JavaScript Functions

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).

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()
      {
```

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>
```

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>
```

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

JavaScript Function Parameters

A JavaScript function does not perform any checking on parameter values (arguments).

Function Parameters and Arguments

Earlier in this tutorial, you learned that functions can have **parameters**:

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

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** passed to (and received by) the function.

Parameter Rules

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.

Parameter Defaults

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:

Example

```
function myFunction(x, y) {
    if (y === undefined) {
          y = 0;
    }
}
```

a function is called with **too many arguments** (more than declared), these arguments can be reached using **the arguments object**.

The Arguments Object

JavaScript functions have a built-in object called the arguments object.

The argument object contains an array of the arguments used when the function was called (invoked).

This way you can simply use a function to find (for instance) the highest value in a list of numbers:

Example

```
x = findMax(1, 123, 500, 115, 44, 88);

function findMax() {
    var i;
    var max = -Infinity;
    for (i = 0; i < arguments.length; i++) {
        if (arguments[i] > max) {
            max = arguments[i];
        }
    }
    return max;
}
```

Or create a function to sum all input values:

Example

```
x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
    var i;
    var sum = 0;
    for (i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    return sum;
}
```

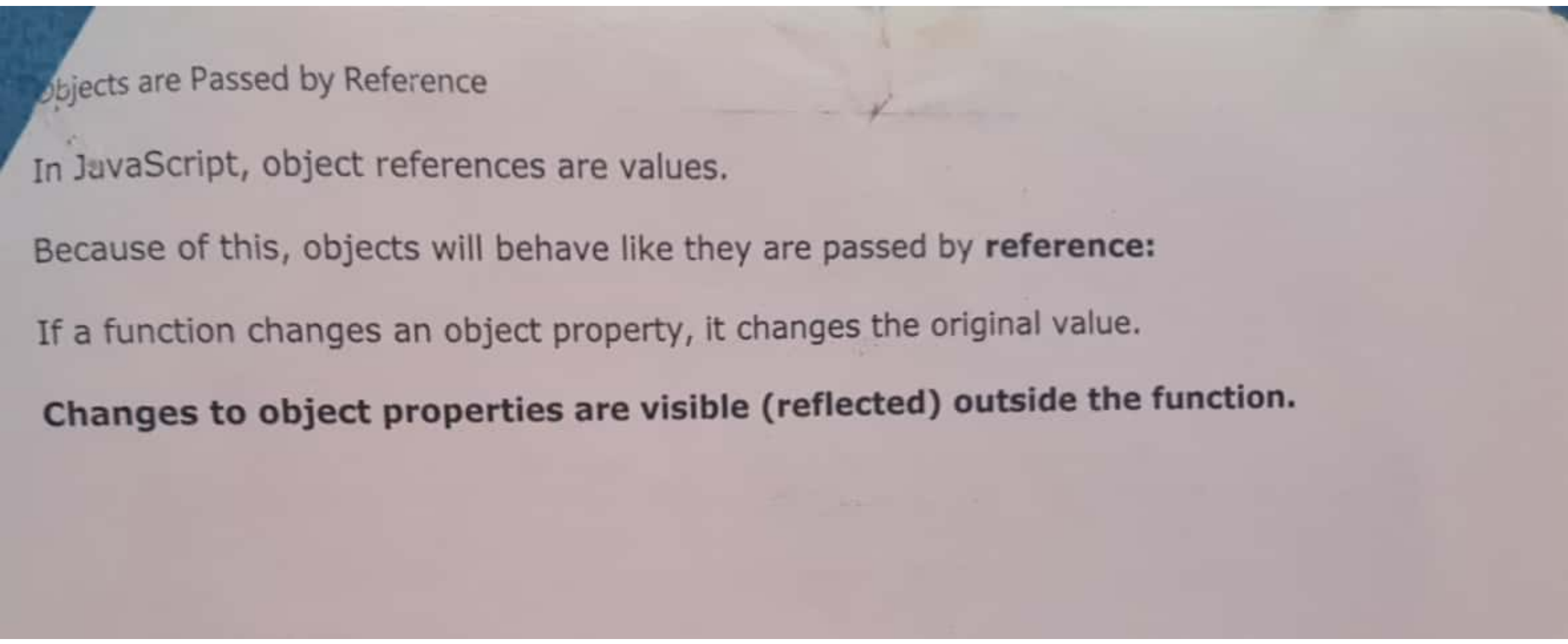
Arguments are Passed by Value

The parameters, in a function call, are the function's arguments.

JavaScript arguments are passed by **value**: The function only gets to know the values, not the argument's locations.

If a function changes an argument's value, it does not change the parameter's original value.

Changes to arguments are not visible (reflected) outside the function.



Objects are Passed by Reference

In JavaScript, object references are values.

Because of this, objects will behave like they are passed by **reference:**

If a function changes an object property, it changes the original value.

Changes to object properties are visible (reflected) outside the function.

JavaScript Functions and Events

A JavaScript function is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an event occurs, like when the user clicks a button

You will learn much more about functions and events in later chapters.

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
```

JavaScript Output

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

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

Using innerHTML

To access an HTML element, JavaScript can use the `document.getElementById(id)` method.

The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Changing the innerHTML property of an HTML element is a common way to display data in HTML

Using document.write()

For testing purposes, it is convenient to use `document.write()`:

Example

```
<!DOCTYPE html>
<html>
<body>
```

```
y First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Using document.write() after an HTML document is loaded, will delete all existing HTML:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:**Example**

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

console.log()

For debugging purposes, you can use the `console.log()` method to display data.

You will learn more about debugging in a later chapter.

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```