

JavaScript

A large, stylized 'Js' logo is centered on a yellow-to-green gradient square. The 'J' and 'S' are dark blue-grey with a 3D effect, featuring a lighter blue-grey outline and a darker blue-grey shadow. The 'j' has a small dot. The background square has a subtle gradient from light yellow at the top to a darker olive green at the bottom.

About the Tutorial

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Audience

This tutorial has been prepared for JavaScript beginners to help them understand the basic functionality of JavaScript to build dynamic web pages and web applications.

Prerequisites

For this tutorial, it is assumed that the reader have a prior knowledge of HTML coding. It would help if the reader had some prior exposure to object-oriented programming concepts and a general idea on creating online applications.

Table of Contents

About the Tutorial	ii
Audience.....	ii
Prerequisites	ii
1. Client Side Scripting Language	1
What is client-side and server-side?.....	1
2. JavaScript	2
What is JavaScript?	2
Client-Side JavaScript	2
Advantages of JavaScript	2
Limitations of JavaScript	3
JavaScript Development Tools.....	3
3. SYNTAX	4
JavaScript Outputs.....	4
Your First JavaScript Code	4
Comments in JavaScript.....	5
4. VARIABLES	6
JavaScript Data types	6
JavaScript Variables	6
JavaScript Variable Scope.....	7
JavaScript Variable Names	8
JavaScript Reserved Words.....	8
5. OPERATORS.....	9
What is an Operator?.....	9
Arithmetic Operators.....	9
Comparison Operators.....	10
Logical Operators	12
Bitwise Operators	13
Assignment Operators	14
Miscellaneous Operators	16
Conditional Operator (? :).....	16
typeof Operator	17

6. IF...ELSE STATEMENT 18

Flow Chart of if-else.....	18
if statement	18
Syntax	19
Example	19
Output	19
if...else statement:.....	20
Syntax	20
Example	20
Output	21
if...else if... statement.....	21
Syntax	21
Example	22
Output	23

7. FOR LOOP..... 24

The for Loop.....	24
Flow Chart	24
Syntax	24
Example	24

8. WHILE LOOP 26

The while Loop	26
Flow Chart	26
The do...while Loop	27

9. SWITCH-CASE 30

Flow Chart.....	30
Syntax.....	30

10. LOOP CONTROL 33

The break Statement.....	33
Flow Chart	33
Example	33
The continue Statement.....	34
Using Labels to Control the Flow	35

Example 2	36
11. FOR-IN LOOP	38
Syntax	38
12. FUNCTIONS	40
Function Definition	40
Syntax	40
Calling a Function	41
Function Parameters	41
The return Statement	42
Nested Functions	43
Function () Constructor	44
Function Literals	46
13. OBJECTS	48
Object Properties	48
Object Methods	48
User-Defined Objects	49
The new Operator	49
The Object () Constructor	49
Defining Methods for an Object	50
The 'with' Keyword	51
Syntax	51
14. ERRORS AND EXCEPTIONS.....	53
Syntax Errors	53
Runtime Errors	53
Logical Errors.....	53
The try...catch Statement.....	53
The throw Statement	55
Finally Statement.....	56
The onerror() Method.....	57
15. PAGE REDIRECT	60
What is Page Redirection?	60
JavaScript Page Refresh	60

Auto Refresh	60
How Page Re-direction Works?	61
16. DIALOG BOX.....	63
Alert Dialog Box	63
Confirmation Dialog Box.....	64
Prompt Dialog Box.....	65
17. jQuery and Its Plugins	67
What is jQuery?.....	67
How to use jQuery?	67
Local Installation	67
CDN Based Version.....	68
How to call a jQuery Library Functions?	69
jQuery slideDown() Method	70
jQuery slideToggle() Method.....	71

1. Client Side Scripting Language

Client-side scripting generally refers to the class of computer programs on the web that are executed client-side, by the user's web browser, instead of server-side (on the web server).

What is client-side and server-side?

- Any machine can play the role of either a client or a server
- You could even have a machine being both
- Some languages, e.g. Javascript, are said to be client-side.
- Run on the user's browser/web client
- Other languages, e.g. PHP, are said to be server-side.
- Run on the server that is delivering content to the user



2. JavaScript

What is JavaScript?

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.

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.

3. SYNTAX

JavaScript can be implemented using JavaScript statements that are placed within the `<script>... </script>` HTML tags in a web page.

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 syntax will look as follows.

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

JavaScript Outputs

JavaScript can "display" data in different ways:

- Writing into an alert box, using `window.alert()`.
- Writing into the HTML output using `document.write()`.
- Writing into an HTML element, using `innerHTML`.
- Writing into the browser console, using `console.log()`.

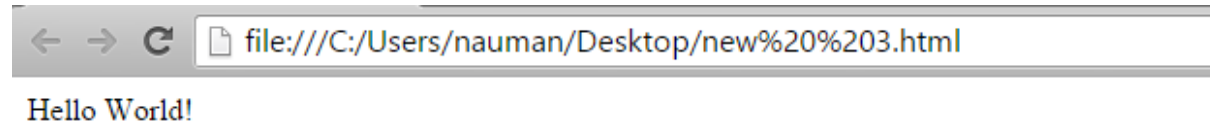
Your First JavaScript Code

Let us take a sample example to print out "Hello World".

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>
```

Result



Comments in JavaScript

JavaScript supports both C-style and C++-style comments. Thus:

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence `<!--`.
- JavaScript treats this as a single-line comment, just as it does the `//` comment.
- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

Example

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
<!--
// This is a comment. It is similar to comments in C++
/*
* This is a multiline comment in JavaScript
* It is very similar to comments in C Programming
*/
//-->
</script>
```

4. VARIABLES

JavaScript Data types

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- Numbers, e.g., 123, 120.50 etc.
- Strings of text, e.g. "This text string" etc.
- Boolean, e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object. We will cover objects in detail in a separate chapter.

Note: Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type="text/javascript">
<!--
    var money;
    var name;
    //-->
</script>
```

You can also declare multiple variables with the same var keyword as follows:

```
<script type="text/javascript">
<!--
    var money, name;
    //-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value

2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.\

Example

```
<script type="text/javascript">
<!--
    var name = "Ali";
    var money;
    money = 2000.50;
    //-->
</script>
```

Note: Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<html>
<head>
    <title>User-defined objects</title>
    <script type="text/javascript">

        var myVar = "global"; // Declare a global variable
        function checkscope() {
            var myVar = "local"; // Declare a local variable
        }
        document.write(myVar); //if function will call then result will
        local
    </script>
</head>
```

```
<body>

</body>
</html>
```

Result:



JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract boolean break byte case catch char class const continue debugger default delete do double	else enum export extends false final finally float for function goto if implements import in	Instanceof int interface long native new null package private protected public return short static super	switch synchronized this throw throws transient true try typeof var void volatile while with
---	---	---	---

5. OPERATORS

What is an Operator?

Let us take a simple expression $4 + 5$ is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Let's have a look at all the operators one by one.

Arithmetic Operators

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

Operator	Description	Example
+ (Addition)	Adds two operands	Ex: A + B will give 30
- (Subtraction)	Subtracts the second operand from the first	Ex: A - B will give -10
* (Multiplication)	Multiply both operands	Ex: A * B will give 200
/ (Division)	Divide the numerator by the denominator	Ex: B / A will give 2
% (Modulus)	Outputs the remainder of an integer division	Ex: B % A will give 0
++ (Increment)	Increases an integer value by one	Ex: A++ will give 11
-- (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

The following code shows how to use arithmetic operators in JavaScript.

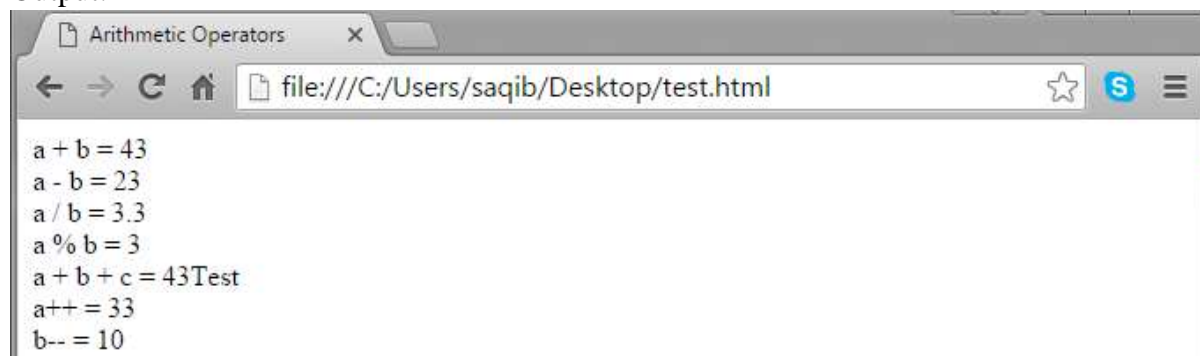
```
<html>
<head>
  <title>Arithmetic Operators
</title>
</head>
<body>
  <script type="text/javascript">
    var a = 33;
    var b = 10;
    var c = "Test";
    var linebreak = "<br />";
    document.write("a + b = "); result = a + b;
```

```

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;
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>
</body>
</html>

```

Output:



Comparison Operators

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

Operator	Description	Example
== (Equal)	Checks if the value of two operands are equal or not, if yes, then the condition becomes true.	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, then the 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	Ex: (A > B) is not true.

	yes, then the condition becomes true.	
< (Less than)	Checks if the value of the left operand is less than the value of the right operand, if yes, then the 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 operand, 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 operand, if yes, then the condition becomes true.	Ex: (A <= B) is true.

Example

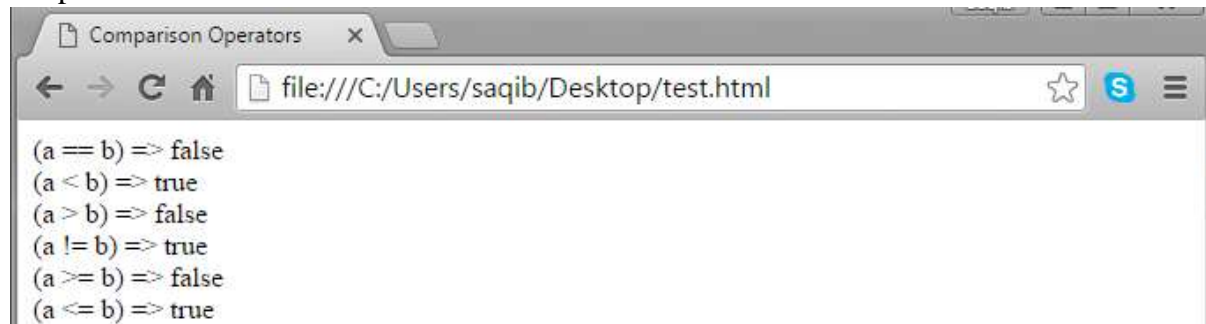
The following code shows how to use comparison operators in JavaScript.

