JavaScript Operators

Arithmetic Operators

- JavaScript supports the following arithmetic operators
- Assume variable A holds 10 and variable B holds 20, then

Sr.No.	Operator & Description
1	+ (Addition) Adds two operands Ex: A + B will give 30
2	- (Subtraction) Subtracts the second operand from the first Ex: A - B will give -10
3	* (Multiplication) Multiply both operands Ex: A * B will give 200
4	/ (Division) Divide the numerator by the denominator Ex: B / A will give 2
5	% (Modulus) Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) Increases an integer value by one Ex: A++ will give 11
7	-- (Decrement) Decreases an integer value by one Ex: A-- will give 9

Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example - Arithmetic operators in JavaScript

```
<html>
  <body>

    <script type = "text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var c = "Test";
        var linebreak = "<br />";

        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);

        document.write("a - b = ");
        result = a - b;
        document.write(result);
        document.write(linebreak);

        document.write("a / b = ");
        result = a / b;

//rest of the code--→
```

```
 document.write(result);
         document.write(linebreak);

         document.write("a % b = ");
         result = a % b;
         document.write(result);
         document.write(linebreak);

         document.write("a + b + c = ");
         result = a + b + c;
         document.write(result);
         document.write(linebreak);

         a = ++a;
         document.write("++a = ");
         result = ++a;
         document.write(result);
         document.write(linebreak);

         b = --b;
         document.write("--b = ");
         result = --b;
         document.write(result);
         document.write(linebreak);
       //-->
     </script>

     Set the variables to different values and then try...
   </body>
</html>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
```

JavaScript Increment ++ and Decrement – –

- The increment and decrement operators in JavaScript will add one (+1) or subtract one (-1), respectively, to their operand, and then return a value.

- An operand is the quantity on which an operation is to be done.
- For example in the math equation 1 + 2, both 1 and 2 are operands, while + is the operator.

Syntax

Consider x to be the operand:

- Increment — x++ or ++x
- Decrement — x- - or - -x

As you can see, the ++/ - - operators can be used before or after the operand. Here’s what that might look in your code:

```
// Increment
let a = 1;
a++;
++a;

// Decrement
let b = 1;
b- -;
- -b;
```

Using ++/-- After the Operand

- When you use the increment/decrement operator after the operand, the value will be returned before the operand is increased/decreased.

Check out this example:

When we first log out the value of a, or b, neither has changed.

That’s because the original value of the operand is being returned prior to the operand being changed.

The next time the operator is used, we get the result of the +1, or -1.

```
// Increment
let a = 1;
console.log(a++);    // 1
console.log(a);      // 2

// Decrement
let b = 1;
console.log(b- -);    // 1
console.log(b);      // 0
```

Using ++/-- Before the Operand

- If you'd rather make the variable increment/decrement before returning, you simply have to use the increment/decrement operator before the operand:

As you can see in the above example, but using ++ or -- prior to our variable, the operation executes and adds/subtracts 1 prior to returning. This allows us to instantly log out and see the resulting value.

```
// Increment

let a = 1;
console.log(++a);    // 2
console.log(a);      // 2

// Decrement
let b = 1;
console.log(--b);    // 0
console.log(b);      // 0
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Increment
If used postfix, with operator after operand (for example, *x*++), the increment operator increments and returns the value before incrementing.

If used prefix, with operator before operand (for example, ++*x*), the increment operator increments and returns the value after incrementing.

Reference 2:
https://www.w3schools.com/js/js_arithmetic.asp

Comparison Operators

- JavaScript supports the following comparison operators
- Assume variable A holds 10 and variable B holds 20, then

Operator	Operator & Description
= = (Equal)	Checks if the value of two operands are equal or not, if yes, then the condition becomes Ex: (A == B) is not true.
!= (Not Equal)	Checks if the value of two operands are equal or not, if the values are not equal, condition becomes true. Ex: (A != B) is true.
>(Greater than)	Checks if the value of the left operand is greater than the value of the right operand, if y the condition becomes true. Ex: (A > B) is not true.
< (Less than)	Checks if the value of the left operand is less than the value of the right operand, if yes, condition becomes true. Ex: (A < B) is true.
>= (Greater than or Equal to)	Checks if the value of the left operand is greater than or equal to the value of the right if yes, then the condition becomes true. Ex: (A >= B) is not true.
<= (Less than or Equal to)	Checks if the value of the left operand is less than or equal to the value of the right op yes, then the condition becomes true. Ex: (A <= B) is true.

Example - Comparison Operators

