JavaScript

What is JavaScript?

- JavaScript is a scripting language. It was designed by Netscape to create interactive and dynamic web pages.
- It is also known as lightweight programming language.
- It is usually embedded directly into HTML pages.
- It is an interpreted language which is executed without complication.

Advantages of JavaScript

1) **It is an interpreted language.**
    - It requires no compilation steps. All steps are interpreted by browser like html.
    - It is interpreted by the scripting engine which is part of browser.
2) **Embedded within HTML**
    - It does not require any special or separate editor for programs to be written.
    - It is written in notepad and saved with .html extension.
3) **Minimal syntax –easy to learn**
    - By learning just few commands and simple rules of syntax JavaScript application can build.

4) Procedural Capabilities

- JavaScript support condition checking, looping, branching etc.

5) Easy debugging and testing

- It is interpreted language so it is tested line by line and if any error is there it will display along with line number.
- So it is easy to locate errors and make changes.

6) Platform Independence

- JavaScript application work on any machine.

Selecting an Developing Environment for JavaScript

- With simple text editor such as notepad, you can create JavaScript program.
- Following are some other tools also

1) Microsoft Frontpage

2) Adobe Deramweaver

3) Adobe GoLive

HTML and JavaScript

There are 3 possibilities to integrate the JavaScript code into the HTML file.

1. Integrated script under the <head> tag
2. Integrated script under the <body> tag
3. Importing the external JavaScript

1. Integrated script under the <head> tag

- The first possibility to incorporate the source code under the <head> tag of HTML file.
- JavaScript in the head section will execute when called.
- Example

```
<html>
<head>
<script type=”text/JavaScript”>
…..
…..
</script>
</head>
```

2. **Integrated script under the <body> tag**

   - When you place JavaScript code under the <body> tag, this generates the contents of the web page.
   - JavaScript code under the <body> tag executes when the web page loads and go in the body section.
   - Example

```
<html>
<head>
</head>
<body>
<script type=”text/JavaScript”>
…..
…..
</script>
</body>
```

3. **Importing the external JavaScript**

   - You can import an external JavaScript file, when you want to run the same JavaScript file on several HTML files, without having to write the same JavaScript code on every HTML file.
   - **External JavaScript file should save with .js extension.**
   - Example

```
<html>
<head>
<script src=”abc.js”>
</script>
</head>
```

First JavaScript Program

- The javascript code used <script>..</script> tags to start and end the code.
- To print something

```
Document.write(“hello”);
```

Example

```
<html>
<head>
</head>
<body>
<script type=”text/JavaScript”>
document.write(“hello”);
</script>
</body>
</html>
```

Elements of JavaScript

1. **JavaScript Statements**

   - JavaScript statements are embedded in between <script>and </script> tags.
   - JavaScript statements are the single line codes in between the <script>and </script> tags.
   - JavaScript statements are separated by **semicolons**.

   ```
   <script type="text/JavaScript">
   document.write("hello");
   </script>
   ```

2. **JavaScript Statement Block**

   - Several statements grouped together in a statement block. The purpose of statements block is to make the sequence of statements execute together.
   - **The statement block begins and ends with open and close brackets.**

   Example

   ```
   <script type="text/JavaScript">
   {
   document.write("<h1>This is a header</h1>");
   document.write("<p>This is a paragraph</p>");
   document.write("<h2>This is a header2</h2>");
   } </script>
   ```

3. JavaScript Comments

- JavaScript comments can be used to explain JavaScript code, and to make it more readable.

1. Single-Line Comments

- Single line comments start with //.
- Any text between // and the end of the line, will be ignored by JavaScript (will not be executed).