```

<html>
<head>
  <title>Comparison Operators</title>
</head>
<body>
  <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>
</body>
</html>

```

Output:



Logical Operators

JavaScript supports the following logical operators:

Assume variable A holds 10 and variable B holds 20, then:

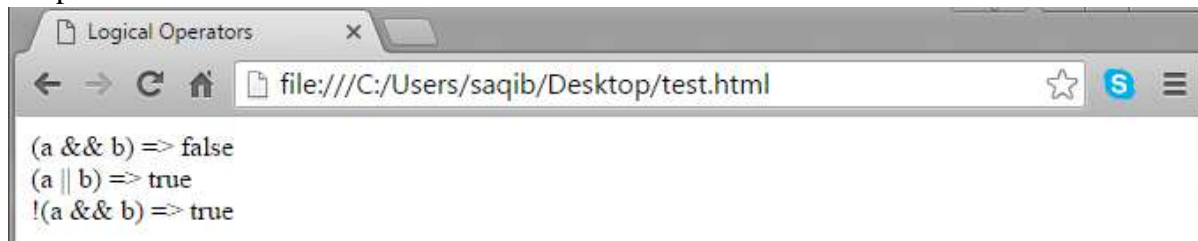
Operator	Description	Example
&& (Logical AND)	If both the operands are non-zero, then the condition becomes true.	Ex: (A && B) is true.
 (Logical OR)	If any of the two operands are non-zero, then the condition becomes true.	Ex: (A B) is true.
! (Logical NOT)	Reverses the logical state of its operand.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

Example

```
<html>
<head>
  <title>Logical Operators</title>
</head>
<body>
  <script type="text/javascript">
    var a = true;
    var b = false;
    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);
  </script>
</body>
</html>
```

Try the following code to learn how to implement Logical Operators in JavaScript.

Output:



The screenshot shows a web browser window with the title 'Logical Operators'. The address bar displays 'file:///C:/Users/saqib/Desktop/test.html'. The main content area shows the following JavaScript expressions and their results:

```
(a && b) => false
(a || b) => true
!(a && b) => true
```

Bitwise Operators

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

Operator	Description	Example
& (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

Try the following code to implement Bitwise operator in JavaScript.

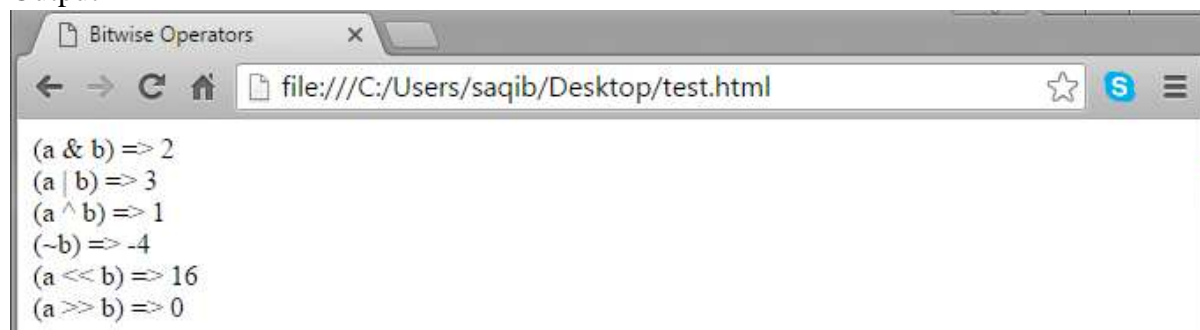
```
<html>
```

```

<head>
  <title>Bitwise Operators
</title>
</head>
<body>
  <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>
</body>
</html>

```

Output



Assignment Operators

JavaScript supports the following assignment operators:

Operator	Description	Example
= (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

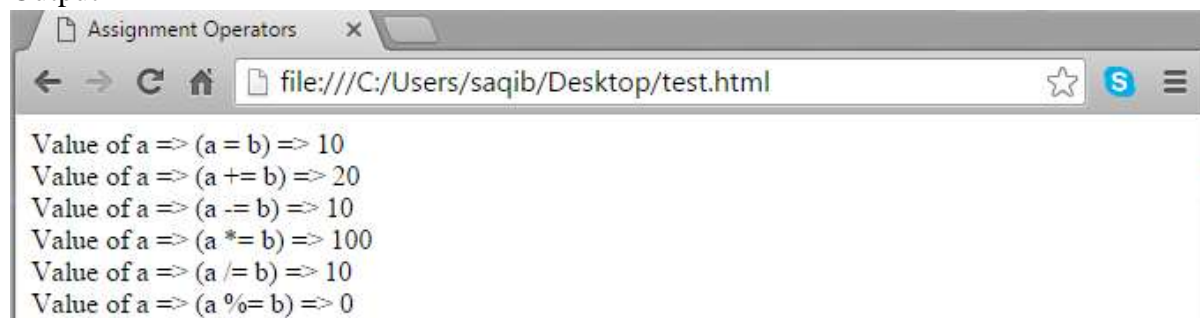
*= (Multiply and Assignment)	It multiplies the right operand with the left operand and assigns the result to the left operand.	Ex: C *= A is equivalent to C = C * A
/= (Divide and Assignment)	It divides the left operand with the right operand and assigns the result to the left operand.	Ex: C /= A is 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

Example

Try the following code to implement assignment operator in JavaScript.

```
<html>
<head>
  <title>Assignment Operators
</title>
</head>
<body>
  <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>
</body>
</html>
```

Output



Miscellaneous Operators

We will discuss two operators here that are quite useful in JavaScript: the conditional operator (`? :`) and the `typeof` operator.

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.

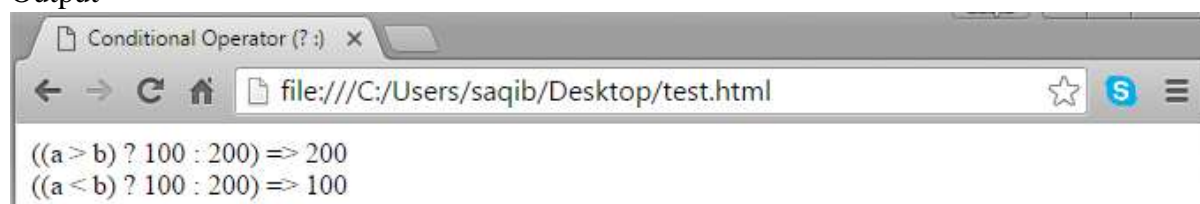
Operator	Description
? : (Conditional)	If Condition is true? Then value X : Otherwise value Y

Example

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

```
<html>
<head>
  <title>Conditional Operator (? :)
</head>
<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>
</body>
</html>
```

Output



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.

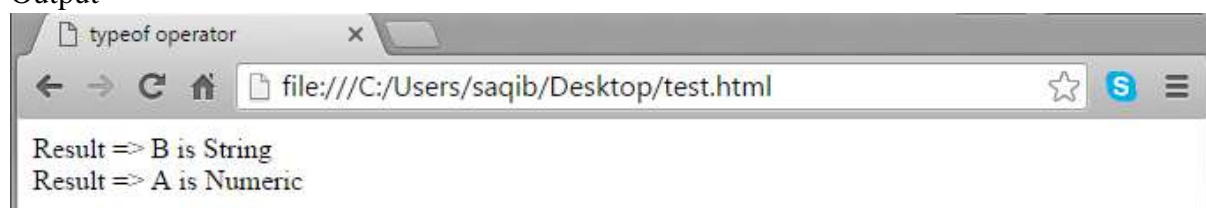
Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

The following code shows how to implement typeof operator.

```
<html>
<head>
  <title>typeof operator
</title>
</head>
<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>
</body>
</html>
```

Output



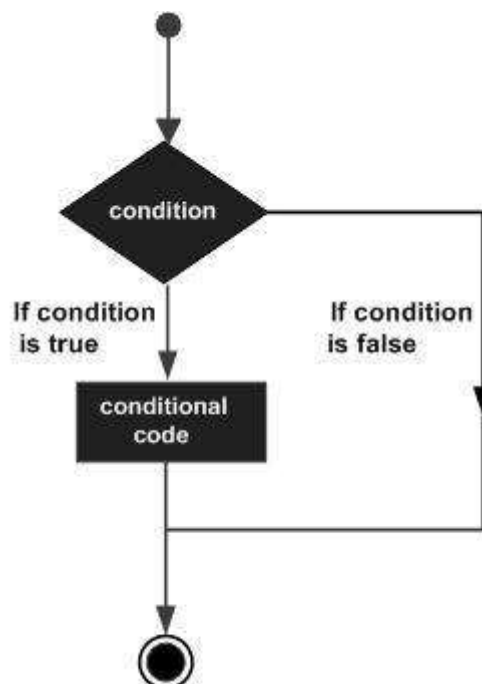
6. IF...ELSE STATEMENT

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..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

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

```
//-->
</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

Math's Book

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

7. FOR LOOP

The for Loop

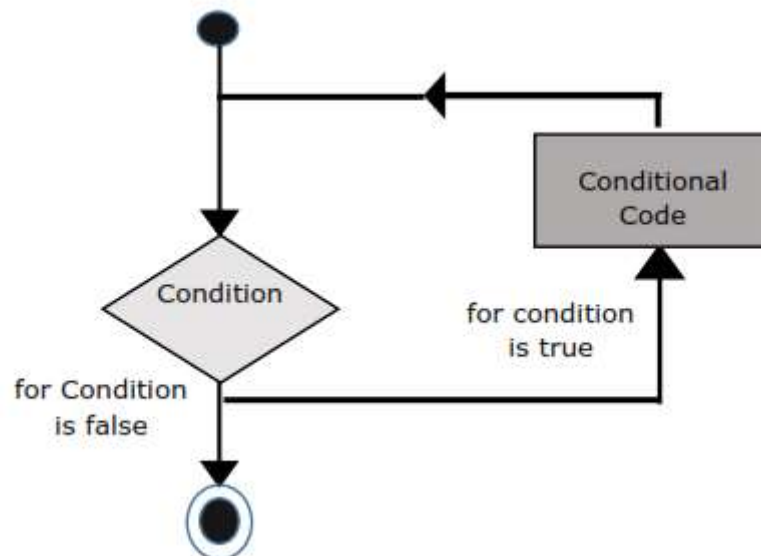
The 'for' loop is the most compact form of looping. It includes the following three important parts:

- 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 for loop in JavaScript would be as follows:



Syntax

The syntax of for loop in JavaScript is as follows:

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

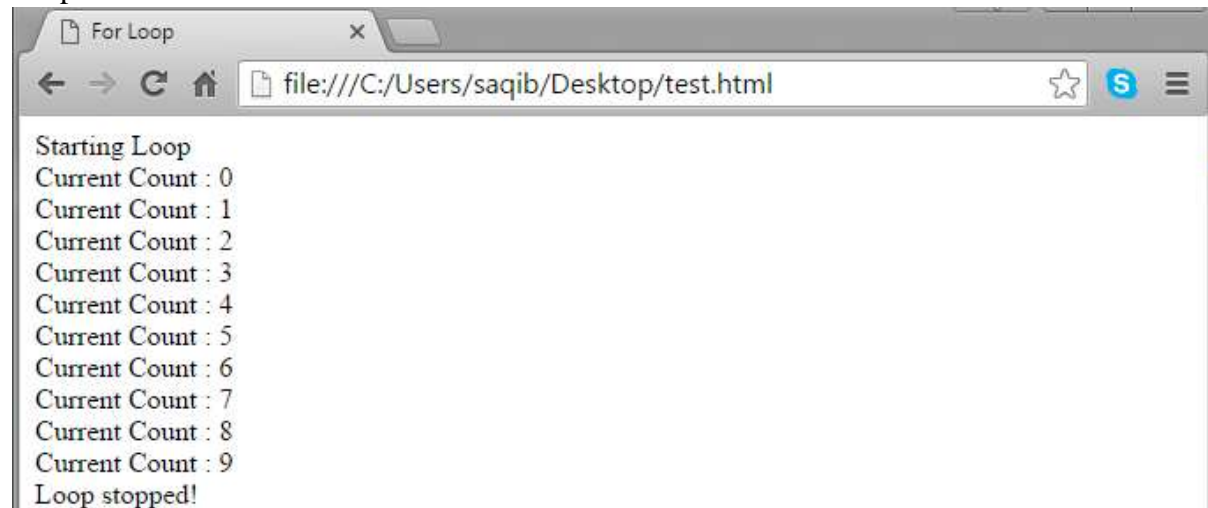
Example

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

```
<html>
<head>
  <title>For Loop
```

```
</title>
</head>
<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>
</body>
</html>
```

Output



8. WHILE LOOP

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.

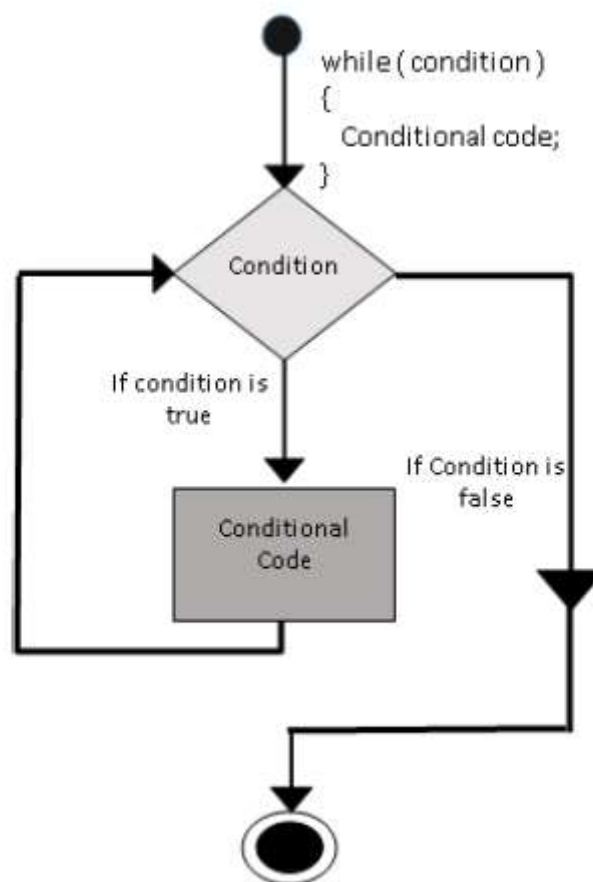
JavaScript supports all the necessary loops to ease down the pressure of programming.

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:

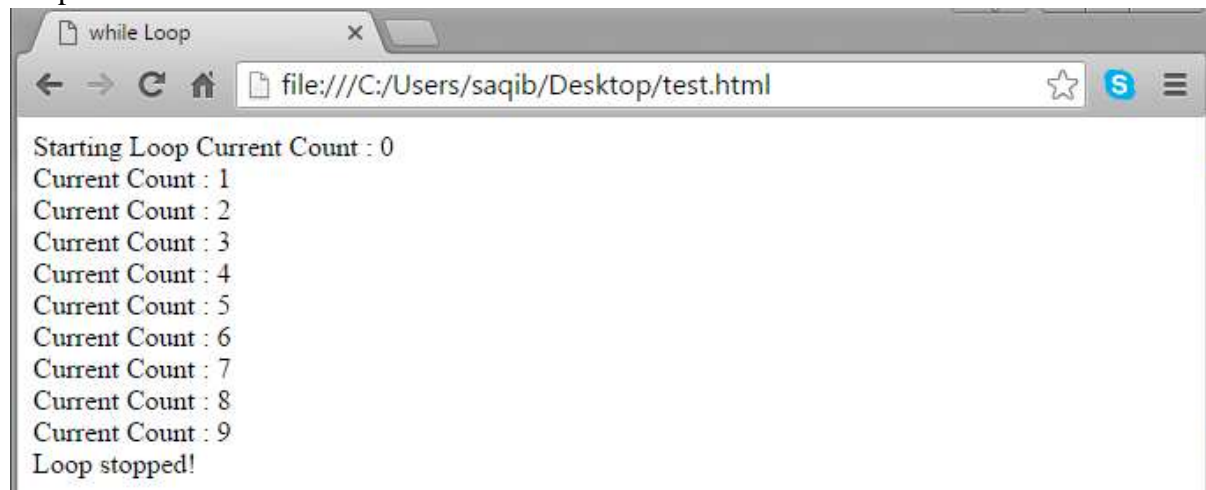
```
while (expression){  
    Statement(s) to be executed if expression is true  
}
```

Example

Try the following example to implement while loop.


```
<html>
<head>
  <title>while Loop
</title>
</head>
<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>
</body>
</html>
```

Output

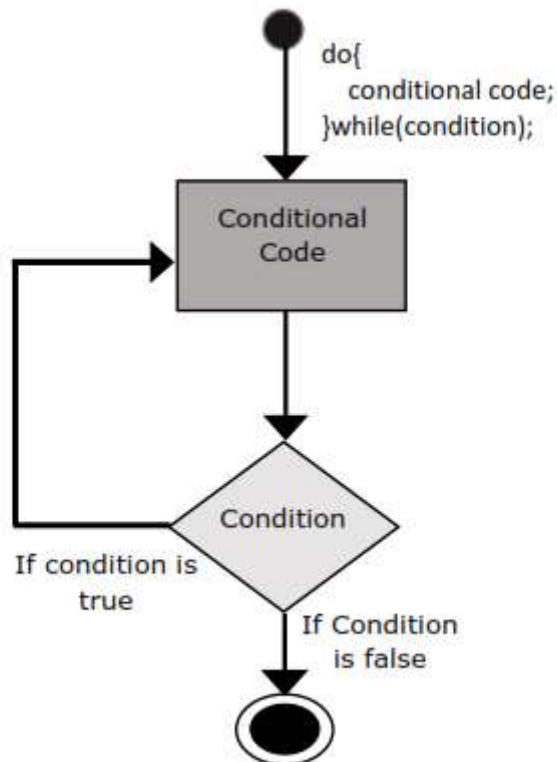


The do...while Loop

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.

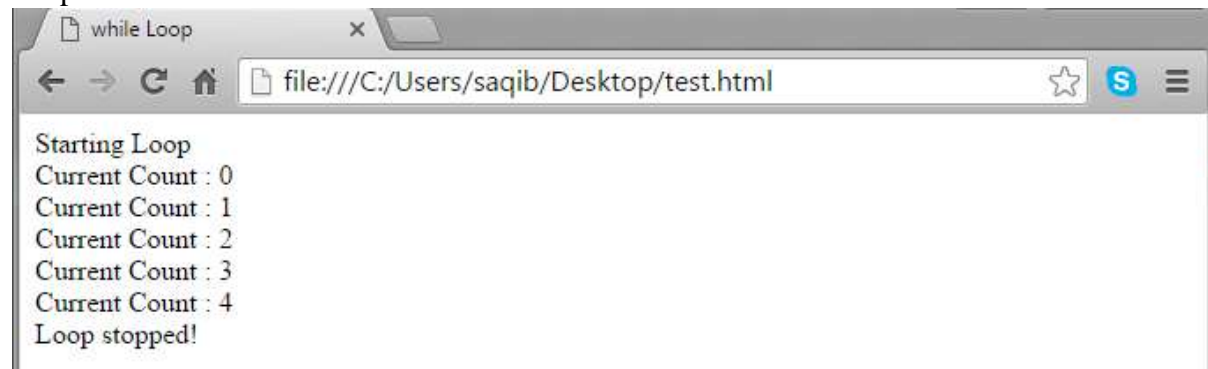
Example

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

```
<html>  
<head>  
    <title>while Loop  
</title>  
</head>  
<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>  
</body>  
</html>
```

Output



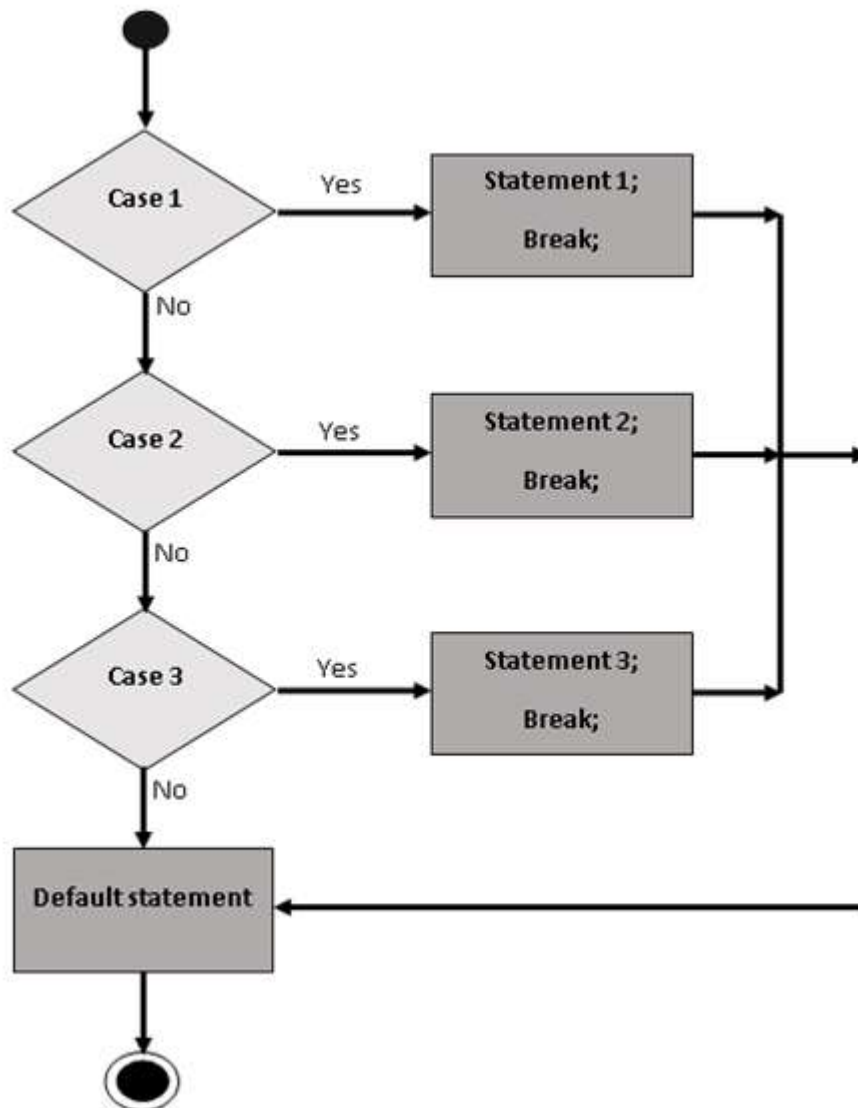
9. SWITCH-CASE

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

You can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

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.