```
<script type = "text/javascript">
        var a = 10; var b = 20;
        var linebreak = "<br />";

        document.write("(a == b) => ");
        result = (a == b);
        document.write(result);
        document.write(linebreak);

        document.write("(a < b) => ");
        result = (a < b);
        document.write(result);
        document.write(linebreak);

        document.write("(a > b) => ");
        result = (a > b);
        document.write(result);
        document.write(linebreak);

        document.write("(a != b) => ");
        result = (a != b);
        document.write(result);
        document.write(linebreak);

        document.write("(a >= b) => ");
        result = (a >= b);
        document.write(result);
        document.write(linebreak);

        document.write("(a <= b) => ");
        result = (a <= b);
        document.write(result);
        document.write(linebreak);
</script>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
```

Logical Operators

- JavaScript supports the following logical operators

Operator	Operator & Description
&& (Logical AND)	If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
|| (Logical OR)	If any of the two operands are non-zero, then the condition becomes true. Ex: (A || B) is true.
! (Logical NOT)	Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Bitwise Operators

- JavaScript supports the following bitwise operators
- Assume variable A holds 2 and variable B holds 3, then

Operator	Operator & Description
& (Bitwise AND)	It performs a Boolean AND operation on each bit of its integer arguments. Ex: (A & B) is 2.
| (BitWise OR)	It performs a Boolean OR operation on each bit of its integer arguments. Ex: (A | B) is 3.
^ (Bitwise XOR)	It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. Ex: (A ^ B) is 1.
~ (Bitwise Not)	It is a unary operator and operates by reversing all the bits in the operand. Ex: (~B) is -4.
<< (Left Shift)	It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. Ex: (A << 1) is 4.
>> (Right Shift)	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. Ex: (A >> 1) is 1.
>>> (Right shift with Zero)	This operator is just like the >> operator, except that the bits shifted in on the left are always zero. Ex: (A >>> 1) is 1.

Example - Bitwise operators

```
<script type = "text/javascript">
      <!--
        var a = 2; // Bit presentation 10
        var b = 3; // Bit presentation 11
        var linebreak = "<br />";

        document.write("(a & b) => ");
        result = (a & b);
        document.write(result);
        document.write(linebreak);

        document.write("(a | b) => ");
        result = (a | b);
        document.write(result);
        document.write(linebreak);

        document.write("(a ^ b) => ");
        result = (a ^ b);
        document.write(result);
        document.write(linebreak);

        document.write("(~b) => ");
        result = (~b);
        document.write(result);
        document.write(linebreak);

        document.write("(a << b) => ");
        result = (a << b);
        document.write(result);
        document.write(linebreak);

        document.write("(a >> b) => ");
        result = (a >> b);
        document.write(result);
        document.write(linebreak);
      //-->
    </script>
```

Output

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
```

