

## 2.1 Array

An array is very similar to a variable in that an array tells the browser to reserve a place in memory that can be used to store information.

In JavaScript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements

An array can comprise one or multiple elements. Each element is like a variable in that an array element refers to a memory location where information can be temporarily stored.

An array is identified by a unique name, similar to the name of a variable. A number called an index identifies an array element.

The combination of the array name and an index is nearly the same as a variable name. In your JavaScript, you use both the array name and the index to specify a particular memory location.

### 2.1.1 Declaration of an Array

You create an array by writing a declaration statement in your JavaScript, which is very similar to the way you declared a variable.

There are basically two ways to declare an array.

**Example:**

```
var House = [ ]; // method 1
var House = new array(); // method 2
```

This declaration statement has five parts:

the first part is the var syntax; the second part is the array name, which you create; the third part is the assignment operator; the fourth part is the new operator; and the fifth part is the Array() constructor. All these parts are shown here: var products = new Array()

### 2.1.2 Initializing an Array:

Initialization is the process of assigning a value when either a variable or an array is declared. The process to initialize an array is a little different than initializing a variable. You use the assignment operator to assign a value to a variable when declaring the variable.

**Example (for Method 1):**

```
// Initializing while declaring
```

```
var house = ["1BHK", "2BHK", "3BHK", "4BHK"];
```

**Example (for Method 2):**

Initializing while declaring

Creates an array having elements 10, 20, 30, 40, 50

```
- var house = new Array(10, 20, 30, 40, 50);
```

Creates an array of 5 undefined elements

```
-var house1 = new Array(5);
```

Creates an array with element 1BHK

```
- var home = new Array("1BHK");
```

As shown in above example the house contains 5 elements i.e. (10 , 20,30,40,50) while house1 contains 5 undefined elements instead of having a single element 5.

Hence, while working with numbers this method is generally not preferred but it works fine with Strings and Boolean as shown in the example above home contains a single element 1BHK.

We can also update after initialization.

```
// Creates an array of 4 undefined elements
```

```
var house1 = new Array(4);
```

```
// Now assign values
```

```
house1[0] = "1BHK"
```

```
house1[1] = "2BHK"
```

```
house1[2] = "3BHK"
```

```
house1[3] = "4BHK"
```

**An array in JavaScript can hold different elements**

**We can store Numbers, Strings and Boolean in a single array.**

**Example:**

```
// Storing number, boolean, strings in an Array
```

```
var house = ["1BHK", 25000, "2BHK", 50000, "Rent", true];
```

### 2.1.3Defining Array Elements

Think of an array as a list containing the same kinds of things—such as a list of product names or a list of customer first names. Each item on the list is identified by the number in which the item appears on the list.

The first item is number 0, the second item is number 1, then 2, and so on. You are probably wondering why the second item on the list is numbered 1 instead of 2. The reason is because the first digit in the decimal numbering system is 0 and not 1.

To assign a product name to an array element, you must specify the name of the array followed by the index of the array element. The index must be enclosed within square brackets. First, let's declare an array called products: `var house = new Array()`

Next, let's specify the first element of that array. In this example, house is the name of the array, and 0 is the index of the first element of the array. (The second element would look just like the first element, except the index is 1, not 0.)

`house[0]` You treat an array element like you treat a variable name in your JavaScript:

- You use the assignment operator (=) to assign a value to an array element: `house[0] = '1BHK'`
- You use the array element (array name plus the index) to tell the browser that you want to use the value that is associated with the array element.

This is the same as using the variable name to tell the browser that you want to use the value that is associated with the variable: `document.write(house[0])`

// Creates an array of 4 undefined elements

```
var house1 = new Array(4);

// Now assign values
house1[0] = "1BHK"
house1[1] = "2BHK"
house1[2] = "3BHK"
house1[3] = "4BHK"
```

### 2.1.4Looping the Array

The power of using an array is evident when you need to process each element of the array. You can use a for loop to access each array element. Let's see how this is done. Suppose you need to display all the array elements on a document.

you remember that you place the information you want displayed between the parentheses of the `document.write()` method.