2. Multi-line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored by JavaScript.

Variables

- JavaScript variables are containers for storing data values.
- Variables are temporary store data and have a name, value and memory address.
- **The general rules** for constructing names for variables (unique identifiers) are:

1. Names can contain letters, digits, underscores, and dollar signs.
2. Names must begin with a letter
3. Names can also begin with $ and _ (but we will not use it in this tutorial)
4. Names are case sensitive (y and Y are different variables)
5. Reserved words (like JavaScript keywords) cannot be used as names

- **Declaring Variables**
  - Before you can use variable for storing any data, it has to be declared.
  - **Syntax**

    Var variable name;

- **Assigning values and Accessing Variables**
  - You can assign values to variables while declaring them or after declaring them.
  - Once you assign value to a variable, you can access the value and use it.
  - Syntax

    Var variable name=value;

Html Code

```
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables</p>
<script>
var x = 5;
var y = 6;
var z = x + y;
```

```
document.write("The value is:", z);
</script>
</body>
</html>
```

Data Types

- Data types denoted the kind of data.
- Depending on the type of data, the data is processed in different ways by JavaScript.
- The basic data types available in JavaScript are string, number and Boolean.

1. String Data type

- A string (or a text string) is a series of characters like "abc".
- Strings are written with quotes. You can use single or double quotes.
- You can make the use of escape character.

Code	Outputs
\'	single quote
\"	double quote
\\	backslash

\n	new line
\r	carriage return
\t	tab
\b	backspace

Example:

```
<html>

<body>

<script>

var carName1 = "maruti";

var carName2 = 'audi';

document.write("car name1" + carName1+"<br>"+"car
name2"+carName2);

</script>

</body>

</html>
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
<html>
<body>
<script>
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';
document.write(answer1 + "<br>" + answer2 + "<br>" + answer3);
</script>
</body>
</html>
```

2. Number Data type

- JavaScript has only one type of numbers.

- Numbers can be written with, or without decimals

Example

```
var x1 = 34.00;     // Written with decimals
var x2 = 34;        // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

HTML Code

```
<html>
<body>
<script>
var x1 = 34.00;
var x2 = 34;
var y = 123e5;
var z = 123e-5;
document.write (x1 + "<br>" + x2 + "<br>" + y + "<br>" + z);
</script>
</body>
</html>
```

3. Boolean Data type

- Booleans can only have two values: true or false.
- Booleans are often used in conditional testing.

Example

```
var x = true;
var y = false;
```

Operators

- JavaScript operators are symbols that are used to perform operations on operands.
- JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.
- There are following types of operators in JavaScript.

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

1. Arithmetic Operators

- Arithmetic operators are used to perform arithmetic operations on the operands.
- The following operators are known as JavaScript arithmetic operators.
- Given that y = 5, the table below explains the arithmetic operators:

Operator	Description	Example	Result in y	Result in x
+	Addition	x = y + 2	y = 5	x = 7
-	Subtraction	x = y - 2	y = 5	x = 3
*	Multiplication	x = y * 2	y = 5	x = 10
/	Division	x = y / 2	y = 5	x = 2.5
%	Modulus (Remainder)	x = y % 2	y = 5	x = 1
++	Increment	x = ++y	y = 6	x = 6
		x = y++	y = 6	x = 5
--	Decrement	x = --y	y = 4	x = 4
		x = y--	y = 4	x = 5

2. Assignment Operators

- Assignment operators are used to assign values to JavaScript variables.
- Given that x = 10 and y = 5, the table below explains the assignment operators:

Operator	**Example**	**Same As**	**Result in x**
=	x = y	x = y	x = 5
+=	x += y	x = x + y	x = 15
-=	x -= y	x = x - y	x = 5

*=	x *= y	x = x * y	x = 50
/=	x /= y	x = x / y	x = 2
%=	x %= y	x = x % y	x = 0

3. Logical Operators

- Logical operators are used to determine the logic between variables or values.
- Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

4. 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
!=	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

5. Bitwise Operators

- Bit operators work on 32 bits numbers. Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.
- The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND It performs a Boolean AND operation on each bit of its integer arguments.	(10==20 & 20==33) = false
|	Bitwise OR It performs a Boolean OR operation	(10==20 | 20==33) =

	on each bit of its integer arguments.	false
^	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.	(10==20 ^ 20==33) = false
~	Bitwise NOT It is a unary operator and operates by reversing all the bits in the operand.	(~10) = -10
<<	Bitwise 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.	(10<<2) = 40
>>	Bitwise Right Shift Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	(10>>2) = 2

>>>	Bitwise Right Shift with Zero This operator is just like the >> operator, except that the bits shifted in on the left are always zero.	(10>>>2) = 2

6. Conditional (Ternary) Operator

The conditional operator assigns a value to a variable based on a condition.

Syntax	Example	Description
variablename = (condition) ? value1:value2	voteable = (age < 18) ? "Too young":"Old enough";	If the variable "age" is a value below 18, the value of the variable "voteable" will be "Too young", otherwise the value of voteable will be "Old enough".

7. The typeof Operator

- The **typeof** operator returns the type of a variable, object, function or expression:

Example

