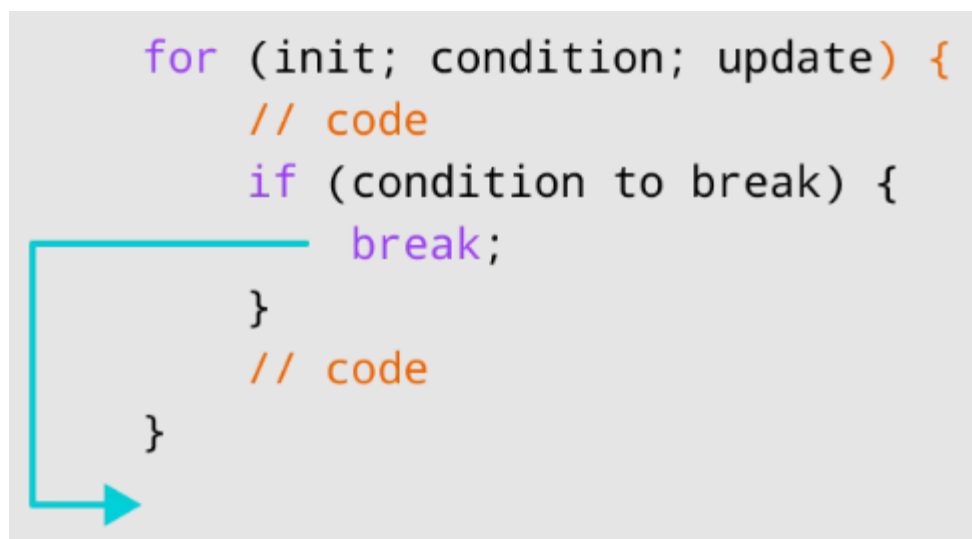
### 8.1 JavaScript break Statement

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: 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);
}
```

**Output**

```
1
2
```

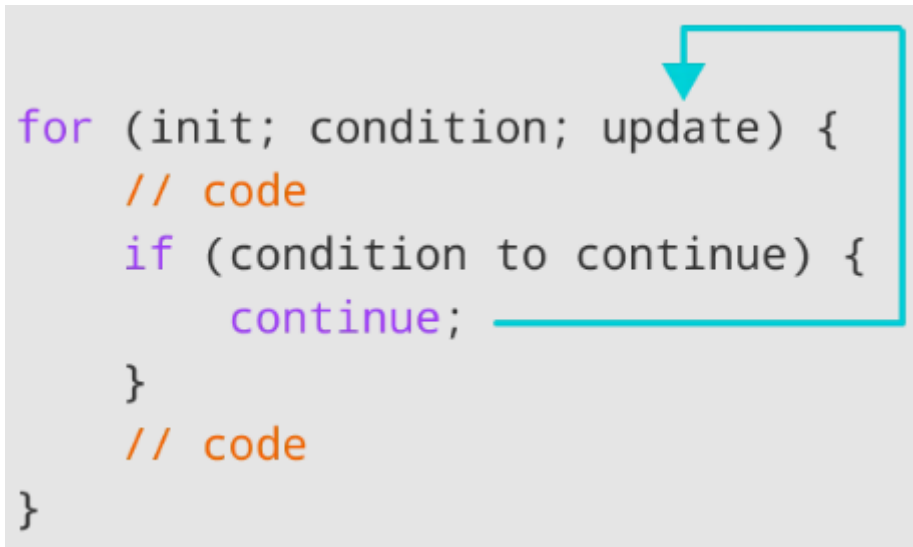
## 8.2 JavaScript continue Statement

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];
```

**Working of JavaScript continue Statement**



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

**Example: 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
```