If you use four variables—one for each product—you'll have to write the document.write() method in four different statements within your JavaScript. But if you use an array, you'll have to write the document.write() method in only one statement. Here's how this is done:

```
for (var i = 0; i < products.length; i++)  
{  
    document.write(products[i])  
}
```

**how the browser displays elements of the products array on the screen**

```
<!DOCTYPE html >  
  
<html >  
  
    <head>  
  
        <title>Display Array Elements</title>  
  
    </head>  
  
    <body>  
  
        <script language="Javascript" type="text/javascript">  
  
            var products = new Array()  
  
            products[0] = 'Soda '  
  
            products[1] = 'Water'  
  
            products[2] = 'Pizza'  
  
            products[3] = 'Icecreame'  
  
            for (var i = 0; i < products.length; i++)  
            {  
  
                document.write(products[i] + '<br>')  
  
            }  
  
        </script>
```

```
</body>
```

```
</html>
```

## 2.1.5 Adding an Array Element

### push():

The push() method adds new items to the end of an array, and returns the new length.

The new item(s) will be added at the end of the array.

Note: This method changes the length of the array.

### Syntax

```
array.push(item1, item2, ..., itemX)
```

### Parameter Values

Parameter	Description
<i>item1</i> , <i>item2</i> , ..., <i>itemX</i>	Required. The item(s) to add to the array

### Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click the button to add a new element to the array.</p>
```

```
<button onclick="myFunction()">Try it</button>
```

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

```
<script>
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
document.getElementById("demo").innerHTML = fruits;
```

```
function myFunction()
{
    fruits.push("Kiwi");
    document.getElementById("demo").innerHTML = fruits;
}
</script>
</body>
</html>
```

### **unshift():**

The unshift() method adds new items to the beginning of an array, and returns the new length.

Note: This method changes the length of an array.

### **Syntax**

array.unshift(*item1*, *item2*, ..., *itemX*)

### **Parameter Values**

Parameter	Description
<i>item1</i> , <i>item2</i> , ..., <i>itemX</i>	Required. The item(s) to add to the beginning of the array

### **Example:**

```
<!DOCTYPE html>
<html>
<body>
    <p>Click the button to add elements to the beginning of the array.</p>
    <button onclick="myFunction()">Try it</button>
```

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

```
<script>
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
document.getElementById("demo").innerHTML = fruits;
```

```
function myFunction()
```

```
{
```

```
fruits.unshift("Lemon", "Pineapple");
```

```
document.getElementById("demo").innerHTML = fruits;
```

```
}
```

```
</script>
```

<p><b>Note:</b> The unshift() method does not work properly in Internet Explorer 8 and earlier: the values will be inserted, but the return value will be <em>undefined</em>.</p>

```
</body>
```

```
</html>
```

### **pop():**

The pop() method removes the last element of an array, and returns that element.

Note: This method changes the length of an array.

```
array.pop()
```

### **Parameters**

None
------

### **Example:**

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click the button to remove the last element from the array.</p>

<button onclick="myFunction()">Try it</button>

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

<script>

    var fruits = ["Banana", "Orange", "Apple", "Mango"];

    document.getElementById("demo").innerHTML = fruits;

function myFunction()

{

    fruits.pop();

    document.getElementById("demo").innerHTML = fruits;

}

</script>

</body>

</html>
```

### **shift() :**

The shift() method removes the first item of an array.

Note: This method changes the length of the array.

Note: The return value of the shift method is the removed item.

array.shift()

### **Parameters**

None
------

### **Example:**

```
<!DOCTYPE html>
```



```
<html>

<body>

    <p>Click the button to remove the first element of the array.</p>

    <button onclick="myFunction()">Try it</button>

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

    <script>

        var fruits = ["Banana", "Orange", "Apple", "Mango"];

        document.getElementById("demo").innerHTML = fruits;

        function myFunction()

        {

            fruits.shift();

            document.getElementById("demo").innerHTML = fruits;

        }

    </script>

</body>

</html>
```

### 2.1.6 Sorting Array Elements

The sort() method sorts the items of an array.

The sort order can be either alphabetic or numeric, and either ascending (up) or descending (down).

By default, the sort() method sorts the values as strings in alphabetical and ascending order.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce an incorrect result when sorting numbers.

You can fix this by providing a "compare function" (See "Parameter Values" below).

Note: This method changes the original array.