Assignment Operators

- JavaScript supports the following assignment operators

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Operator	Operator & Description
=(Simple Assignment)	Assigns values from the right side operand to the left side operand Ex: C = A + B will assign the value of A + B into C
+=(Add and Assignment)	It adds the right operand to the left operand and assigns the result to the left operand. Ex: C += A is equivalent to C = C + A
−= (Subtract and Assignment)	It subtracts the right operand from the left operand and assigns the result to the left operand. Ex: C -= A is equivalent to C = C - A
***= (Multiply and Assignment)**	It multiplies the right operand with the left operand and assigns the result to the left operand. Ex: C *= A is equivalent to C = C * A
/= (Divide and Assignment)	It divides the left operand with the right operand and assigns the result to the left operand. Ex: C /= A is equivalent to C = C / A
%= (Modules and Assignment)	It takes modulus using two operands and assigns the result to the left operand. Ex: C %= A is equivalent to C = C % A

```
<script type = "text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var linebreak = "<br />";

        document.write("Value of a => (a = b) => ");
        result = (a = b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a += b) => ");
        result = (a += b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a -= b) => ");
        result = (a -= b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a *= b) => ");
        result = (a *= b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a /= b) => ");
        result = (a /= b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a %= b) => ");
        result = (a %= b);
        document.write(result);
        document.write(linebreak);
      //-->
</script>
```

Example - Assignment Operator

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
```

Conditional Operator (? :)

- The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No.	Operator and Description
1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        document.write ("((a > b) ? 100 : 200) => ");
        result = (a > b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);

        document.write ("((a < b) ? 100 : 200) => ");
        result = (a < b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);
      //-->
    </script>
    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
```

typeof Operator

- The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.
- The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the typeof Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

The following code shows how to implement typeof operator.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var a = 10;
        var b = "String";
        var linebreak = "<br />";

        result = (typeof b == "string" ? "B is String" : "B is Numeric");
        document.write("Result => ");
        document.write(result);
        document.write(linebreak);

        result = (typeof a == "string" ? "A is String" : "A is Numeric");
        document.write("Result => ");
        document.write(result);
        document.write(linebreak);
      //-->
    </script>
    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

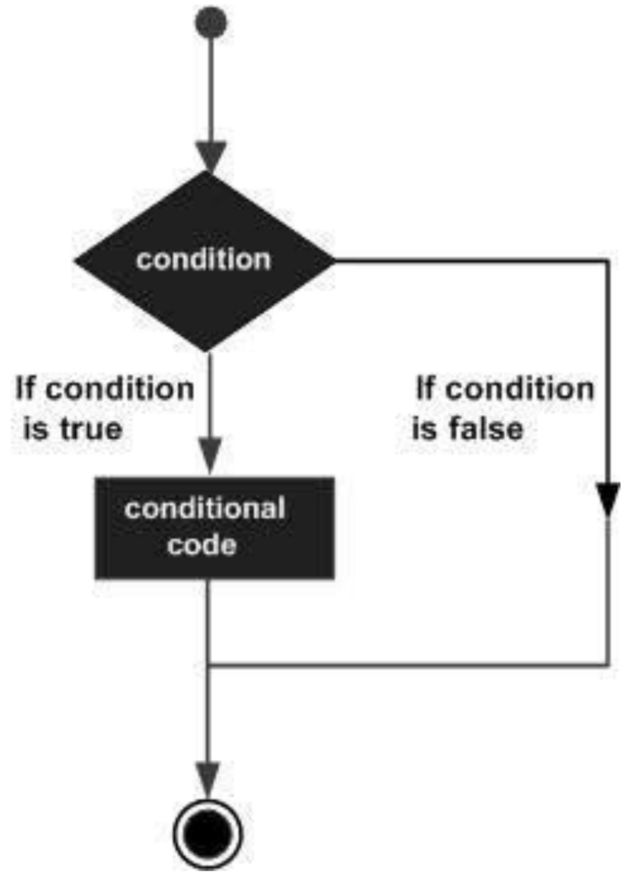
Output

```
Result => B is String
Result => A is Numeric
```

JavaScript - if...else Statement

Flow Chart of if-else

- The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of if..else statement

- if statement
- if...else statement
- if...else if... statement.

if statement

- The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows

```
if (expression) {
   Statement(s) to be executed if expression is true
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 20;

            if( age > 18 ) {
               document.write("<b>Qualifies for driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

