Introduction

JavaScript Tutorial for beginners and professionals is a solution of client side dynamic pages.

JavaScript is *an object-based scripting language* that is lightweight and cross-platform.

JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

Client-side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are −

- Less server interaction − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors − They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

Where JavaScript is used

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)

- Displaying clocks etc.

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here −

- Microsoft FrontPage − Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- Macromedia Dreamweaver MX − Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- Macromedia HomeSite 5 − HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Where is JavaScript Today ?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site: http://www.ecmascript.org/

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script>.

You can place the <script> tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
  JavaScript code
</script>
```

The script tag takes two important attributes −

- Language − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- Type − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like −

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

Your First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function document.write which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
  <body>
    <script language="javascript" type="text/javascript">
      <!--
```

```
      document.write("Hello World!")
    //-->
  </script>
 </body>
</html>
```

This code will produce the following result −

```
Hello World!
```

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10
    var2 = 20
  //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons −

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10; var2 = 20;
  //-->
```

```
</script>
```

Note − It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers Time and TIME will convey different meanings in JavaScript.

NOTE − Care should be taken while writing variable and function names in JavaScript.

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

JavaScript in Internet Explorer

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer −

- Follow Tools → Internet Options from the menu.
- Select Security tab from the dialog box.
- Click the Custom Level button.
- Scroll down till you find Scripting option.
- Select *Enable* radio button under Active scripting.
- Finally click OK and come out

To disable JavaScript support in your Internet Explorer, you need to select Disable radio button under Active scripting.

JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox −

- Open a new tab → type about: config in the address bar.
- Then you will find the warning dialog. Select I'll be careful, I promise!
- Then you will find the list of configure options in the browser.
- In the search bar, type javascript.enabled.

- There you will find the option to enable or disable javascript by right-clicking on the value of that option → select toggle.

If javascript.enabled is true; it converts to false upon clicking toogle. If javascript is disabled; it gets enabled upon clicking toggle.

JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome −

- Click the Chrome menu at the top right hand corner of your browser.
- Select Settings.
- Click Show advanced settings at the end of the page.
- Under the Privacy section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera −

- Follow Tools → Preferences from the menu.
- Select Advanced option from the dialog box.
- Select Content from the listed items.
- Select Enable JavaScript checkbox.
- Finally click OK and come out.

To disable JavaScript support in your Opera, you should not select the Enable JavaScript checkbox.

Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using <noscript> tags.

You can add a noscript block immediately after the script block as follows −

```
<html>
  <body>

    <script language="javascript" type="text/javascript">
```

```
      <!--
        document.write("Hello World!")
      //-->
    </script>

    <noscript>
      Sorry...JavaScript is needed to go ahead.
    </noscript>

  </body>
</html>
```

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.

JavaScript Where To

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript in Body</h2>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
</body>
</html>
```

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows −

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows −

```
<html>

  <head>

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

  </head>

  <body>
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </body>

</html>
```

This code will produce the following results −

JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

```
<html>

  <head>
  </head>

  <body>

    <script type="text/javascript">
      <!--
        document.write("Hello World")
```

```
      //-->
    </script>

    <p>This is web page body </p>

  </body>
</html>
```

This code will produce the following results −

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows −

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

  <body>
    <script type="text/javascript">
      <!--
        document.write("Hello World")
      //-->
    </script>

    <input type="button" onclick="sayHello()" value="Say Hello" />

  </body>
</html>
```

JavaScript in External File

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

```
<html>

  <head>
    <script type="text/javascript" src="filename.js" ></script>
  </head>

  <body>
    .......
  </body>
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use sayHello function in your HTML file after including the filename.js file.

```
function sayHello() {

  alert("Hello World")

}
```

JavaScript Comment

1. JavaScript comments
2. Advantage of javaScript comments
3. Single-line and Multi-line comments

The JavaScript comments are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. To make code easy to understand It can be used to elaborate the code so that end user can easily understand the code.

2. To avoid the unnecessary code It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.

Let's see the example of single-line comment i.e. added before the statement.

```
<script>
// It is single line comment
document.write("hello javascript");
</script>
```

Let's see the example of single-line comment i.e. added after the statement.

```
<script>
var a=10;
var b=20;
var c=a+b;//It adds values of a and b variable
document.write(c);//prints sum of 10 and 20
</script>
```

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

```
/* your code here  */
```

It can be used before, after and middle of the statement.