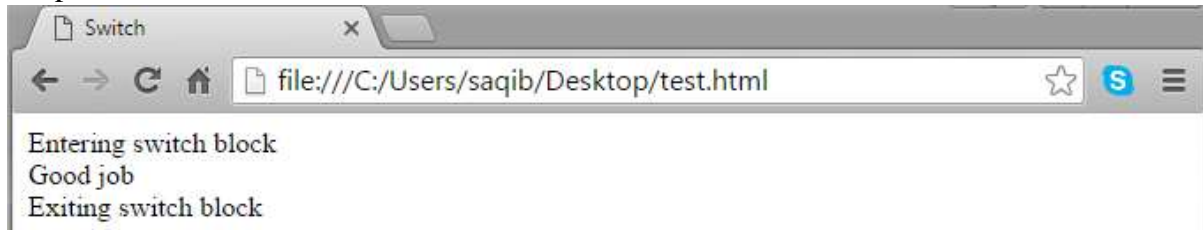
We will explain break statement in Loop Control chapter.

Example

Try the following example to implement switch-case statement.

```
<html>
<head>
    <title>Switch
    </title>
</head>
<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>
</body>
</html>
```

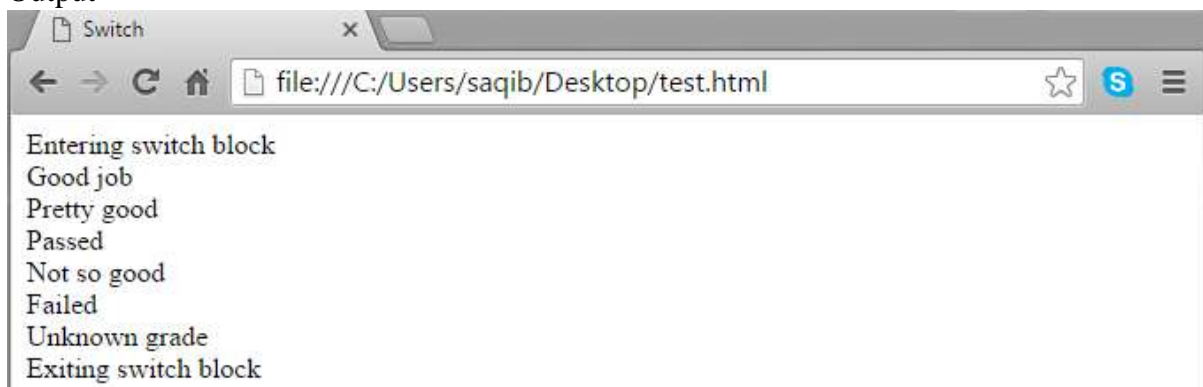
Output



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

```
<html>
<head>
  <title>Switch
</head>
<body>
  <script type="text/javascript">
    var grade = 'A';
    document.write("Entering switch block<br />");
    45
    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>
</body>
</html>
```

Output



10. LOOP CONTROL

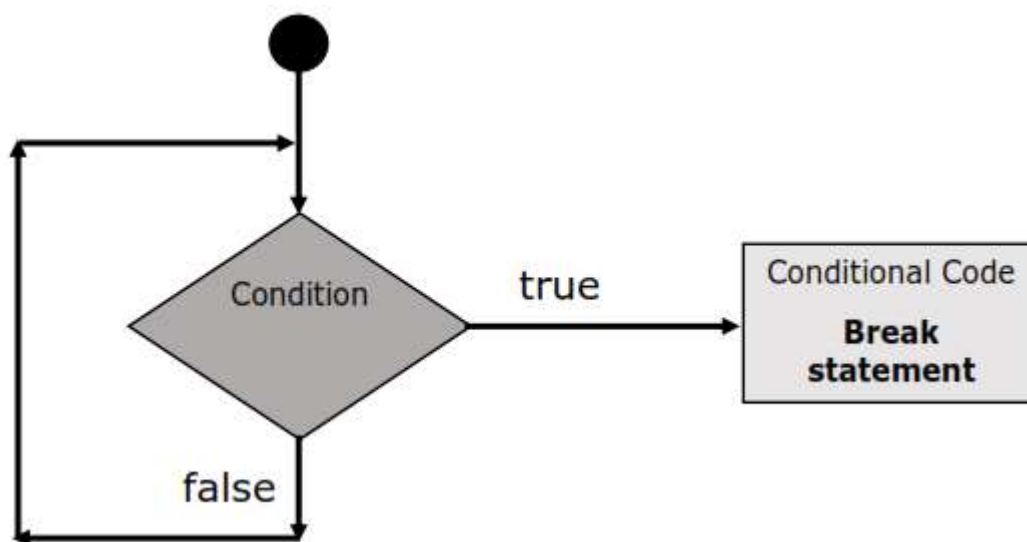
JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

To handle all such situations, JavaScript provides break and continue statements. 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



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>
<head>
  <title>Break
</title>
</head>
<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>
</body>
</html>
```

Output



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

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

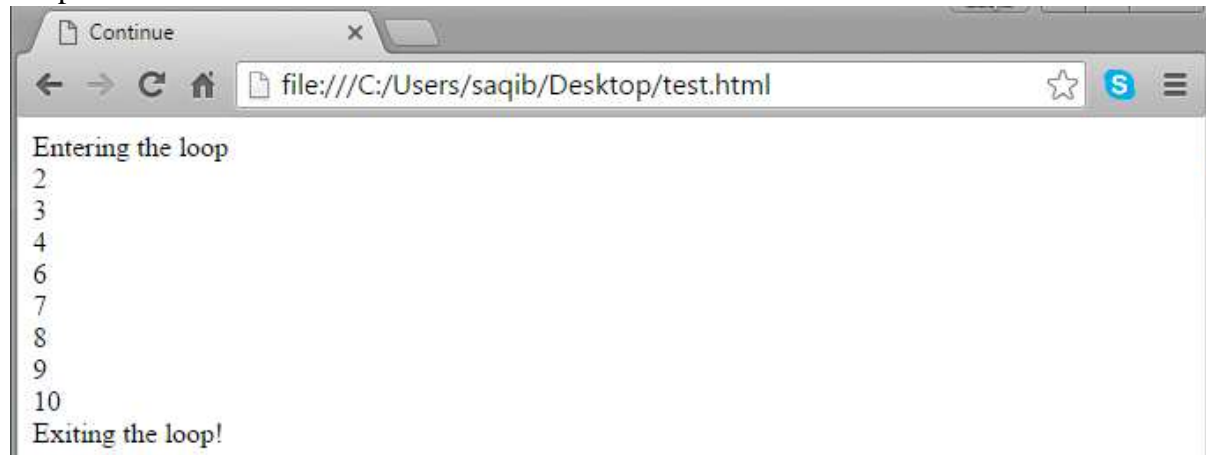
This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5.

```
<html>
<head>
    <title>Continue
    </title>
</head>
<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>
</body>
</html>
```



```
</script>
</body>
</html>
```

Output



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>
<head>
  <title>Using Labels to Control the Flow
</title>
</head>
<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



Example 2

The following example shows how to implement Label with continue.

```

<html>
<head>
    <title>Using Labels to Control the Flow
    </title>
</head>
<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;
                }
            }
        }
    </script>

```

