JavaScript II

Condition, Iteration, Function

(Based on Tutorialpoints.com)

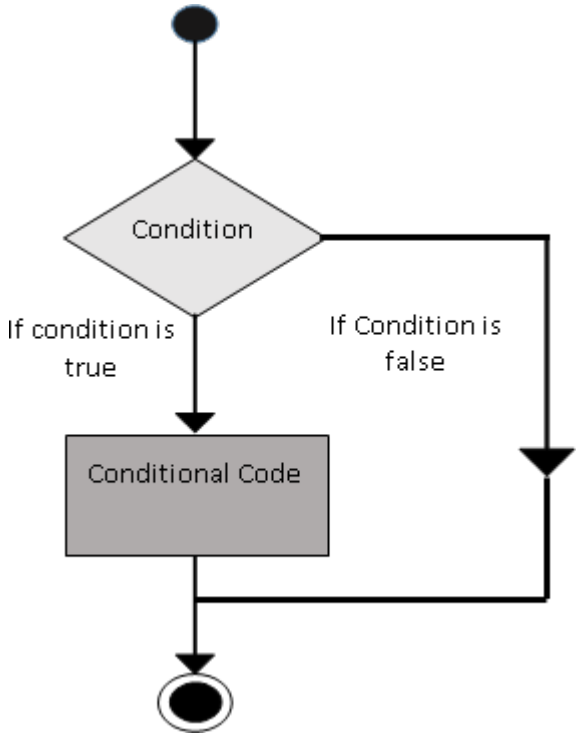
1. IF-ELSE

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

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

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

```
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 inJavaScript

```
<html>
<body>

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

if( age > 18 ){
   document.write("<b>Qualifies for driving</b>");
}else{
   document.write("<b>Does not qualify for driving</b>");
}
//-->
</script>