**Syntax:**

```
array.sort(compareFunction)
```

**Parameter Values:**

Parameter	Description
<i>compareFunction</i>	<p>Optional. A function that defines an alternative sort order. The function should return a negative, zero, or positive value, depending on the arguments, like:</p> <pre>function(a, b){return a-b}</pre> <p>When the sort() method compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.</p> <p><b>Example:</b></p> <p>When comparing 40 and 100, the sort() method calls the compare function(40,100). The function calculates 40-100, and returns -60 (a negative value). The sort function will sort 40 as a value lower than 100.</p>

**Example: sort the array.**

```
<!DOCTYPE html>

<html>

<body>

    <p>Click the button to sort the array.</p>

    <button onclick="myFunction()">Try it</button>

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

    <script>

        var fruits = ["Banana", "Orange", "Apple", "Mango"];

        document.getElementById("demo").innerHTML = fruits;

        function myFunction()

        {

            fruits.sort();

            document.getElementById("demo").innerHTML = fruits;

        }

    </script>

</body>

</html>
```

**Example: to sort the array in ascending order**

```
<!DOCTYPE html>

<html>

<body>

    <p>Click the button to sort the array in ascending order.</p>

    <button onclick="myFunction()">Try it</button>

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

    <script>

        var points = [40, 100, 1, 5, 25, 10];

        document.getElementById("demo").innerHTML = points;

        function myFunction()

        {

            points.sort(function(a, b){return a-b});

            document.getElementById("demo").innerHTML = points;

        }

    </script>

</body>

</html>
```

**Example: to sort the array in descending order**

```
<!DOCTYPE html>
<html>
<body>
    <p>Click the button to sort the array in descending order.</p>
    <button onclick="myFunction()">Try it</button>
    p id="demo"></p>
    <script>
        var points = [40, 100, 1, 5, 25, 10];
        document.getElementById("demo").innerHTML = points;

        function myFunction()
        {
            points.sort(function(a, b)
            {
                return b-a
            });
            document.getElementById("demo").innerHTML = points;
        }
    </script>
</body>
</html>
```

**Example: to get the highest number in the array**

```
<!DOCTYPE html>
<html>
<body>
  <p>Click the button to get the highest number in the array.</p>
  <button onclick="myFunction()">Try it</button>
  <p id="demo"></p>
  <script>
    var points = [40, 100, 1, 5, 25, 10];
    document.getElementById("demo").innerHTML = points;
    function myFunction()
    {
      points.sort(function(a, b){return b-a});
      document.getElementById("demo").innerHTML = points[0];
    }
  </script>
</body>

</html>
```

**Example: to get the lowest number in the array**

```
<!DOCTYPE html>
<html>
<body>
  <p>Click the button to get the lowest number in the array.</p>
  <button onclick="myFunction()">Try it</button>
  <p id="demo"></p>
  <script>
    var points = [40, 100, 1, 5, 25, 10];
    document.getElementById("demo").innerHTML = points;
    function myFunction()
    {
      points.sort(function(a, b){return a-b});
      document.getElementById("demo").innerHTML = points[0];
    }
  </script>
</body>
</html>
```

**Example: sort the array in descending order**

```
<!DOCTYPE html>

<html>

<body>

    <p>Click the button to sort the array in descending order.</p>

    <button onclick="myFunction()">Try it</button>

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

    <script>

        var fruits = ["Banana", "Orange", "Apple", "Mango"];

        document.getElementById("demo").innerHTML = fruits;

        function myFunction()

        {

            fruits.sort();

            fruits.reverse();

            document.getElementById("demo").innerHTML = fruits;

        }

    </script>

</body>

</html>
```

### **2.1.7 Making a New Array from an Existing Array**

The slice() method returns the selected elements in an array, as a new array object.

The slice() method selects the elements starting at the given start argument, and ends at, but does not include, the given end argument.

Note: The original array will not be changed.