```
Output

Qualifies for driving
```

if...else statement

- The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression) {
   Statement(s) to be executed if expression is true
} else {
   Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 15;

            if( age > 18 ) {
               document.write("<b>Qualifies for driving</b>");
            } else {
               document.write("<b>Does not qualify for driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

```
Output

Does not qualify for driving
```

if...else if... Statement

- The if...else if... statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows

```
if (expression 1) {
   Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
   Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
   Statement(s) to be executed if expression 3 is true
} else {
   Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var book = "maths";
        if( book == "history" ) {
          document.write("<b>History Book</b>");
        } else if( book == "maths" ) {
          document.write("<b>Maths Book</b>");
        } else if( book == "economics" ) {
          document.write("<b>Economics Book</b>");
        } else {
          document.write("<b>Unknown Book</b>");
        }
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
<html>
```

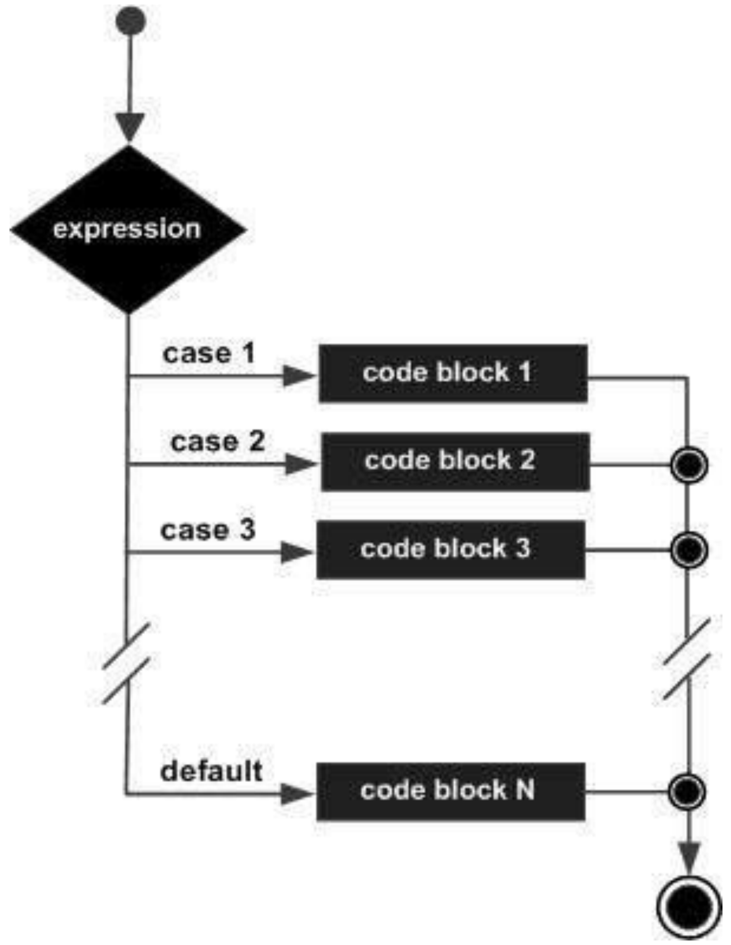
Output

```
Maths Book
```

JavaScript - Switch Case

Flow Chart

- The following flow chart explains a switch-case statement works.



Syntax

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
switch (expression) {
   case condition 1: statement(s)
   break;

   case condition 2: statement(s)
   break;
   ...

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.
We will explain break statement in Loop Control chapter.

Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          break;

          case 'B': document.write("Pretty good<br />");
          break;

          case 'C': document.write("Passed<br />");
          break;

          case 'D': document.write("Not so good<br />");
          break;

          case 'F': document.write("Failed<br />");
          break;

          default:  document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Entering switch block
Good job
Exiting switch block
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          case 'B': document.write("Pretty good<br />");
          case 'C': document.write("Passed<br />");
          case 'D': document.write("Not so good<br />");
          case 'F': document.write("Failed<br />");
          default: document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
    </script>
    <p>Set the variable to different value and then
try...</p>
  </body>
</html>
```