```
        }  
        document.write("Innerloop: " + j + "<br />");  
    }  
    }  
    document.write("Exiting the loop!<br /> ");  
</script>  
</body>  
</html>
```

Output



11. 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

The syntax of 'for..in' loop is:

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

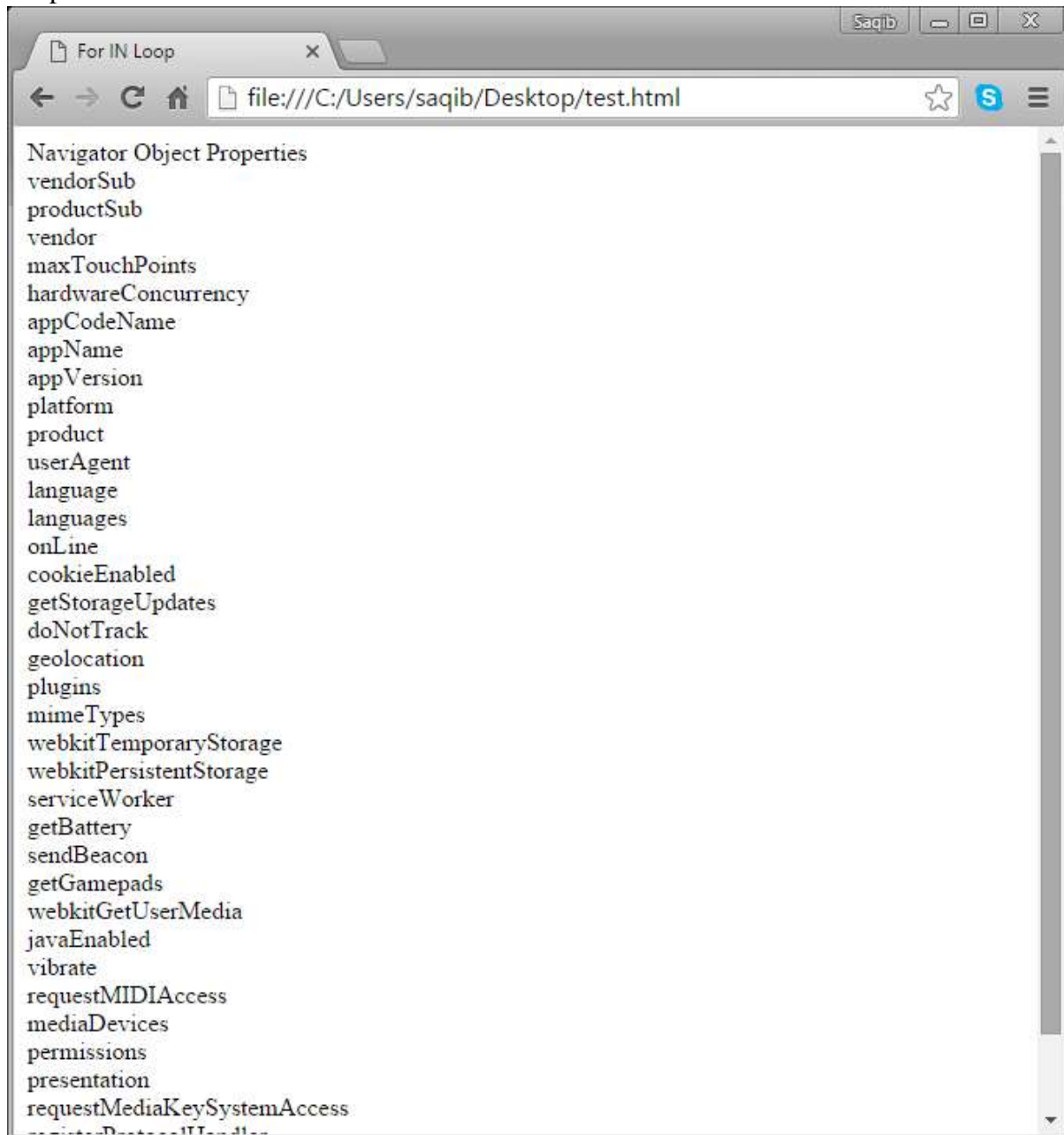
In each iteration, one property from object is assigned to variablename and this loop continues till all the properties of the object are exhausted.

Example

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

```
<html>  
<head>  
    <title>For IN Loop  
    </title>  
</head>  
<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>  
</body>  
</html>
```

Output:



12. FUNCTIONS

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like `alert()` and `write()` in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

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(parameterlist)
    {
        statements
    }
</script>
```

Example

Try the following example. It defines a function called `sayHello` that takes no parameters:

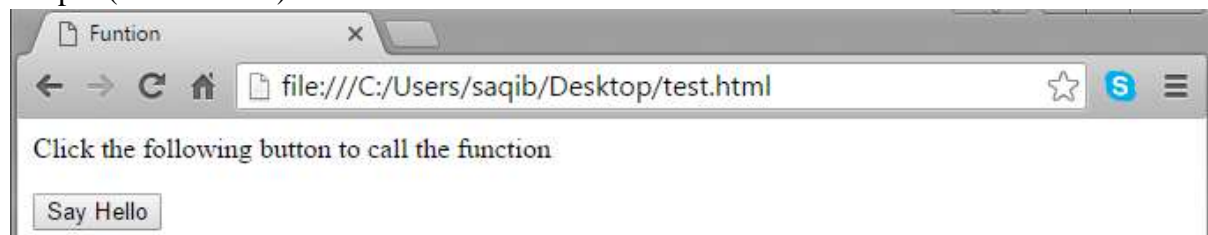
```
<!DOCTYPE html>
<html>
<head>
    <title>Funcations</title>
    <script type="text/javascript">
        function sayHello() {
            alert("Hello there");
        }
    </script>
</head>
<body>
    <p>JavaScriot funcations</p>
</body>
</html>
```

Calling a Function

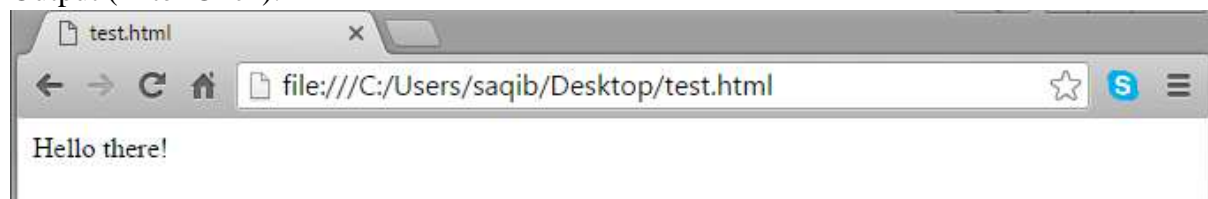
To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<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>
```

Output (before Click):



Output (After Click):



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

Try the following example. We have modified our sayHello function here. Now it takes two parameters.

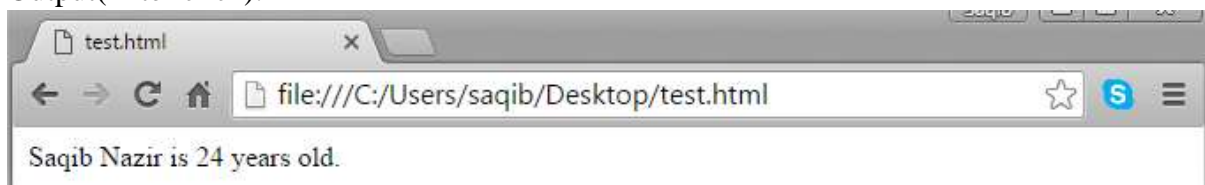
```
<html>
```

```
<head>
  <title>Function Parameters</title>
  <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('Saqib Nazir', 24)"
value="Say Hello">
  </form>
  <p>Use different parameters inside the function and then
try...</p>
</body>
</html>
```

Output (Before Click):



Output(After click):



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

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
<head>
  <title>The return Statement</title>
  <script type="text/javascript">
```



```
function concatenate(first, last) {  
    var full;  
    full = first + last;  
    return full;  
}  
function secondFunction() {  
    var result;  
    result = concatenate('Saqib ', 'Nazir');  
    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>
```

Output (Before Click):



Output (After Click):



There is a lot to learn about JavaScript functions, however we have covered the most important concepts in this tutorial.

Nested Functions

Prior to JavaScript 1.2, function definition was allowed only in top level global code, but JavaScript 1.2 allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the function statement.

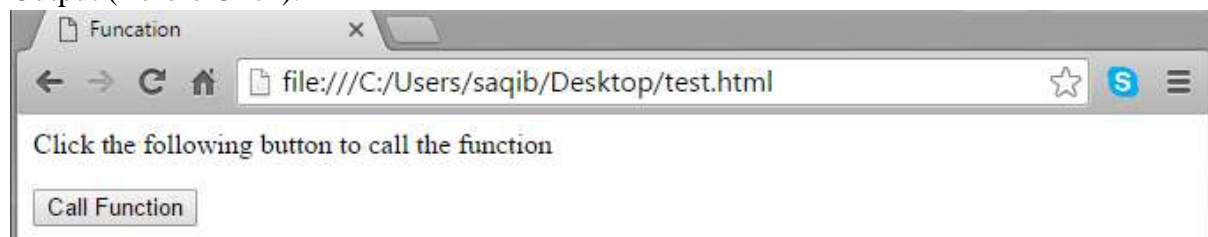
As we'll discuss later in the next chapter, function literals (another feature introduced in JavaScript 1.2) may appear within any JavaScript expression, which means that they can appear within if and other statements.

Example

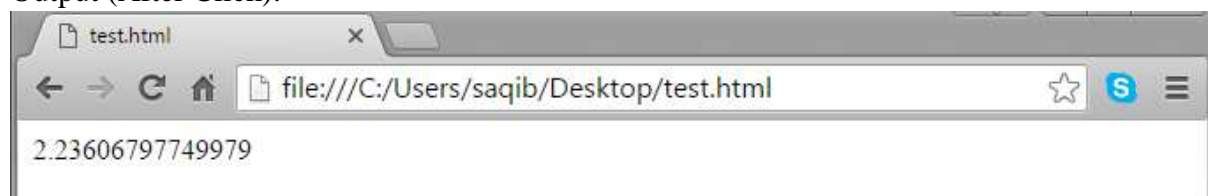
Try the following example to learn how to implement nested functions.

```
<html>
<head>
<title>Funcation</title>
  <script type="text/javascript">
    function hypotenuse(a, b) {
      function square(x) { return x * x; }
      return Math.sqrt(square(a) + square(b));
    }
    function secondFunction() {
      var result;
      result = hypotenuse(1, 2);
      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>
</body>
</html>
```

Output (Before Click):



Output (After Click):



Function () Constructor

The function statement is not the only way to define a new function; you can define your function dynamically using Function() constructor along with the new operator.

Note: Constructor is a terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.

Syntax

Following is the syntax to create a function using Function() constructor along with the new operator.

```
<script type="text/javascript">
    var variablename = new Function(Arg1, Arg2...,
    "FunctionBody");
</script>
```

The Function() constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the Function() constructor is not passed any argument that specifies a name for the function it creates. The unnamed functions created with the Function() constructor are called anonymous functions.

Example

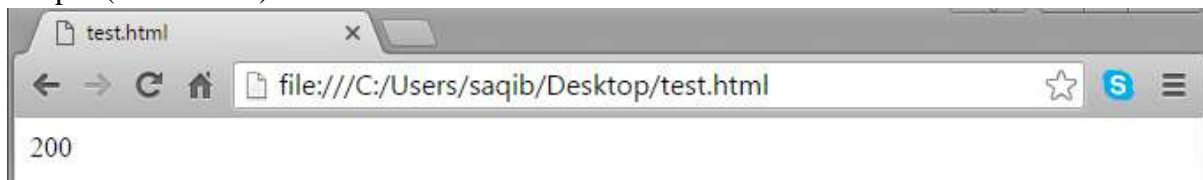
Try the following example.

```
<html>
<head>
    <title>Function () Constructor</title>
    <script type="text/javascript">
        var func = new Function("x", "y", "return x*y;");
        function secondFunction() {
            var result;
            result = func(10, 20);
            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>
</body>
</html>
```

Output (Before Click)



Output (After Click)



Function Literals

JavaScript 1.2 introduces the concept of function literals which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

Syntax

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 variablename = function(Argument List){
        Function Body
    };
</script>
```

Syntactically, you can specify a function name while creating a literal function as follows.