**Syntax**array.slice(*start*, *end*)**Parameter Values**

Parameter	Description
<i>start</i>	Optional. An integer that specifies where to start the selection (The first element has an index of 0). Use negative numbers to select from the end of an array. If omitted, it acts like "0"
<i>end</i>	Optional. An integer that specifies where to end the selection. If omitted, all elements from the start position and to the end of the array will be selected. Use negative numbers to select from the end of an array

**Example: to extract the second and the third elements from the array**

```

<!DOCTYPE html>

<html>

<body>

<p>Click the button to extract the second and the third elements from the array.</p>

<button onclick="myFunction()">Try it</button>

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

<script>

    function myFunction()

```

```
{  
    var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
    var citrus = fruits.slice(1, 3);  
    document.getElementById("demo").innerHTML = citrus;  
}  
</script>  
</body>  
</html>
```

**Example: to extract the third and fourth elements, using negative numbers**

```
<!DOCTYPE html>  
<html>  
<body>  
    <p>Click the button to extract the third and fourth elements, using negative  
numbers.</p>  
    <button onclick="myFunction()">Try it</button>  
    <p id="demo"></p>  
    <script>  
        function myFunction()  
        {  
            var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
            var myBest = fruits.slice(-3, -1);  
            document.getElementById("demo").innerHTML = myBest;  
        }  
    </script>  
</body>
```

</html>

## 2.1.8Combining Array Elements into a String

### Concat():-

The concat() method is used to join two or more arrays.

This method does not change the existing arrays, but returns a new array, containing the values of the joined arrays.

### Syntax

array1.concat(array2, array3, ..., arrayX)

### Parameter Values

Parameter	Description
array2, array3, ..., arrayX	Required. The arrays to be joined

### Example: to join three arrays

<!DOCTYPE html>

<html>

<body>

<p>Click the button to join three arrays.</p>

<button onclick="myFunction()">Try it</button>

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

<script>

function myFunction()

```
    {  
  
        var hege = ["Cecilie", "Lone"];  
  
        var stale = ["Emil", "Tobias", "Linus"];  
  
        var kai = ["Robin"];  
  
        var children = hege.concat(stale,kai);  
  
        document.getElementById("demo").innerHTML = children;  
  
    }  
  
</script>  
  
</body>  
  
</html>
```

### **Join():-**

The join() method returns the array as a string.

The elements will be separated by a specified separator. The default separator is comma (,).

Note: this method will not change the original array.

### **Syntax:**

`array.join(separator)`

### **Parameter Values:**

Parameter	Description
<i>separator</i>	Optional. The separator to be used. If omitted, the elements are separated with a comma

**Example: to join the array elements into a string**

```
<!DOCTYPE html>

<html>

<body>

    <p>Click the button to join the array elements into a string.</p>

    <button onclick="myFunction()">Try it</button>

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

    <script>

        function myFunction()

        {

            var fruits = ["Banana", "Orange", "Apple", "Mango"];

            var x = document.getElementById("demo");

            x.innerHTML = fruits.join(" and ");

        }

    </script>

</body>

</html>
```

### **2.1.9 Changing Elements of the Array**

We can change elements of array by using above explained methods like push(), shift(), pop() etc.

## 2.2 Function

### What Is a Function?

A javascript function is block of code designed to perform particular task.

A javascript function is executed when something invokes it.

A function can be called from anywhere in the JavaScript file or in the HTML document.

### 2.2.1 Defining a function

The process of creating a function is called defining a function. The process of using the function is referred to as calling a function.

Defining a Function -A function must be defined before it can be called in a JavaScript statement.

A function definition consists of four parts: the name, parentheses, a code block, and an optional return keyword.

#### Function Name

The function name is the name that you've assigned the function.

It is placed at the top of the function definition and to the left of the parentheses.

Any name will do, as long as it follows certain naming rules.

#### The name must be

- Letter(s), digit(s), or underscore character
  - Unique to JavaScripts on your web page, as no two functions can have the same name
- The name cannot
- Begin with a digit
  - Be a keyword
  - Be a reserved word

The name should be

- Representative of the action taken by statements within the function

#### Parentheses

Parentheses are placed to the right of the function name at the top of the function definition. Parentheses are used to pass values to the function; these values are called arguments.

### **Code Block**

The code block is the part of the function definition where you insert JavaScript statements that are executed when the function is called by another part of your JavaScript application

**Return (Optional)** The return keyword tells the browser to return a value from the function definition to the statement that called the function

## **2.2.2 Writing a Function Definition**

Following is a simple function definition. It is called IncreaseSalary() and tells the browser the steps that are necessary to give you a raise in pay (at least on paper). This function contains all the values needed to calculate your new salary; therefore, no argument is needed:

```
function IncreaseSalary()
{
    var salary = 500000 * 1.25
    alert("Your new salary is " + salary)
}
```

## **2.2.3 Adding Arguments**

An argument is one or more variables that are declared within the parentheses of a function definition. This is illustrated in the following code sample.