Output

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
```

JavaScript - While Loops

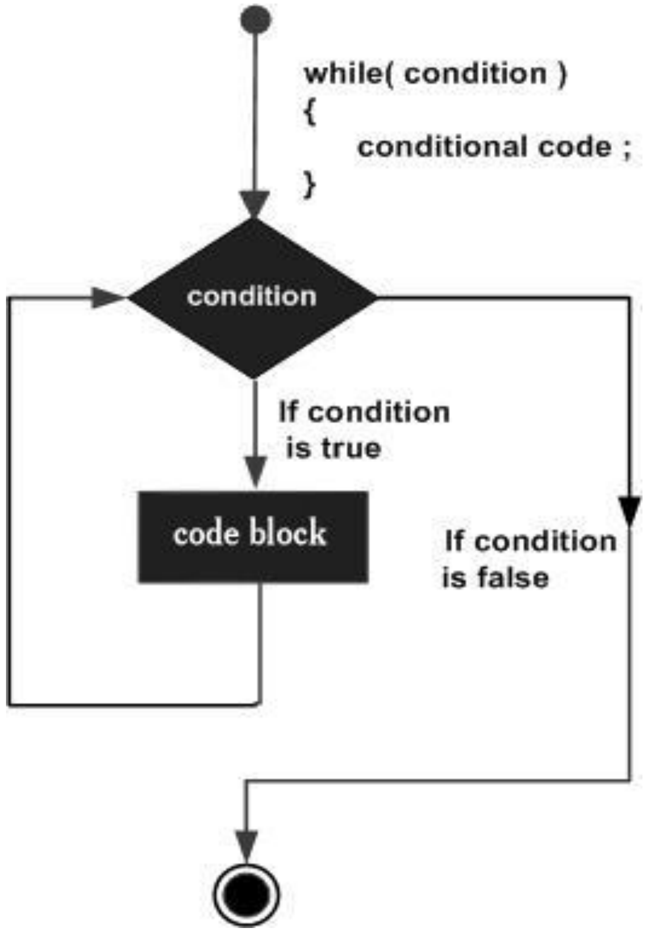
The while Loop

- The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Syntax

The syntax of while loop in JavaScript is as follows −

```
while (expression) {
   Statement(s) to be executed if expression is true
```



Example

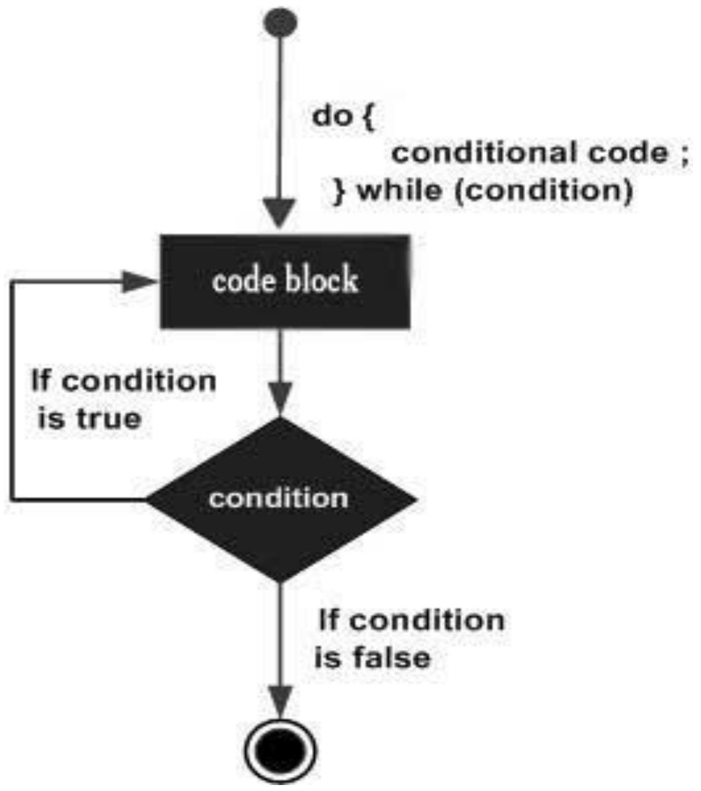
```
<html>
  <body>