```
<script type="text/javascript">
    var variablename = function FunctionName(Argument List){
        Function Body
    };
</script>
```

But this name does not have any significance, so it is not worthwhile.

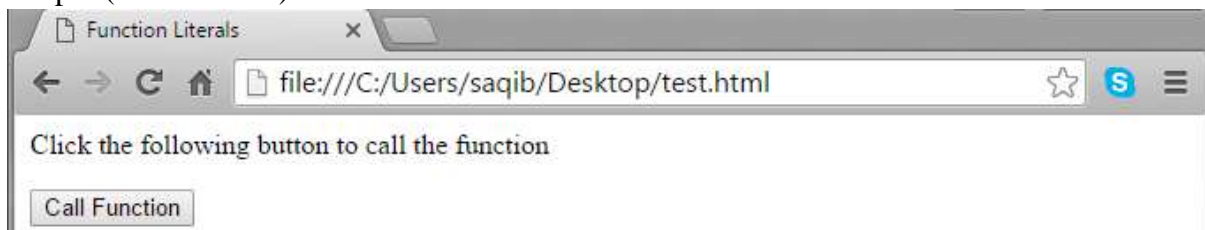
Example

Try the following example. It shows the usage of function literals.

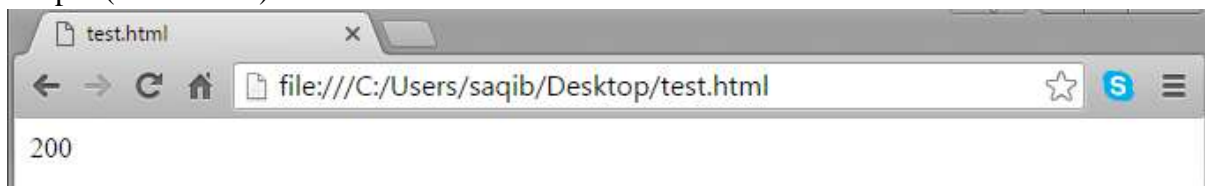
```
<html>
<head>
    <title>Function Literals</title>
    <script type="text/javascript">
        var func = function (x, y) { return x * y };
        function secondFunction() {
            var result;
            result = func(10, 20);
```

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

Output (Before Click)



Output (After Click)



13. OBJECTS

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers:

- **Encapsulation:** the capability to store related information, whether data or methods, together in an object.
- **Aggregation:** the capability to store one object inside another object.
- **Inheritance:** the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism:** the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is:

```
objectName.objectProperty = propertyValue;
```

For example: The following code gets the document title using the "title" property of the document object.

```
var str = document.title;
```

Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the this keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example: Following is a simple example to show how to use the write() method of document object to write any content on the document.

```
document.write ("This is test");
```

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called Object.

The new Operator

The new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();  
  
var books = new Array("C++", "Perl", "Java");  
  
var day = new Date("August 15, 1947");
```

The Object () Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object() constructor is assigned to a variable.

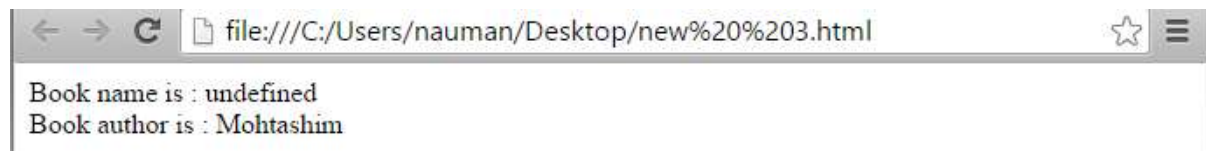
The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

Example 1:

Try the following example; it demonstrates how to create an Object.

```
<html>  
<head>  
  <title>User-defined objects</title>  
  <script type="text/javascript">  
    var book = new Object(); // Create the object book.subject  
= "Perl"; // Assign properties to the object  
    book.author = "Mohtashim";  
  </script>  
</head>  
<body>  
  <script type="text/javascript">  
    document.write("Book name is : " + book.subject + "<br>");  
    document.write("Book author is : " + book.author + "<br>");  
  </script>  
</body>  
</html>
```

Result



Example 2

This example demonstrates how to create an object with a User-Defined Function. Here this keyword is used to refer to the object that has been passed to a function.

```
<html>
<head>
  <title>User-defined objects</title>
  <script type="text/javascript">
    function book(title, author) {
      this.title = title;
      this.author = author;
    }
  </script>
</head>
<body>
  <script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author +
" <br>");
  </script>
</body>
</html>
```

Result



Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

Try the following example; it shows how to add a function along with an object.

```
<html>
<head>
  <title>User-defined objects</title>
  <script type="text/javascript">
    // Define a function which will work as a method
    function addPrice(amount) {
```



```
        this.price = amount;
    }
    function book(title, author) {
        this.title = title; this.author = author;
        this.addPrice = addPrice; // Assign that method as
property.
    }
</script>
</head>
<body>
    <script type="text/javascript">
        var myBook = new book("Perl", "Mohtashim");
        myBook.addPrice(100);
        document.write("Book title is : " + myBook.title + "<br>");
        document.write("Book author is : " + myBook.author + "<br>");
        document.write("Book price is : " + myBook.price + "<br>");
    </script>
</body>
</html>
```

Result



The 'with' Keyword

The 'with' keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to with becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows:

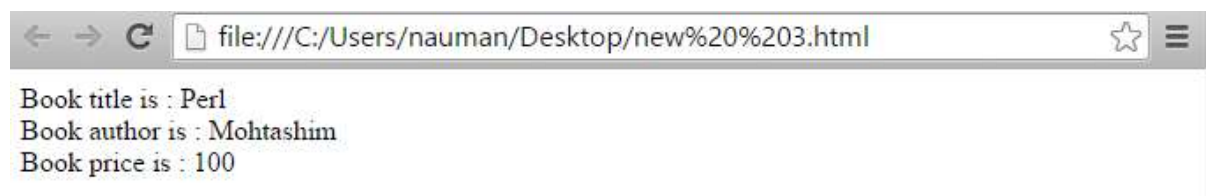
```
with (object){
    properties used without the object name and dot
}
```

Example

```
<html>
<head>
    <title>User-defined objects</title>
```

```
<script type="text/javascript">
    // Define a function which will work as a method
    function addPrice(amount) {
        with (this) {
            price = amount;
        }
    }
    function book(title, author) {
        this.title = title; this.author = author; this.price =
0;
        this.addPrice = addPrice; // Assign that method as
property.
    }
</script>
</head>
<body>
    <script type="text/javascript">
        var myBook = new book("Perl", "Mohtashim");
        myBook.addPrice(100);
        document.write("Book title is : " + myBook.title + "<br>");
document.write("Book author is : " + myBook.author + "<br>");
document.write("Book price is : " + myBook.price + "<br>");
    </script>
</body>
</html>
```

Result



14. ERRORS AND EXCEPTIONS

There are three types of errors in programming:

- Syntax Errors
- Runtime Errors
- Logical Errors

Syntax Errors

Syntax errors, also called parsing errors, occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type="text/javascript">  
    window.print(  
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime Errors

Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type="text/javascript">  
    window.printme();  
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch construct as well as the throw operator to handle exceptions.

You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax:

```
<script type="text/javascript">
  try
  { // Code to run
    [break;]
  } catch ( e ) {
    // Code to run if an exception occurs
    [break;]
  } [ finally {
    // Code that is always executed regardless of
    // an exception occurring
  } ]
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in e and the catch block is executed. The optional finally block executes unconditionally after try/catch.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Error
</title>
</head>
<body>
  <p id="demo"></p>
  <script>
    try {
      adddler("Welcome guest!");
    }
    catch (err) {
      document.getElementById("demo").innerHTML =
err.message;
    }
  </script>
</body>
</html>
```

Output:



The throw Statement

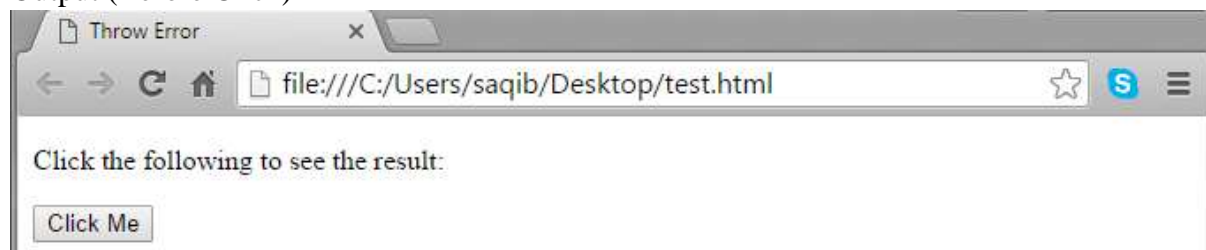
You can use a throw statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Example

The following example demonstrates how to use a throw statement.

```
<!DOCTYPE html>
<html>
<head>
  <title>Finally Block
</title>
  <script type="text/javascript">
    function myFunc() {
      var a = 100;
      var b = 0;
      try {
        if (b == 0) {
          throw ("Divide by zero error.");
        } else {
          var c = a / b;
        }
      } catch (e) {
        document.write("Error: " + e);
      }
    }
  </script>
</head>
<body>
  <p>Click the following to see the result:</p>
  <form>
    <input type="button" value="Click Me" onclick="myFunc();" />
  </form>
</body>
</html>
```

Output (Before Click)



Output (After Click)



You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a try...catch block.

Finally Statement

You can use a finally block which will always execute unconditionally after the try/catch. Here is an example.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Finally Block
</title>
</head>
<body>
  <p>Please input a number between 5 and 10:</p>
  <input id="demo" type="text">
  <button type="button" onclick="myFunction()">Test
  Input</button>
  <p id="message"></p>
  <script>
    function myFunction() {
      var message, x;
      message = document.getElementById("message");
      message.innerHTML = "";
      x = document.getElementById("demo").value;
      try {
        if (x == "") throw "is empty";
        if (isNaN(x)) throw "is not a number";
        x = Number(x);
        if (x > 10) throw "is too high";
        if (x < 5) throw "is too low";
      }
      catch (err) {
        message.innerHTML = "Input " + err;
      }
      finally {
        document.getElementById("demo").value = "";
      }
    }
  </script>
</body>
</html>
```

```
</script>
</body>
</html>
```

Output when user enter less than 5:



Output when user enter more than 5:



Output when user enter any Character:



The onerror() Method

The onerror event handler was the first feature to facilitate error handling in JavaScript. The error event is fired on the window object whenever an exception occurs on the page.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>The onerror( ) Method
  </title>
  <script type="text/javascript">
    window.onerror = function () {
      document.write("An error occurred.");
    }
  </script>
```

```

</head>
<body>
  <p>Click the following to see the result:</p>
  <form>
    <input type="button" value="Click Me" onclick="myFunc();"
  />
  </form>
</body>
</html>

```

Output (Before Click)



Output (After Click)



The onerror event handler provides three pieces of information to identify the exact nature of the error:

- Error message: The same message that the browser would display for the given error
- URL: The file in which the error occurred
- Line number: The line number in the given URL that caused the error

Here is the example to show how to extract this information.