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

Output

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

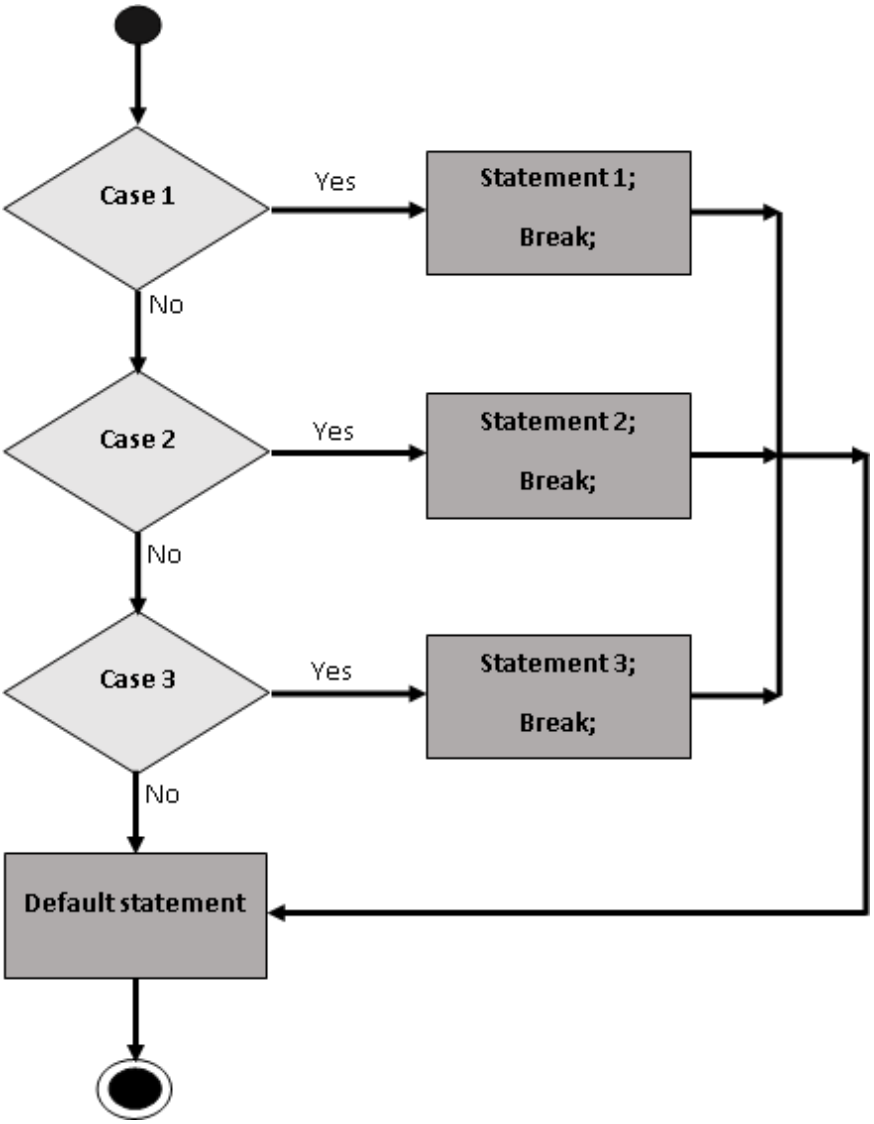
2. SWITCH-CASE

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

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

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

```
            break;
  case 'B': document.write("Pretty good<br />");
            break;
  case 'C': document.write("Passed<br />");
            break;
  case 'D': document.write("Not so good<br />");
            break;
  case 'F': document.write("Failed<br />");
            break;
  default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>

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

Output

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

```
<html>
<body>

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

```
switch (grade)
{
  case 'A': document.write("Good job<br />");
  case 'B': document.write("Pretty good<br />");
  case 'C': document.write("Passed<br />");
  case 'D': document.write("Not so good<br />");
  case 'F': document.write("Failed<br />");
  default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>

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

Output

3. 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.

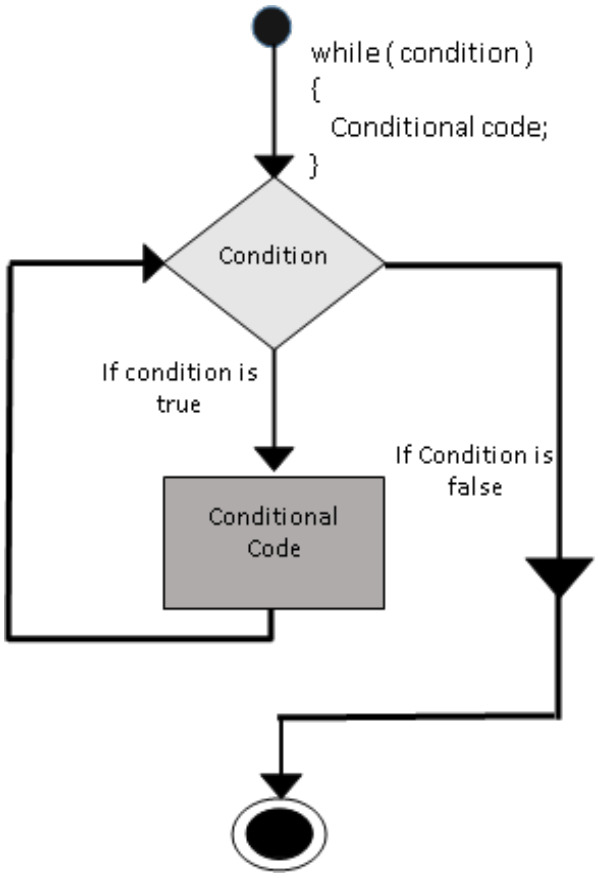
JavaScript supports all the necessary loops to ease down the pressure of programming.

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>
<body>

<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop ");
while (count < 10){
  document.write("Current Count : " + count + "<br />");
  count++;
}
document.write("Loop stopped!");
//-->
</script>

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

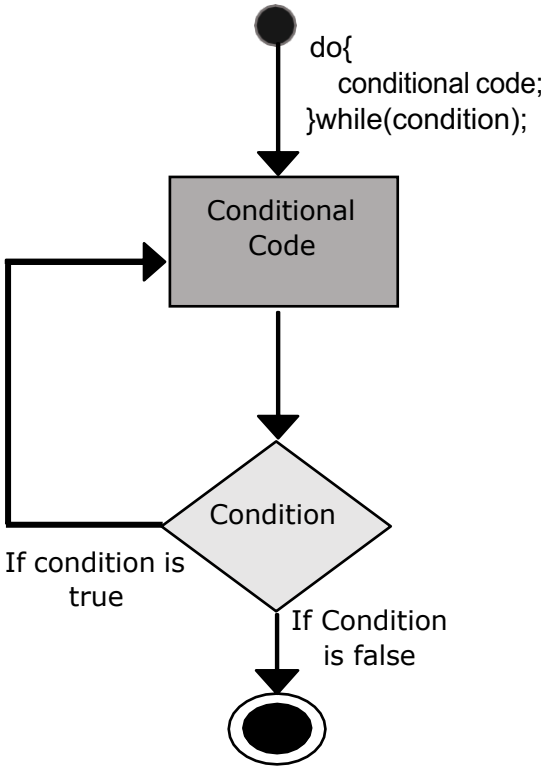
Output

The do...while Loop

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

Flow Chart

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



Syntax

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

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

Note: Don't miss the semicolon used at the end of the **do...while** loop.

Example

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

