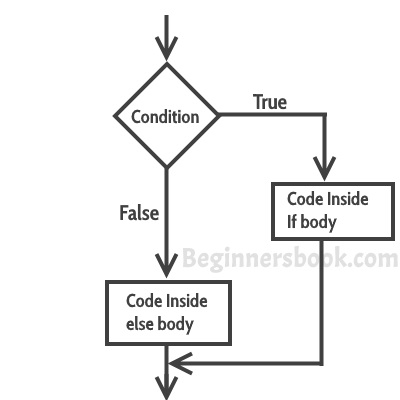
**JAVASCRIPT – If-Else**

1. **FlowChart of if-else**

****

1. **if Statement**

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

if (*condition*) {

// *block of code to be executed if the condition is true*

}

1. **if...else Statement**

Use the else statement to specify a block of code to be executed if the condition is false.

if (*condition*) {

// *block of code to be executed if the condition is true*

} else {

// *block of code to be executed if the condition is false*

}

1. **if...else if... Statement**

Use the else if statement to specify a new condition if the first condition is false.

if (*condition1*) {

// *block of code to be executed if condition1 is true*

} else if (*condition2*) {

// *block of code to be executed if the condition1 is false and condition2 is true*

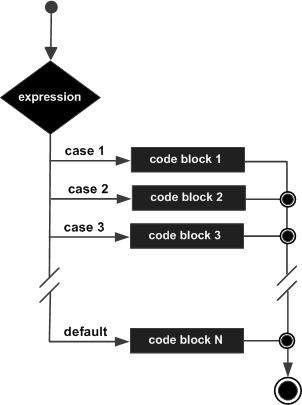
} else {

// *block of code to be executed if the condition1 is false and condition2 is false*

}

**JAVASCRIPT – Switch-Case**

1. **Flow Chart**

****

Use the switch statement to select one of many code blocks to be executed.

### Syntax

**switch(*expression*) {**

**case *x*:**

***// code block***

**break;**

**case *y*:**

***// code block***

**break;**

**default:**

**// *code block***

**}**

**JAVASCRIPT – While Loop**

1. **The While Loop**

The while loop loops through a block of code as long as a specified condition is true.

**while (*condition*) {**

***// code block to be executed***

**}**

1. **The do...while Loop**

The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

**do {**

***// code block to be executed***

**}**

**while (*condition*);**

**JAVASCRIPT – For Loop**

1. **The for Loop**

The for loop has the following syntax:

**for (*statement 1*; *statement 2*; *statement 3*) {**

**// *code block to be executed***

**}**

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

**JAVASCRIPT – For-in Loop**

The JavaScript for in statement loops through the properties of an Object:

**for (key in object) {**

**// *code block to be executed***

**}**

**JAVASCRIPT – Loop Control**

1. **The break Statement**

The break statement breaks out of a switch or a loop. In a switch, it breaks out of the switch block. This stops the execution of more code inside the switch. In a loop, it breaks out of the loop and continues executing the code after the loop (if any).

1. **The continue Statement**

The continue statement breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop.

1. **Using Labels to Control the Flow**

To control the flow in JavaScript, use labels. A label can be used with a break and continue statement to control the flow more precisely. A label is simply an identifier followed by a colon (:) that is applied to a statement or a block of code.

**JAVASCRIPT – Functions**

1. **Function Definition**

JavaScript functions are defined with the function keyword.

**function *functionName*(*parameters*) {**

**// *code to be executed***

**}**

1. **Calling a Function**

The code inside a JavaScript function will execute when "something" invokes it.

**function myFunction(a, b) {**

**return a \* b;**

**}**

**myFunction(10, 2); // Will return 20**

1. **Function Parameters**

A JavaScript function does not perform any checking on parameter values (arguments).

**function *functionName*(*parameter1, parameter2, parameter3*) {**

**// *code to be executed***

**}**

1. **The return Statement**

The return statement stops the execution of a function and returns a value.

**function myFunction() {**

**return Math.PI;**

**}**

1. **Nested Functions**

All functions have access to the global scope.

In fact, in JavaScript, all functions have access to the scope "above" them.

JavaScript supports nested functions. Nested functions have access to the scope "above" them.

In this example, the inner function plus() has access to the counter variable in the parent function:

**function add() {**

**let counter = 0;**

**function plus() {counter += 1;}**

**plus();**

**return counter;**

**}**

1. **Function () Constructor**

In JavaScript, a constructor function is used to create objects. For example,

// constructor function

function Person () {

this.name = 'John',

this.age = 23

}

// create an object

const person = new Person();

1. **Function Literals**

The syntax for a function literal is much like a function statement, except that it is used as an expression rather than a statement and no function name is required.

**<script type = "text/javascript">**

**<!--**

**var func = function(x,y) {**

**return x\*y**

**};**

**function secondFunction() {**

**var result;**

**result = func(10,20);**

**document.write ( result );**

**}**

**//-->**

**</script>**