Example

```

<!DOCTYPE html>
<html>
<head>
  <title>The onerror( ) Method
  </title>
  <script type="text/javascript">
    window.onerror = function (msg, url, line) {
      document.write("Message : " + msg); document.write("url : "
+ url);
      document.write("Line number : " + line);
    }
  </script>
</head>
<body>

```

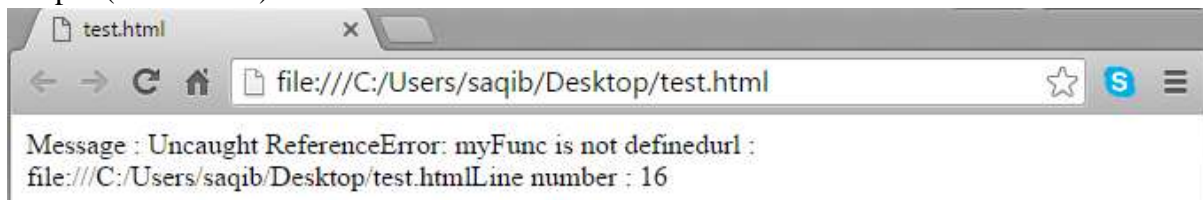


```
<p>Click the following to see the result:</p>
<form>
  <input type="button" value="Click Me" onclick="myFunc();"
/>
</form>
</body>
</html>
```

Output (Before Click)



Output (After Click)



You can display extracted information in whatever way you think it is better.

15. PAGE REDIRECT

What is Page Redirection?

You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to page redirection. This concept is different from JavaScript Page Refresh.

There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons:

- You did not like the name of your domain and you are moving to a new one. In such a scenario, you may want to direct all your visitors to the new site. Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.
- You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.
- The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

JavaScript Page Refresh

You can refresh a web page using JavaScript location.reload method. This code can be called automatically upon an event or simply when the user clicks on a link. If you want to refresh a web page using a mouse click, then you can use the following code:

```
<a href="javascript:location.reload(true)">Refresh Page</a>
```

Auto Refresh

You can also use JavaScript to refresh the page automatically after a given time period. Here setTimeout() is a built-in JavaScript function which can be used to execute another function after a given time interval.

Example

Try the following example. It shows how to refresh a page after every 5 seconds. You can change this time as per your requirement.

```
<html>
<head>
  <title>Error</title>
  <script type="text/JavaScript">
    function AutoRefresh(t) {
      setTimeout("location.reload(true);", t);
    }
  </script>
</head>
```

```
<body onload="JavaScript:AutoRefresh(5000);">
  <p>This page will refresh every 5 seconds.</p>
</body>
</html>
```

How Page Re-direction Works?

The implementations of Page-Redirection are as follows.

Example 1

It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```
<html>
<head>
  <title>How Page Re-direction Works?</title>
  <script type="text/javascript">
    function Redirect() {
      window.location = "http://www.kics.edu.pk";
    }
  </script>
</head>
<body>
  <p>Click the following button, you will be redirected to home
page.</p>
  <form>
    <input type="button" value="Redirect Me"
onclick="Redirect();" />
  </form>
</body>
</html>
```

Output (Before Click):



Output (After Click)



Example 2

You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. The following example shows how to implement the same. Here `setTimeout()` is a built-in JavaScript function which can be used to execute another function after a given time interval.

```
<html>
<head>
  <title>How Page Re-direction Works?</title>
  <script type="text/javascript">
    function Redirect() {
      window.location = "http://www.kics.edu.pk";
    }
    document.write("You will be redirected to our main page in 10
seconds!");
    setTimeout('Redirect()', 10000);
  </script>
</head>
<body>
</body>
</html>
```

Output (Before Redirect)



Output (After Redirect)



16. DIALOG BOX

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

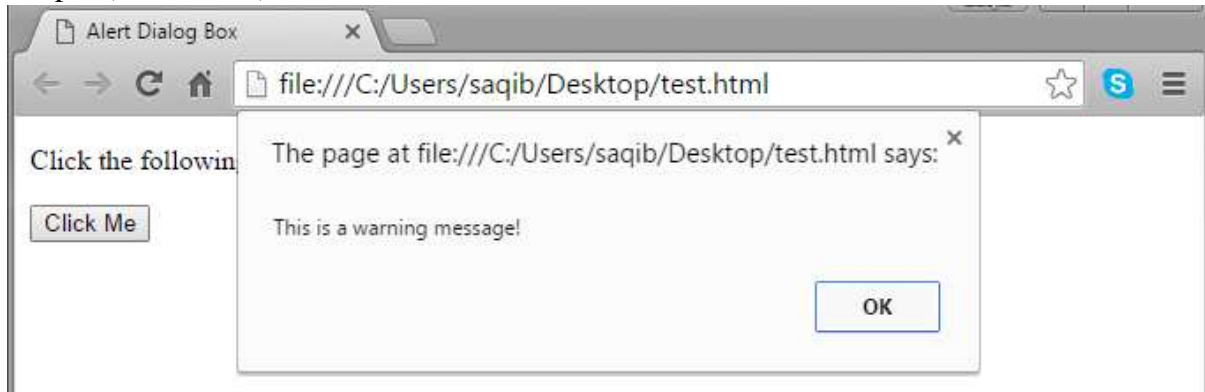
Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Alert Dialog Box
</title>
  <script type="text/javascript">
    function Warn() {
      alert("This is a warning message!");
      document.write("This is a warning message!");
    }
  </script>
</head>
<body>
  <p>Click the following button to see the result: </p>
  <form>
    <input type="button" value="Click Me" onclick="Warn();" />
  </form>
</body>
</html>
```

Output (Before Click)



Output (After Click)



Confirmation Dialog Box

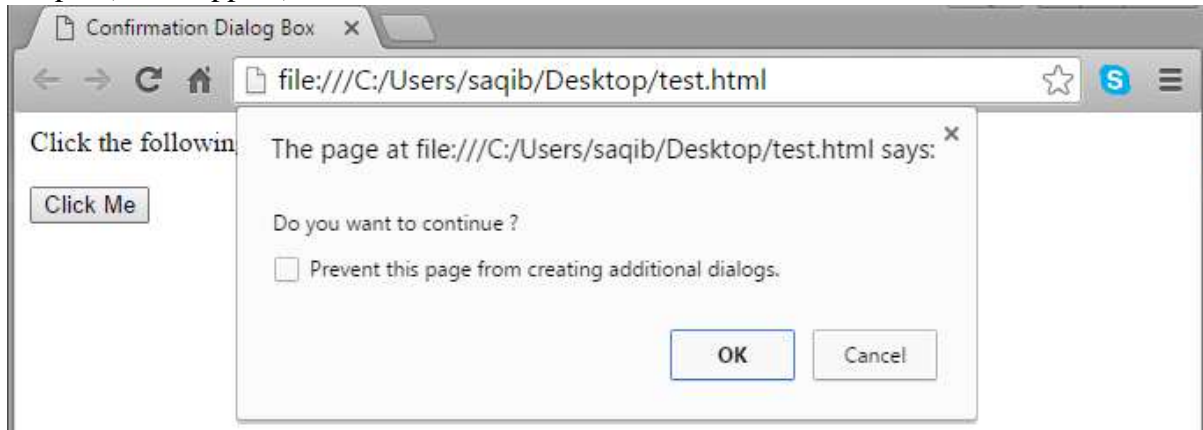
A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method `confirm()` will return `true`. If the user clicks on the Cancel button, then `confirm()` returns `false`. You can use a confirmation dialog box as follows.

Example

```
<html>
<head>
  <title>Confirmation Dialog Box
</title>
  <script type="text/javascript">
    function getConfirmation() {
      var retVal = confirm("Do you want to continue ?");
      if (retVal == true) {
        document.write("User wants to continue!");
        return true;
      } else {
        Document.write("User does not want to continue!");
        return false;
      }
    }
  </script>
</head>
<body>
  <p>Click the following button to see the result: </p>
  <form>
    <input type="button" value="Click Me"
    onclick="getConfirmation();" />
  </form>
</body>
</html>
```

Output (Alert Appear)



Output (Click ok on Alert)



Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called `prompt()` which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method `prompt()` will return the entered value from the text box. If the user clicks the Cancel button the window method `prompt()` returns null.

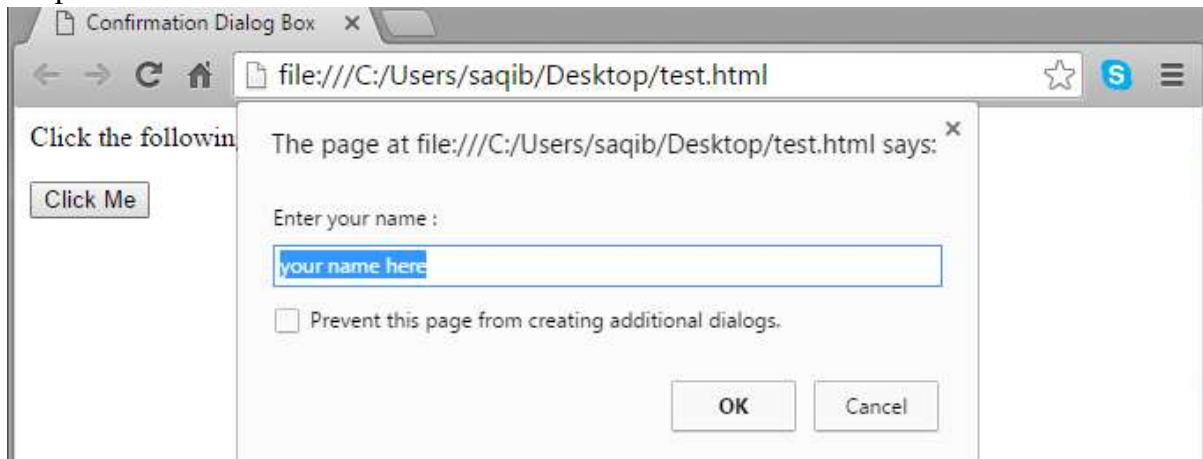
Example

The following example shows how to use a prompt dialog box:

```
<html>
<head>
  <title>Confirmation Dialog Box
  </title>
  <script type="text/javascript">
    function getValue() {
      var retVal = prompt("Enter your name : ", "your name
here");
      document.write("You have entered : " + retVal);
    }
  </script>
</head>
```

```
<body>
  <p>Click the following button to see the result: </p>
  <form>
    <input type="button" value="Click Me"
onclick="getValue();" />
  </form>
</body>
</html>
```

Output



17. jQuery and Its Plugins

What is jQuery?

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto: Write less, do more. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery:

- **DOM manipulation:** The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called Sizzle.
- **Event handling:** The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support:** The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- **Animations:** The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight:** The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- **Cross Browser Support:** The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology:** The jQuery supports CSS3 selectors and basic XPath syntax.