```
typeof "John"                 // Returns string
typeof 3.14                   // Returns number
typeof false                  // Returns boolean
typeof new Date()             // Returns object
typeof function () {}         // Returns function
typeof myCar                  // Returns undefined (if myCar is not
```

```
declared)
typeof null                    // Returns object


<html>

<body>

<p>The typeof operator returns the type of a variable, object,
function or expression.</p>

<script>

document.write (

    typeof "John" + "<br>" +

    typeof 3.14 + "<br>" +

     typeof false + "<br>" +

     typeof new Date() + "<br>" +

    typeof function () {} + "<br>" +

    typeof myCar + "<br>" +

    typeof null)

</script>


</body>

</html>
```

8. The delete Operator

- The **delete** operator deletes a property from an object
- After deletion, the property cannot be used before it is added back again.
- The delete operator deletes both the value of the property and the property itself.

Example

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
delete person.age;   // or delete person["age"];
```

```
<html>
<body>
<p>The delete operator deletes a property from an object.</p>
<script>
var person = {
   firstname:"John",
   lastname:"Doe",
   age:50,
   eyecolor:"blue"
};
delete person.age;
```

```
document.write(person.firstname + " is " + person.age + " years old.")

</script>


</body>

</html>
```

Flow Control Statements

If-else

The JavaScript if-else statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

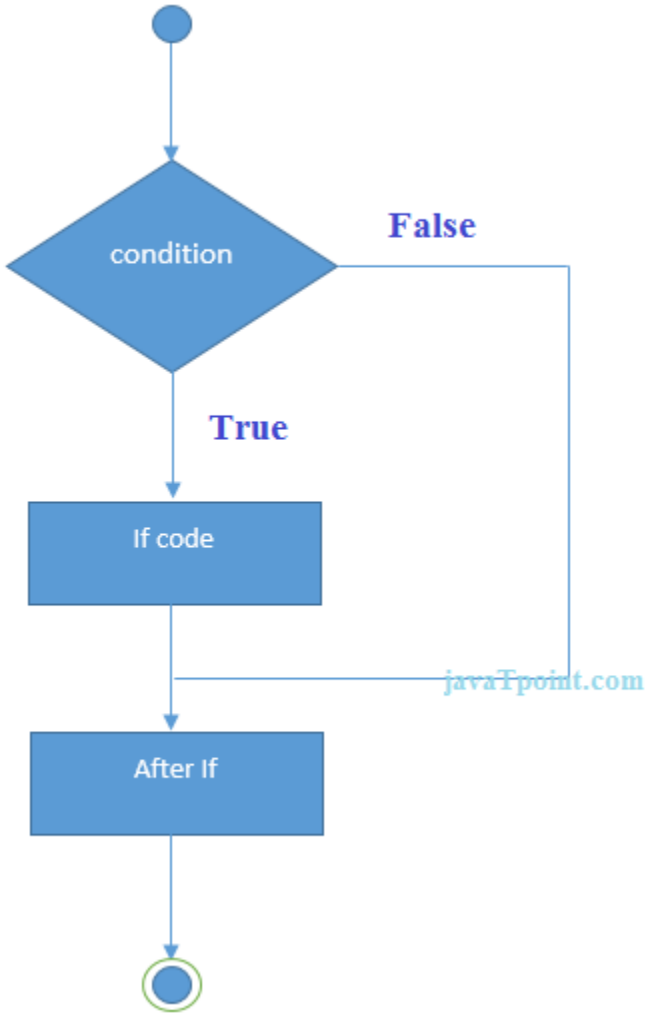
1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
if(expression){