OldSalary is an argument of the IncreaseSalary() function.

```
function IncreaseSalary(OldSalary)
{
    var NewSalary = OldSalary * 1.25
    alert("Your new salary is " + NewSalary)
}
```

The JavaScript statement or the HTML code provides the value when it calls the function. This is called passing a value to the function.

### **Adding Multiple Arguments**

You can use as many arguments as necessary for the function to carry out its task.

Each argument must have a unique name, and each argument within the parentheses must be separated by a comma.

Let's revise the preceding example and make the percentage of salary increase an argument.

Here, two arguments are used: OldSalary and PercIncrease.

```
function IncreaseSalary(OldSalary, PercIncrease)
{
    var NewSalary = OldSalary * (1 + (PercIncrease / 100))
    alert("Your new salary is " + NewSalary)
}
```

## 2.2.4 The Scope of Variables and Arguments

A variable's scope is the area of the script in which that variable can be used. There are two types of variables.

**. Local variables** have a single function as their scope. They can be used only within the function they are created in.

A variable can be declared within a function, such as the NewSalary variable in the IncreaseSalary() function.

This is called a local variable, because the variable is local to the function. Other parts of your JavaScript don't know that the local variable exists because it's not available outside the function.

**.Global variables** have the entire script (and other scripts in the same HTML document) as their scope. They can be used anywhere, even within functions.

A variable can be declared outside a function, Such a variable is called a global variable because it is available to all parts of your JavaScript—that is, statements within any function and statements outside the function can use a global variable

JavaScript developers use the term scope to describe whether a statement of a JavaScript can use a variable. A variable is considered **in scope** if the statement can access the variable. A variable is **out of scope** if a statement cannot access the variable.



## 2.3 Calling a Function

A function is called by using the function name followed by parentheses.

If the function has arguments, values for each argument are placed within the parentheses.

You must place these values in the same order that the arguments are listed in the function definition. A comma must separate each value.

Here's how the IncreaseSalary() function is called: IncreaseSalary(500000, 6) Notice that the first value (500000) is the old salary and the second value (6) is the percentage of the salary increase.

### 2.3.1 Calling a Function Without an Argument

Here is an example of how to define and call a function that does not have any arguments.

The function definition is placed within the <head> tag and the function call is placed within the <body> tag.

When the function is called, the browser goes to the function definition and executes statements within the code block of the function.

The first statement declares the salary variable and initializes it with the increased salary that is produced by the calculation. The value of the salary is then displayed in an alert dialog box

```
<!DOCTYPE html >

<html>

<head>

    <title>Functions</title>

    <script language="Javascript" type="text/javascript">

        function IncreaseSalary()

        {

            var salary = 500000 * 1.25

            alert('Your new salary is ' + salary)

        }

    </script>

</head>
```

```
<body>

    <script language="Javascript" type="text/javascript">

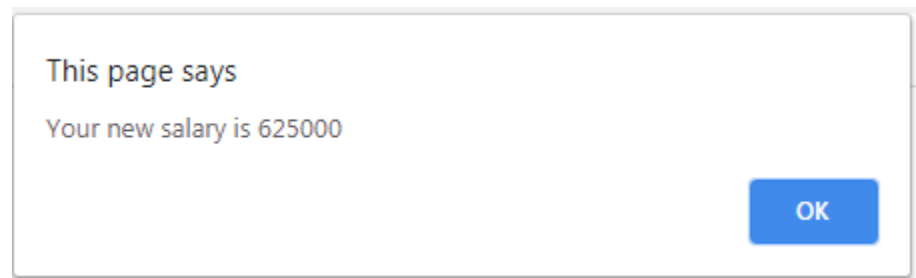
        IncreaseSalary()

    </script>

</body>

</html>
```

OUTPUT:



## Calling a Function with an Argument

Let's revise the previous example and modify the IncreaseSalary() function to accept the old salary and the percentage increase as arguments.

Before calling this function, we prompt the user to enter the old salary and the percentage increase. The values entered are used to initialize two variables: Salary and Increase. Both of these are global variables, because they are defined outside of a function.

