JavaScript 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 practical/programs. JavaScript allows us to write our own functions as well. A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it). We will see 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 in parenthesis (that might be empty), and a statement block surrounded by curly braces: {}. Function names can contain letters, digits, underscores, and dollar signs (same rules as variables). Function arguments (function parameters) are the values received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables. A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Syntax:
The basic syntax is shown here.

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

Example:
1) Function without parameters: A function called sayHello is defined that takes no parameters −

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

2) Function with arguments: Example of function that has one argument.

```
<html>
<body>
<script>
function getcube(number){
alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
</body>
</html>
```

Function Parameters:

Consider a simple syntax of a function:

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

Function parameters are the names listed in the function definition. In above example, parameter1,2 and 3 are Function parameters. Function arguments are the real values passed to (and received by) the function and it is specified when we call the function.

Example: sayHello() function given below 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>
```

JavaScript function definitions do not specify data types for parameters. JavaScript functions do not perform type checking on the passed arguments. JavaScript functions do not check the number of arguments received. If a function is called with missing arguments (less than declared), the missing values are set to: undefined. Sometimes this is acceptable, but sometimes it is better to assign a default value to the parameter. Consider following example:

```
<html>
<body>
<p id="demo"></p>
<script>
function myFunction(x, y) {
  if (y === undefined) {
    y = 0;
  }
  return x * y;
}
document.getElementById("demo").innerHTML = myFunction(4);
</script>
</body>
</html>
```

Function Invocation:

The code inside a function is not executed when the function is defined. The code inside the function will execute when "something" invokes (calls) the function. This "something" can be one of the following:

1) When an event occurs (when a user clicks a button)
2) When function is invoked (called) from JavaScript code
3) Automatically (self invoked)

Let us understand each option one by one.

1) When an event occurs (when a user clicks a button):

Consider following example. A function named sayHello() is called when you click the button.

```
<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>
```

2) When function is invoked (called) from JavaScript code:

Syntax:

```
function myFunction(a, b) {
  return a * b;
}
myFunction(10, 2);          // Will return 20
```

In above syntax, "myFunction(10,2)" statement calls the myFunction().

Example:

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>The global function (myFunction) returns the product of the arguments (a ,b):</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = myFunction(10, 2);
</script>
</body>
</html>
```

3) Automatically (self-invoked):
Function expressions can be made "self-invoking". A self-invoking expression is invoked (started) automatically, without being called. Function expressions will execute automatically if the expression is followed by (). You cannot self-invoke a function declaration. You have to add parentheses around the function to indicate that it is a function expression. Such function is actually an anonymous self-invoking function (function without name).
Syntax:

```
(function () {
  var x = "Hello!!";  // I will invoke myself
})();
```

Example:

```
<html>
<body>
<p>Functions can be invoked automatically without being called:</p>
<p id="demo"></p>
<script>
(function () {
  document.getElementById("demo").innerHTML = "Hello! I called myself";
})();
</script>
</body>
</html>
```

Function Call() (built-in function call()) :

There is a built-in function named **call()** in JavaScript. With the call() method, you can write a method that can be used on different objects. In JavaScript all functions are object methods. If a function is not a method of a JavaScript object, it is a function of the global object. The example below creates an object with 3 properties, firstName, lastName, fullName.
Syntax:

```
var person = {
  firstName:"John",
  lastName: "Doe",
  fullName: function () {
    return this.firstName + " " + this.lastName;
  }
}
person.fullName();   // Will return "John Doe"
```

Example:

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p id="demo"></p>
<script>
var myObject = {
  firstName:"John",
  lastName: "Doe",
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }}
```

```
x = myObject.fullName();
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Use of "this" Keyword: In a function definition, "this" refers to the "owner" of the function. In the example above, "this" is the "myObject" object that "owns" the "fullName" function. In other words, "this.firstName" means the "firstName" property of "this" object.

JavaScript call() Method: The call() method is a predefined JavaScript method. It can be used to invoke (call) a method with an owner object as an argument (parameter). With call(), an object can use a method belonging to another object. Following example calls the fullName method of person, using it on person1:

Example:

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p id="demo"></p>
<script>
var person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
var person1 = {
  firstName:"John",
  lastName: "Doe"
}
var x = person.fullName.call(person1);
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

The call() Method with Arguments: The call() method can accept arguments. Let us continue previous example with some additions.

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p id="demo"></p>
<script>
var person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + "," + city + "," + country;
  }
}
var person1 = {
  firstName:"John",
  lastName: "Doe" }
```

```
var x = person.fullName.call(person1, "Oslo", "Norway");
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Return Statement of Function:

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. When JavaScript reaches a return statement, the function will stop executing. Functions often compute a return value. The return value is "returned" back to the "caller".
Following function calculates the product of two numbers, and returns the result:

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p id="demo"></p>
<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b) {
  return a * b;
}
</script>
</body>
</html>
```

Function Closures:

A closure is a function having access to the parent scope, even after the parent function has closed. Let us understand it by one example: (Write following code in 'script' tag of html file.)

```
function foo()                  //1
{ var b = 1;                    //2
var temp=function innerFn()     //3
        {  b++;                 //4
        return b;               //5
        }                       //6
return temp;                    //7
}                               //8
var get_func_inner = foo();     //9
document.write(get_func_inner());   //10
```

Explanation of above code: At line number 9 we are done with the execution of function foo() and the entire body of function innerFn() is returned and stored in variable get_func_inner, due to the line 7 returns innerFn() with the help of variable "temp". The return statement does not execute the inner function – function is executed only when the variable name is followed by () , but rather the return statement returns the entire body of the function.
We can access the variable 'b' which is defined in function foo() through function innerFn() as the inner function preserves the scope chain of outer function at the time of execution of outer function i.e. the inner function knows the value of b through it's scope chain. This is closure in action, that is inner function can have access to the outer function variables as well as all the global variables.

Output of above code is: 2

We can use closure to count the number of times some specific function is called. We can also find out how many times some specific button of webpage is clicked using this concept.

Exercise:

1) Find factorial of a number with and without recursion.

2) Display N^{th} term of a Fibonacci series with and without recursion.

3) Find given number is perfect or not, use built-in method call(). A number is called perfect if it is equal to sum of its divisors.

4) Write a function which converts Celsius to Fahrenheit. Make this function self-invoking.

5) There are 2 buttons on a webpage: Like & Dislike. Count number of Likes and Dislikes.