JavaScript - Functions

A **function** in JavaScript is a group of reusable code that can be called anywhere in your program. It 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.

All statements you need to execute on the function call must be written inside the curly braces.

Syntax

The basic syntax to define the function in JavaScript is as follows −

```
function functionName(parameter-list) {
    statements
}
```

This type of function definition is called function declaration or function statement. We can also define a function using function expression. We will discuss function expression in details in the next chapter.

The following example defines a function called sayHello that takes no parameter −

```
function sayHello() {
    alert("Hello there");
}
```

Function Expression

The Function expression in JavaScript allows you to define a function as an expression. The function expression is similar to the anonymous function declaration. The function expression can be assigned to a varaible.

The syntax of function expression in JavaScript is as follows–

```
const varName = function (parameter-list) {
    statements
};
```

In the example below, we have defined a JavaScript function using function expression and assigned it to a vriable name myFunc.

```
const myFunc = function (x, y){
   return x + y;
};
```

Calling a Function

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

Example

The below code shows the button in the output. When you click the button, it will execute the sayHello() function. The sayHello() function prints the "Hello there!" message in the output.

Open Compiler

```
<html>
```

```
  <head>
     <script type="text/javascript">
        function sayHello() {
           alert("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 the write method and then try... </p>
  </body>
  </html>
```

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.

Open Compiler

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

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

The code below defines a function that concatenates two parameters before returning the resultant in the calling program. Also, you may take a look that how it returns the value using the return statement.

Open Compiler

```
<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');
         alert(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>
```

Functions as Variable Values

In JavaScript, functions can be used same as other variables. That’s why JavaScript is said to have a first-calss functions. The functions can be passed as arguments to the other functions.

Example

In the example below, we have declared a function using function expression and use the function to concatenate with other string.

Open Compiler

```
   <html>
   <body>
      <div id = "output"> </div>
      <script>
         const myFunc = function (){ return "Hello ";};
         document.getElementById("output").innerHTML = myFunc() + "Users.";
    </script>
   </body>
   </html>
```

We have covered two important concepts in separate chapters.

- JavaScript Nested Functions
- JavaScript Function Literals