    <script type = "text/javascript">
      <!--
        var count = 0;
        document.write("Starting Loop ");

        while (count < 10) {
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        document.write("Loop stopped!");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

The do...while Loop

- The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.



Syntax

The syntax for do-while loop in JavaScript is as follows −

```
do {
   Statement(s) to be executed;
} while (expression);
```

Note − Don’t miss the semicolon used at the end of the **do...while** loop.

Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var count = 0;

        document.write("Starting Loop" + "<br />");
        do {
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        while (count < 5);
        document.write ("Loop stopped!");
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
```

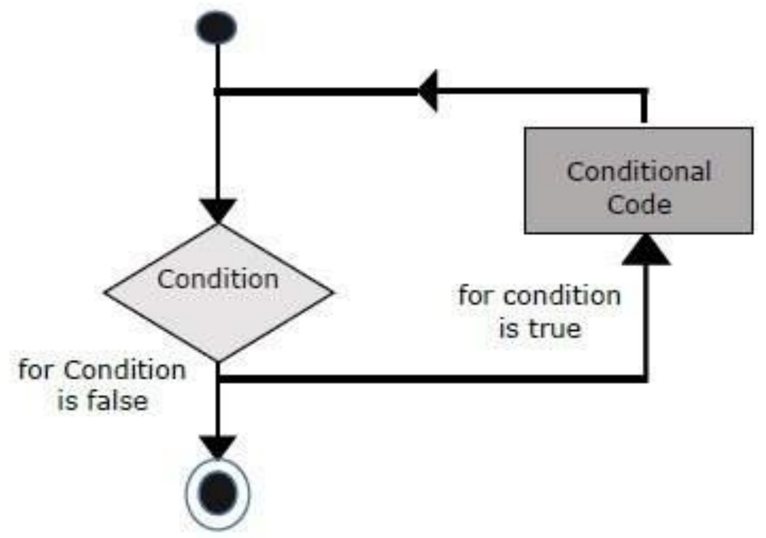
JavaScript - For Loop

JavaScript - For Loop

The 'for' loop is the most compact form of looping. It includes the following three important parts

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons



Syntax

The syntax of for loop is JavaScript is as follows

```
for (initialization; test condition; iteration statement) {
   Statement(s) to be executed if test condition is true
}
```

Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var count;
        document.write("Starting Loop" + "<br />");

        for(count = 0; count < 10; count++) {
          document.write("Current Count : " + count );
          document.write("<br />");
        }
        document.write("Loop stopped!");
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

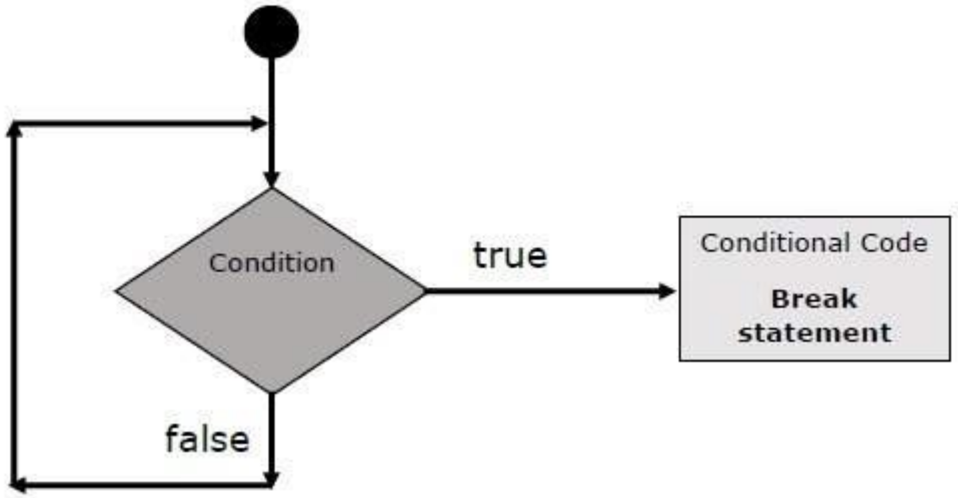
Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

JavaScript - Loop Control

The break Statement

The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.



Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
      var x = 1;
      document.write("Entering the loop<br /> ");

      while (x < 20) {
        if (x == 5) {
          break;   // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }
      document.write("Exiting the loop!<br /> ");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...
```