The Salary and Increase variables are then used within the parentheses of the function call, which tells the browser to assign these values to the corresponding arguments in the function definition.

The function calculates and displays the new salary.

```
<!DOCTYPE html >

<html >

    <head>

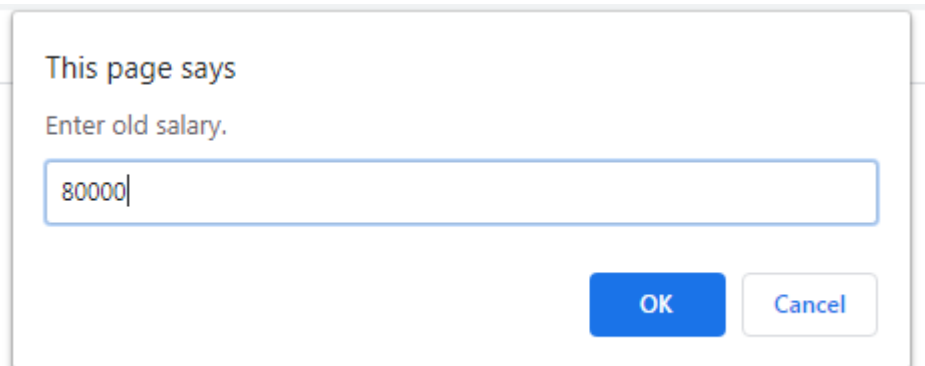
        <title>Functions</title>

        <script language="Javascript" type="text/javascript">

            function IncreaseSalary(OldSalary, PercIncrease)
```

```
        {  
            var NewSalary = OldSalary * (1 + (PercIncrease / 100))  
            alert("Your new salary is " + NewSalary)  
        }  
    </script>  
</head>  
<body>  
    <script language="Javascript" type="text/javascript">  
        var Salary = prompt('Enter old salary.', ' ')  
        var Increase = prompt('Enter salary increase as percent.', ' ')  
        IncreaseSalary(Salary, Increase)  
    </script>  
</body>  
</html>
```

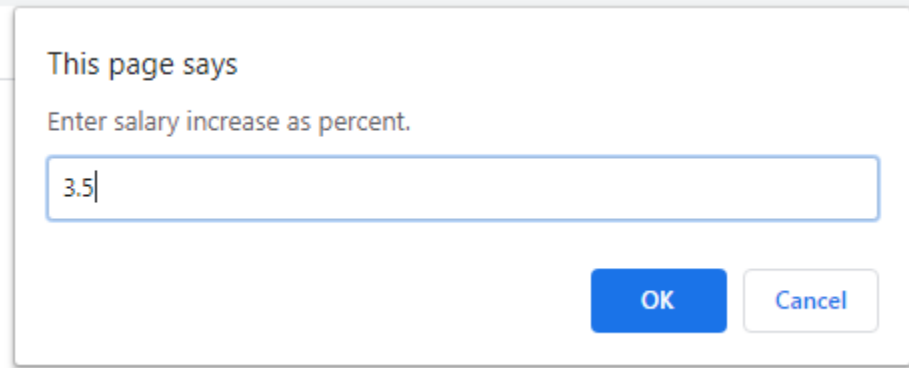
OUTPUT:



This page says

Enter old salary.

OK Cancel

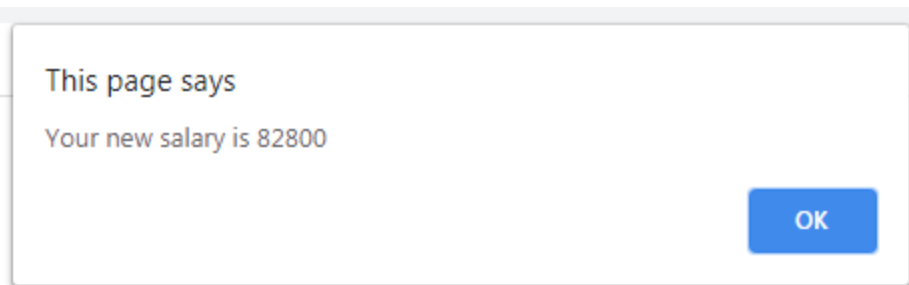


This page says

Enter salary increase as percent.