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

JavaScript Variable

1. JavaScript variable
2. JavaScript Local variable
3. JavaScript Global variable

A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
var x = 10;
var _value="sonoo";
```

Incorrect JavaScript variables

```
var  123=30;
var *aa=320;
```

Example of JavaScript variable

Let’s see a simple example of JavaScript variable.

```
<script>
var x = 10;
```

```
var y = 20;
var z=x+y;
document.write(z);
</script>
```

Output of the above example

```
30
```

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

Or,

```
<script>
If(10<13){
var y=20;//JavaScript local variable
}
</script>
```

JavaScript global variable

A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
```

```
}
a();//calling JavaScript function
b();
</script>
```

Javascript Data Types

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
1. var a=40;//holding number
2. var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

1. var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false

>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
In	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

```
<!DOCTYPE html>

<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation, and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(p1, p2)
{
   return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```

JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: {}

```
function name(parameter1, parameter2, parameter3) {
   code to be executed
}
```

Function parameters are listed inside the parentheses () in the function definition.

Function arguments are the values received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```

The result in x will be:

12

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

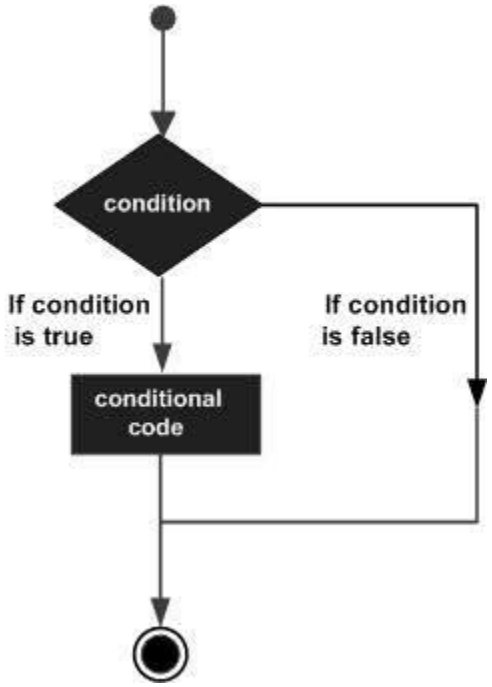
Control Statements

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..elsestatement.

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

Try the following example to understand how the if statement works.

```
<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
Set the variable to different value and then try...
```

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

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<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
Set the variable to different value and then try...
```

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

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<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
Set the variable to different value and then try...
```

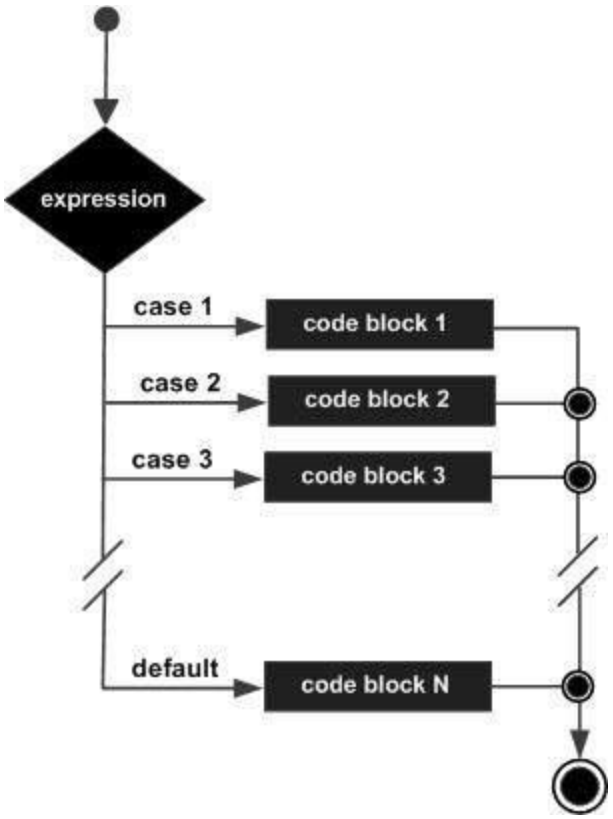
Switch case

You can use multiple if...else…if statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else ifstatements.

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

Try the following example to implement switch-case 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 />");
          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
Set the variable to different value and then try...
```