How to use jQuery?

There are two ways to use jQuery.

- **Local Installation:** You can download jQuery library on your local machine and include it in your HTML code.
- **CDN Based Version:** You can include jQuery library into your HTML code directly from Content Delivery Network (CDN).

Local Installation

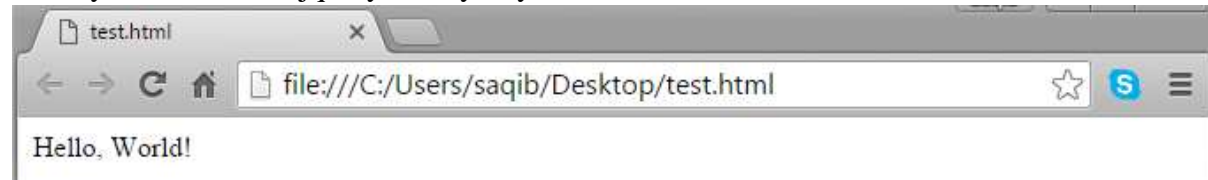
- Go to the <https://jquery.com/download/> to download the latest version available.
- Now, insert downloaded jquery-2.1.3.min.js file in a directory of your website, e.g. /jquery.

Example

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type="text/javascript" src="/jquery/jquery-2.1.3.min.js"></script>
    <script type="text/javascript">
      $(document).ready(function () {
        document.write("Hello, World!");
      });
    </script>
  </head>
</html>
```

```
});  
</script>  
</head>  
<body>  
  <h1>Hello</h1>  
</body>  
</html>
```

Now, you can include jquery library in your HTML file as follows:



CDN Based Version

You can include jQuery library into your HTML code directly from Content Delivery Network (CDN). Google and Microsoft provides content deliver for the latest version.

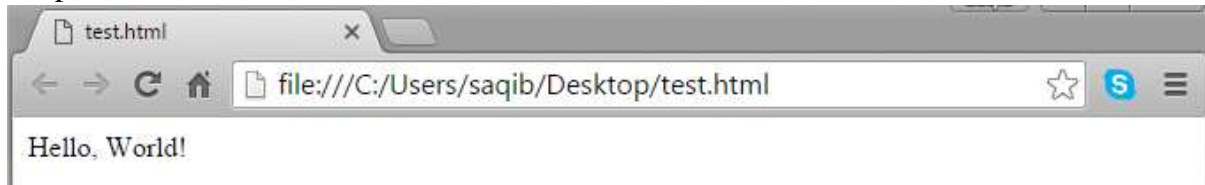
We are using Google CDN version of the library throughout this tutorial.

Example

Now let us rewrite above example using jQuery library from Google CDN.

```
<html>  
<head>  
  <title>The jQuery Example</title>  
  <script type="text/javascript"  
src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"  
>  
  </script>  
  
  <script type="text/javascript">  
    $(document).ready(function () {  
      document.write("Hello, World!");  
    });  
  </script>  
</head>  
  
<body>  
  
  <h1>Hello</h1>  
  
</body>  
</html>
```

Output:



How to call a jQuery Library Functions?

As almost everything, we do when using jQuery reads or manipulates the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the `$(document).ready()` function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded.

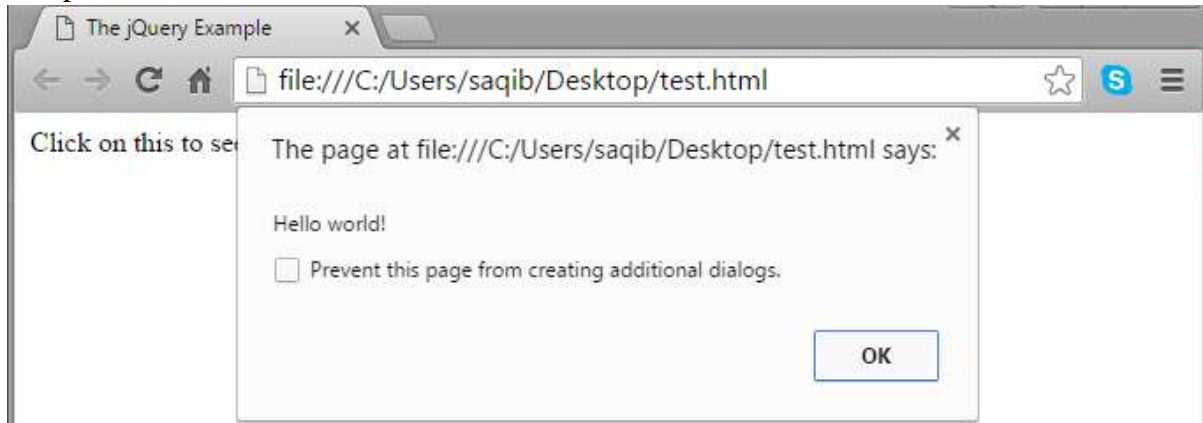
To do this, we register a ready event for the document as follows:

```
$(document).ready(function() {  
    // do stuff when DOM is ready  
});
```

To call upon any jQuery library function, use HTML script tags as shown below:

```
<html>  
<head>  
    <title>The jQuery Example</title>  
    <script type="text/javascript" src="jquery-2.1.4.min.js">  
    </script>  
    <script type="text/javascript" language="javascript">  
        $(document).ready(function () {  
            $("div").click(function () {  
                alert("Hello world!");  
            });  
        });  
    </script>  
</head>  
<body>  
    <div id="newdiv">  
        Click on this to see a dialogue box.  
    </div>  
</body>  
</html>
```

Output:



jQuery slideDown() Method

The jQuery slideDown() method is used to slide down an element.

Syntax:

```
$(selector).slideDown(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the slideDown() method:

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery slideDown() Method
  </title>
  <script src="jquery-2.1.4.min.js"></script>
  <script>
    $(document).ready(function () {
      $("#flip").click(function () {
        $("#panel").slideDown("slow");
      });
    });
  </script>
  <style>
    #panel, #flip {
      padding: 5px;
      text-align: center;
      background-color: #e5eccc;
      border: solid 1px #c3c3c3;
    }

    #panel {
```

```

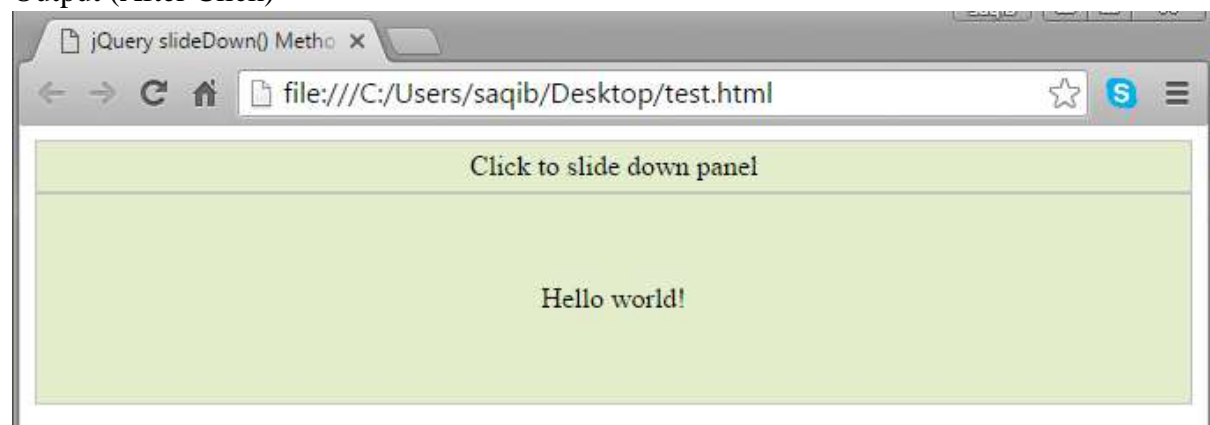
padding: 50px;
display: none;
}
</style>
</head>
<body>
  <div id="flip">Click to slide down panel</div>
  <div id="panel">Hello world!</div>
</body>
</html>

```

Output (Before Click)



Output (After Click)



jQuery slideToggle() Method

The jQuery slideToggle() method toggles between the slideDown() and slideUp() methods.

If the elements have been slid down, slideToggle() will slide them up.

If the elements have been slid up, slideToggle() will slide them down.

```
$(selector).slideToggle(speed,callback);
```

The optional speed parameter can take the following values: "slow", "fast", milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the slideToggle() method:

```

<!DOCTYPE html>
<html>
<head>
  <title>jQuery slideToggle() Method

```

```
</title>
<script src="jquery-2.1.4.min.js"></script>
<script>
    $(document).ready(function () {
        $("#flip").click(function () {
            $("#panel").slideToggle("slow");
        });
    });
</script>

<style>
    #panel, #flip {
        padding: 5px;
        text-align: center;
        background-color: #e5eccc;
        border: solid 1px #c3c3c3;
    }

    #panel {
        padding: 50px;
        display: none;
    }
</style>
</head>
<body>

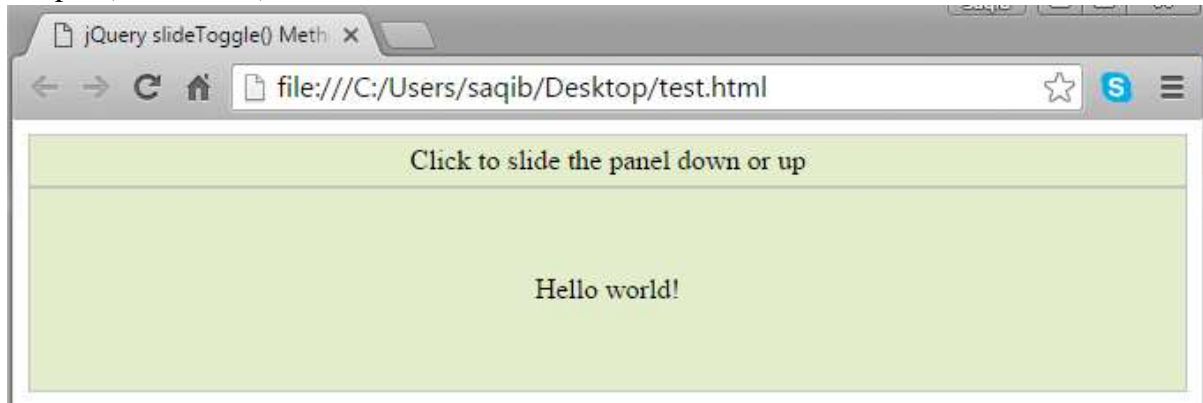
    <div id="flip">Click to slide the panel down or up</div>
    <div id="panel">Hello world!</div>

</body>
</html>
```

Output (Before Click):



Output (After Click)



Output (Again Click):