3.5

OK Cancel



This page says

Your new salary is 82800

OK

### 2.3.2 Calling a Function from HTML

A function can be called from HTML code on your web page. Typically, a function will be called in response to an event, such as when the web page is loaded or unloaded by the browser.

You call the function from HTML code nearly the same way as the function is called from within a JavaScript, except in HTML code you assign the function call as a value of an HTML tag attribute. Let's say that you want to call a function when the browser loads the web page. Here's what you'd write in the tag of the web page:

Here's what you'd write to call the function right before the user moves on to another web page:

The next example shows how to call these functions in a web page.

- i. We define each function in a JavaScript placed in the tag.
- ii. We assign a call to the WelcomeMessage() function to the onload attribute of the tag. This displays the welcome message when the browser loads the web page.
- iii. The call to the GoodbyeMessage() function is assigned to the onunload attribute of the tag. This displays the goodbye message when the browser unloads the web page to make room for a new web page.

```
<!DOCTYPE html >

<html >

    <head>

        <title>Calling a function from HTML</title>

        <script language="Javascript" type="text/javascript">

            function WelcomeMessage()

            {

                alert('Glad to see you.')

            }

            function GoodbyeMessage()

            {

                alert('So long.')

            }

        </script>

    </head>

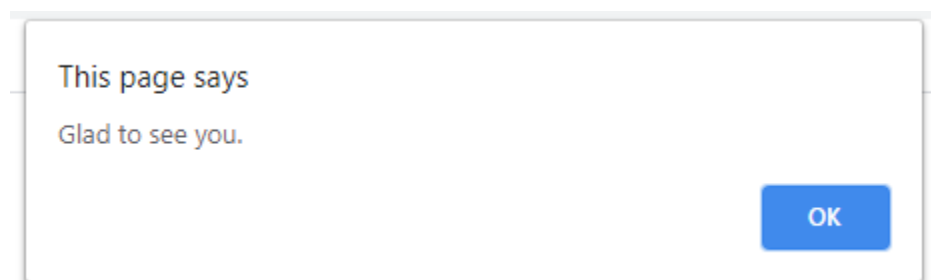
    <body onload="WelcomeMessage()"

        onunload="GoodbyeMessage()">

    </body>

</html>
```

OUTPUT:



### 2.3.3 Function Calling Another Function

JavaScript developers typically divide an application into many functions, each of which handles a portion of the application.

Functions, can be called from any JavaScript or from HTML code on a web page itself. This means that a function can also be called from another function.

We defined a logon function that handles all the tasks that are necessary for a user to log on to your application. This includes displaying dialog boxes prompting the user to enter a user ID and password.

We defined another function whose only tasks are to validate a user ID and password and report back whether or not the logon information is valid.

The logon function passes the user ID and password to the validation function and then waits for the validation function to signal whether or not they are valid.

The logon function then proceeds by telling the user whether the logon is valid or not valid.

### Returning Values from a Function

A function can be designed to do something and then report back to the statement that calls after it's finished—such as the validation function in the previous section, which validates a user ID and password and then reports back whether they are valid or not.

A function reports back to the statement that calls the function by returning a value to that statement using the return keyword, followed by a return value in a statement.

Here's what this looks like:

```
function name ()  
{  
    return value  
}
```

In this code segment, the return statement returns a Boolean value true.

You can return any value or variable in a return statement. The return value is typically assigned to a variable by the statement that called the function and then used by other statements in the JavaScript.

This is illustrated in the following code segment:

```
valid = ValidateLogon('ScubaBob', 'diving')
```

This statement calls the ValidateLogon() and passes it a user ID (the first argument) and password (the second argument).

The return value, which in this example is either true or false, is then assigned to the valid variable.

```
<!DOCTYPE html >

<html >

    <head>

        <title>Returning a value from a function</title>

        <script language="Javascript" type="text/javascript">

            function Logon()

            {

                var userID

                var password

                var valid

                userID = prompt('Enter user ID',' ')

                password = prompt('Enter password',' ')

                valid = ValidateLogon(userID, password)

                if ( valid == true)

                {

                    alert('Valid Logon')

                }

                else

                {

                    alert('Invalid Logon')

                }

            }

        }

    }

</head>

</html>
```

```
        }  
        function ValidateLogon(id,pwd)  
        {  
            var ReturnValue  
            if (id == 'ScubaBob' && pwd == 'diving')  
            {  
                ReturnValue = true  
            }  
            else  
            {  
                ReturnValue = false  
            }  
            return ReturnValue  
        }  
    </script>  
</head>  
<body>  
    <script language="Javascript" type="text/javascript">  
        Logon()  
    </script>  
</body>  
</html>
```



This page says

Enter user ID

This page says

Enter password

This page says

Invalid Logon

## 2.4 JavaScript Strings:

Strings are useful for holding data that can be represented in text form.