```
<html>
<body>

<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
  document.write("Current Count : " + count + "<br />");
  count++;
}while (count < 5);
document.write ("Loop stopped!");
//-->
</script>

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

Output

4. FOR LOOP

The for Loop

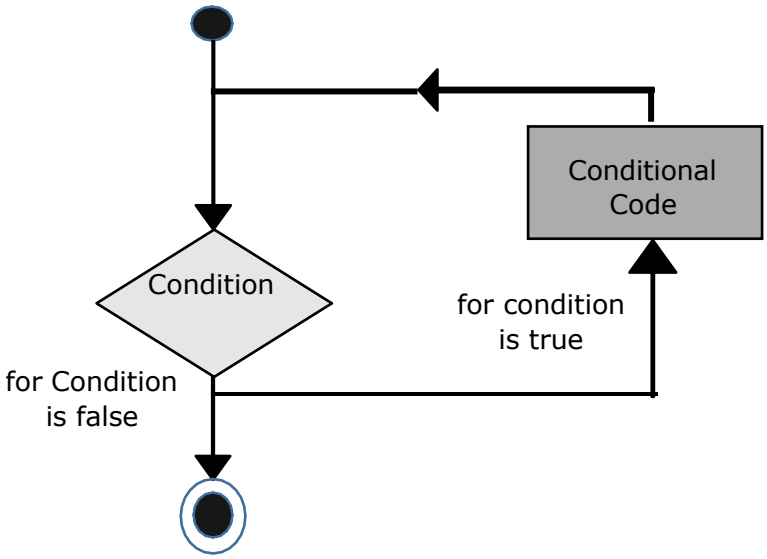
The '**for**' loop is the most compact form of looping. It includes the following three important parts:

- 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 of a **for** loop in JavaScript would be as follows:



Syntax

The syntax of **for** loop is JavaScript is as follows:

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

Example

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

```
<html>
<body>

<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
  document.write("Current Count : " + count );
  document.write("<br />");
}
document.write("Loop stopped!");
//-->
</script>

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

Output

5. 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>
<body>

<script type="text/javascript">
<!--
var aProperty;
document.write("Navigator Object Properties<br /> ");
for (aProperty in navigator)
{
  document.write(aProperty);
  document.write("<br />");
}
document.write ("Exiting from the loop!");
//-->
</script>
<p>Set the variable to different object and then try...</p>
</body>
</html>
```

Output

6. 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.

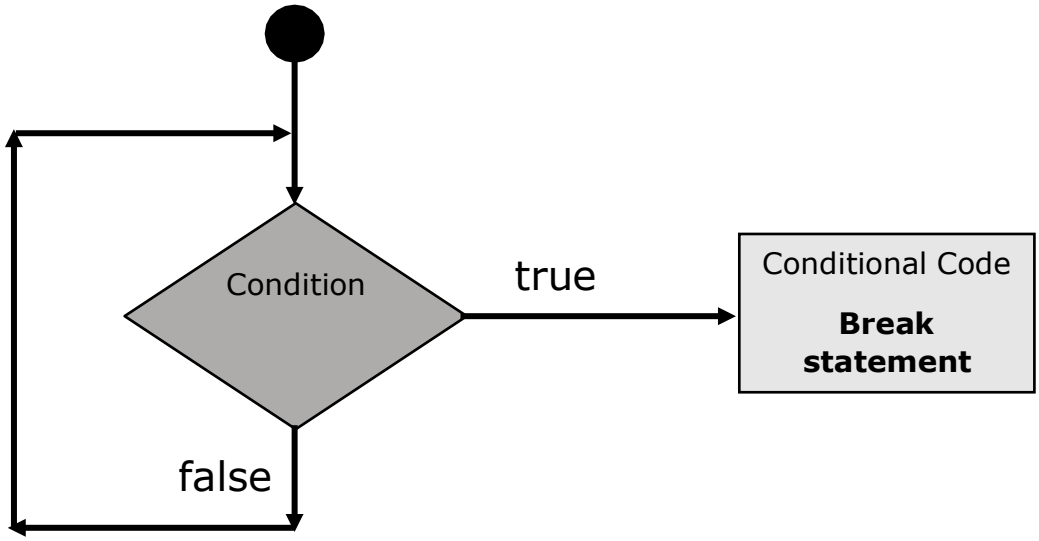
To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

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.

Flow Chart

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



Example

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

```
<html>
<body>

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

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

Output

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. Whena **continue** statement is encountered, the program flow moves to the loopcheck expression immediately and if the condition remains true, then it startsthe 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>

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

Output

7. 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(parameter-list)
{
  statements
}
//-->
</script>
```

Example

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

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

Calling a Function

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

```
<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

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

<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

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>
<script type="text/javascript">
function concatenate(first, last)
{
   var full;

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

```
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output