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
Set the variable to different value and then try...
```

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

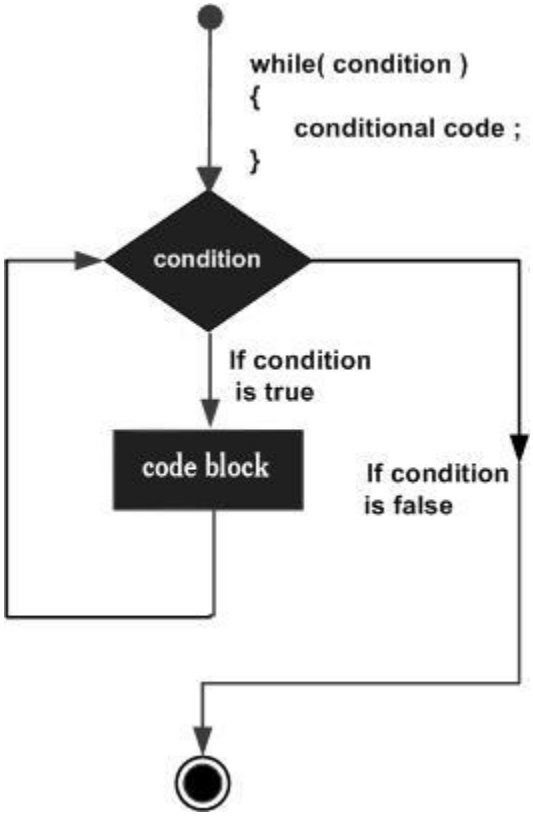
JavaScript supports all the necessary loops to ease down the pressure of programming.

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.

Flow Chart

The flow chart of while loop looks as follows −



Syntax

The syntax of while loop in JavaScript is as follows −

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

Example

Try the following example to implement while loop.

```
<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

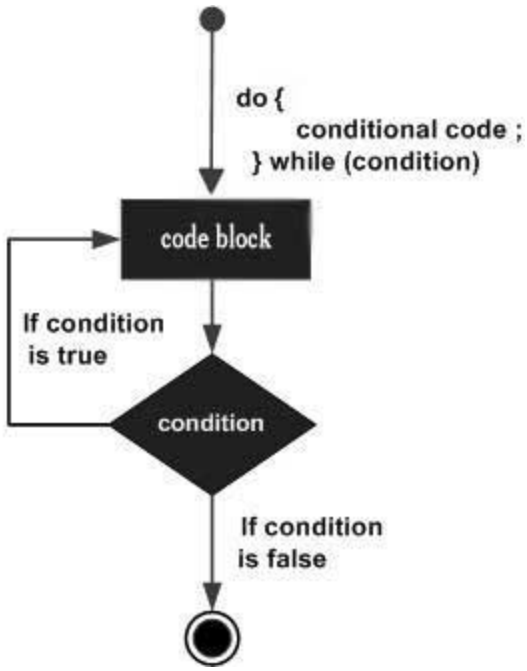
```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

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.

Flow Chart

The flow chart of a do-while loop would be as follows −



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

Try the following example to learn how to implement a do-while loop in JavaScript.

```
<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!
Set the variable to different value and then try...
```

The for loop

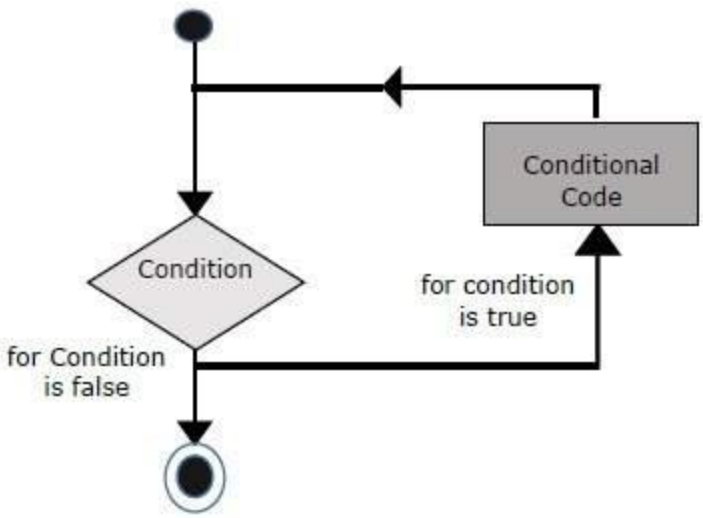
The 'for' loop is the most compact form of looping. It includes the following three important parts −

- 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.

Flow Chart

The flow chart of a for loop in JavaScript would be as follows −



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

Try the following example to learn how a for loop works in JavaScript.