//content to be evaluated

}
```

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



Let's see the simple example of if statement in javascript.

```
<script>
var a=20;
if(a>10){
    document.write("value of a is greater than 10");
}
```

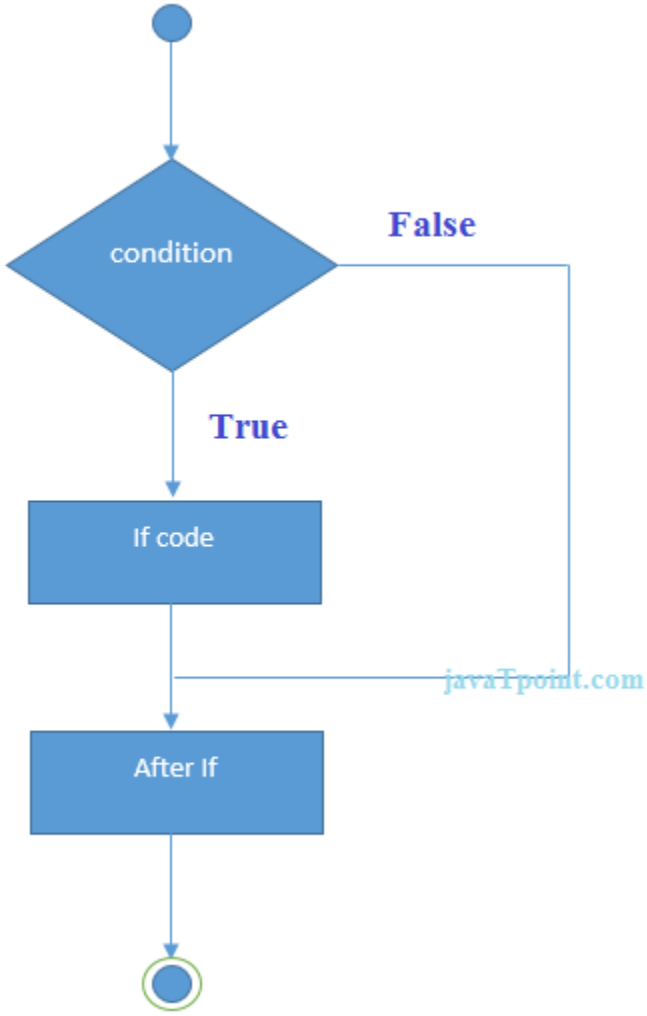
</script>

JavaScript If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
if(expression){

//content to be evaluated if condition is true

}

else{

//content to be evaluated if condition is false

}
```

Flowchart of JavaScript If...else statement



Let's see the example of if-else statement in JavaScript to find out the even or odd number.

```
<script>

var a=20;

if(a%2==0){

document.write("a is even number");

}
```

```
else{
document.write("a is odd number");
}
</script>
```