The continue Statement

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop

Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var x = 1;
        document.write("Entering the loop<br /> ");

        while (x < 10) {
          x = x + 1;

          if (x == 5) {
            continue;   // skip rest of the loop body
          }
          document.write( x + "<br />");
        }
        document.write("Exiting the loop!<br /> ");
      //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
```

Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with break and continue 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. We will see two different examples to understand how to use labels with break and contin

Note − Line breaks are not allowed between the **'continue'** or **'break'** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Example

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        document.write("Entering the loop!<br /> ");
        outerloop:        // This is the label name
        for (var i = 0; i < 5; i++) {
          document.write("Outerloop: " + i + "<br />");
          innerloop:
          for (var j = 0; j < 5; j++) {
            if (j > 3 ) break ;           // Quit the innermost loop
            if (i == 2) break innerloop;  // Do the same thing
            if (i == 4) break outerloop;  // Quit the outer loop
            document.write("Innerloop: " + j + " <br />");
          }
        }
        document.write("Exiting the loop!<br /> ");
      //-->
    </script>
  </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
```

Using Labels to Control the Flow

Example

```
<html>
  <body>

    <script type = "text/javascript">
      <!--
      document.write("Entering the loop!<br /> ");
      outerloop:     // This is the label name

      for (var i = 0; i < 3; i++) {
        document.write("Outerloop: " + i + "<br />");
        for (var j = 0; j < 5; j++) {
          if (j == 3) {
            continue outerloop;
          }
          document.write("Innerloop: " + j + "<br />");
        }
      }

      document.write("Exiting the loop!<br /> ");
      //-->
    </script>

  </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
Innerloop: 2
Exiting the loop!
```

JavaScript - Functions

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type = "text/javascript">
   <!--
      function functionname(parameter-list) {
         statements
      }
   //-->
</script>
```

Example

```
<script type = "text/javascript">
   <!--
      function sayHello() {
         alert("Hello there");
      }
   //-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

Example

```
<html>
  <head>
    <script type = "text/javascript">
      function sayHello() {
        document.write ("Hello there!");
      }
    </script>

  </head>

  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type = "button" onclick = "sayHello()" value = "Say Hello">
    </form>
    <p>Use different text in write method and then try...</p>
  </body>
</html>
```

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

```
<html>
  <head>
    <script type = "text/javascript">
      function sayHello(name, age) {
        document.write (name + " is " + age + " years old.");
      }
    </script>
  </head>

  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type = "button" onclick = "sayHello('Zara', 7)" value = "Say
Hello">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.
For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

```
<html>
  <head>
    <script type = "text/javascript">
      function concatenate(first, last) {
        var full;
        full = first + last;
        return full;
      }
      function secondFunction() {
        var result;
        result = concatenate('Zara', 'Ali');
        document.write (result );
      }
    </script>
  </head>

  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type = "button" onclick = "secondFunction()" value = "Call Function">
    </form>
    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

Thank you