```
<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!
Set the variable to different value and then try...
```

For in loop

The for...in loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

```
for (variablename in object){
  statement or block to execute
}
```

In each iteration, one property from object is assigned to variablename and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to implement 'for-in' loop. It prints the web browser's Navigator object.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var aProperty;
        document.write("Navigator Object Properties<br /> ");
```

```
        for (aProperty in navigator) {
          document.write(aProperty);
          document.write("<br />");
        }
        document.write ("Exiting from the loop!");
      //-->
    </script>

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

Output

```
Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
hardwareConcurrency
maxTouchPoints
vendorSub
vendor
productSub
cookieEnabled
mimeTypes
plugins
javaEnabled
getStorageUpdates
getGamepads
webkitGetUserMedia
vibrate
getBattery
sendBeacon
registerProtocolHandler
unregisterProtocolHandler
```

```
Exiting from the loop!
Set the variable to different object and then try...
```

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

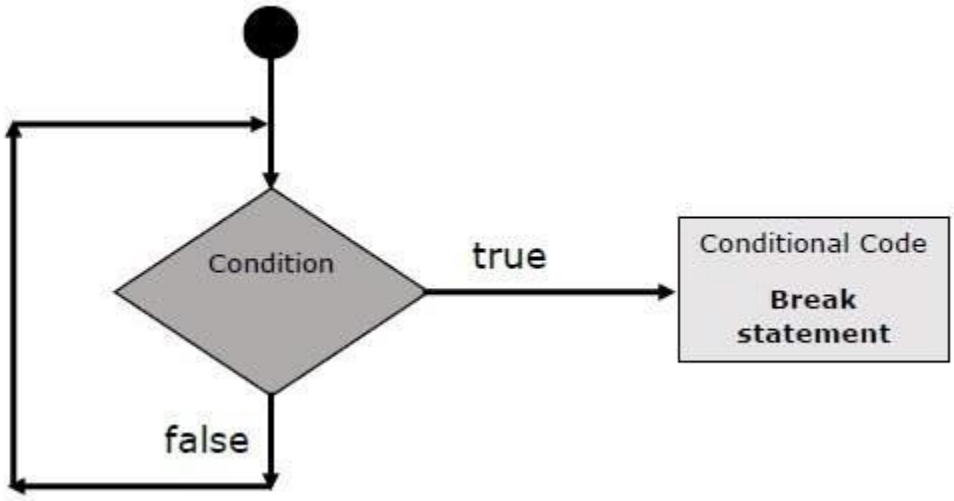
To handle all such situations, JavaScript provides break and continuestatements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

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.

Flow Chart

The flow chart of a break statement would look as follows −



Example

The following example illustrates the use of a break statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to document.write (..) statement just below to the closing curly brace −

```
<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...
```

We already have seen the usage of break statement inside a switchstatement.

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 continuestatement 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

This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5 −

```
<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 continue.

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.

Try the following two examples for a better understanding of Labels.

Example 1

The following example shows how to implement Label with a break statement.

```
<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
Exiting the loop!
```

Example 2

```
<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 Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.
When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

window.alert("*sometext*");

The window.alert() method can be written without the window prefix.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Alert</h2>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  alert("I am an alert box!");
}
</script>

</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.
When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

window.confirm("*sometext*");

The window.confirm() method can be written without the window prefix.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Confirm Box</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var txt;
   if (confirm("Press a button!") == true) {
      txt = "You pressed OK!";
   } else {
      txt = "You pressed Cancel!";
   }
   document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.
When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

window.prompt("*sometext*","*defaultText*");

The window.prompt() method can be written without the window prefix.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Prompt</h2>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>
```

```
<script>
function myFunction() {
   var txt;
   var person = prompt("Please enter your name:", "Harry Potter");
   if (person == null || person == "") {
      txt = "User cancelled the prompt.";
   } else {
      txt = "Hello " + person + "! How are you today?";
   }
   document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```