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){
//content to be evaluated if expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
```

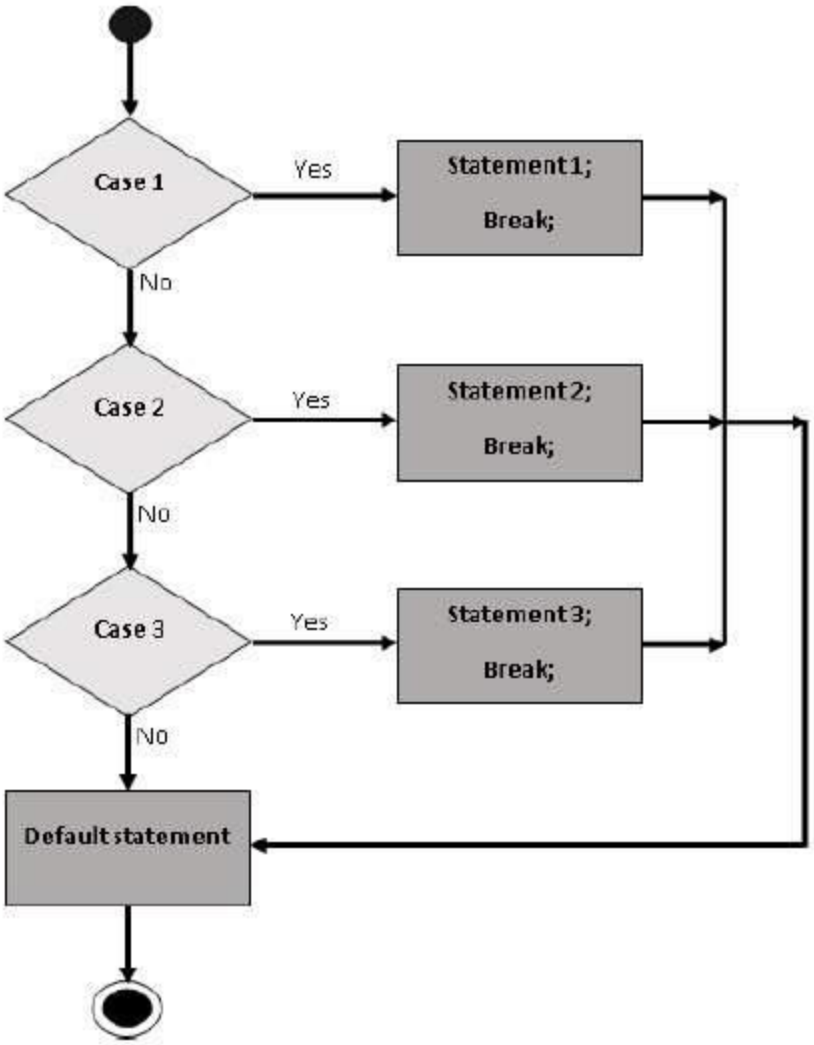
Let’s see the simple example of if else if statement in javascript.

```
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20");
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
```

Switch Case

Flow Chart

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



Syntax

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

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

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

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

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

Example

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var grade='A';
        document.write("Entering switch block<br />");
```

```
          switch (grade)
          {
            case 'A': document.write("Good job<br />");
            break;

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

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

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

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

            default:  document.write("Unknown grade<br />")
          }
          document.write("Exiting switch block");
        //-->
      </script>

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

While Loops

The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Syntax

The syntax of while loop in JavaScript is as follows –

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

Example

```
<html>
  <body>

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

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

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

  </body>
</html>
```

The do...while Loop

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

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

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

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

Example

```
<html>
  <body>

     <script type="text/javascript">
          var count = 0;

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

          while (count < 5);
          document.write ("Loop stopped!");
     </script>
        </body>
</html>
```

For Loop

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

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

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

Syntax

The syntax of for loop is JavaScript is as follows −

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

Example

```
<html>
  <body>

    <script type="text/javascript">

        var count;
        document.write("Starting Loop" + "<br />");

        for(count = 0; count < 10; count++){
```

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

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

        </body>
</html>
```

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 look.

- 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.

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>

  </body>
</html>
```

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>
  <body>

    <script type="text/javascript">

        var x = 1;
        document.write("Entering the loop<br /> ");

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

          if (x == 5){
            continue; // skill rest of the loop body
          }
          document.write( x + "<br />");
        }

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

    </script>

  </body>
</html>
```

Using Labels to Control the Flow

- 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.

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>
```

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>
```