A JavaScript string stores a series of characters like "John Doe".

A string can be any text inside double or single quotes:

```
var carName1 = "Volvo XC60";  
var carName2 = 'Volvo XC60';
```

String indexes are zero-based: The first character is in position 0, the second in 1, and so on.

String methods help you to work with strings.

### 2.4.1 String manipulation :

String manipulation (or string handling) is the process of changing, parsing, splicing, pasting, or analyzing strings. String manipulation typically comes as a mechanism or a library feature of in most programming languages.

Reasons for manipulating strings is to creating data elements from text.

Typically, most programming languages provide a string data type that holds a sequence of characters.

#### **Common operations:**

Concatenation is the process of joining two strings together into a single string. For example "race" concatenated with "car" results in "racecar".

Splitting is the process of breaking down a string into multiple strings according to a certain delimiter or rule (e.g. regex pattern). For example "A B C" could be split into three separate strings, ("A", "B", "C"), using the space character as a delimiter.

Substrings is the process of extracting a portion of the string from a bigger string. Such operations typically involve a starting offset and a length. For example, one possible substring of "apples" is "pp".

Case conversion is the process of converting a string into a specific case for example into all lowercase or titlecase.

Searching is the process of searching for a specific pattern in a string.

### 2.4.2 Joining a string

#### **The concat() Method**

concat() joins two or more strings:

**Example**

```
<!DOCTYPE html>

<html>

<body>

    <h2>JavaScript String Methods</h2>

    <p>The concat() method joins two or more strings:</p>

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

    <script>

        var text1 = "Hello";

        var text2 = "World!";

        var text3 = text1.concat(" ",text2);

        document.getElementById("demo").innerHTML = text3;

    </script>

</body>

</html>
```

The concat() method can be used instead of the plus operator. These two lines do the same:

**Example**

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

All string methods return a new string. They don't modify the original string. Formally said: Strings are immutable: Strings cannot be changed, only replaced.

## 2.4.3 Retrieving a character from given position:

### The charAt() Method

The charAt() method returns the character at a specified index (position) in a string:

**Example**

```
var str = "HELLO WORLD";  
str.charAt(0);      // returns H
```

#### **2.4.4.Retrieving position of character of a given string:**

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

##### **Example**

```
<!DOCTYPE html>  
  
<html>  
  
<body>  
  
    <h2>JavaScript String Methods</h2>  
  
    <p>The indexOf() method returns the position of the first occurrence of a specified  
text:</p>  
  
    <p id="demo"></p>  
  
<script>  
  
    var str = "Please locate where 'locate' occurs!";  
  
    var pos = str.indexOf("locate");  
  
    document.getElementById("demo").innerHTML = pos;  
  
</script>  
  
</body>  
  
</html>
```

#### **String Length**

The length property returns the length of a string:

### Example

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript String Properties</h2>

<p>The length property returns the length of a string:</p>

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

<script>

    var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    var sln = txt.length;

    document.getElementById("demo").innerHTML = sln;

</script>

</body>

</html>
```

## 2.4.5 Dividing Text:

### Converting a String to an Array

A string can be converted to an array with the `split()` method:

### Example

```
var txt = "a,b,c,d,e"; // String
txt.split(",");        // Split on commas
txt.split(" ");        // Split on spaces
txt.split("|");        // Split on pipe
```

If the separator is omitted, the returned array will contain the whole string in index [0].

If the separator is `"`, the returned array will be an array of single characters:

### Example

```
var txt = "Hello"; // String
txt.split("");     // Split in characters
```

## The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

### Example

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new string:</p>

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

<script>
```

```
var str = "Apple, Banana, Kiwi";

var res = str.slice(7,13);
```

```
document.getElementById("demo").innerHTML = res;

</script>

</body>

</html>
```

The result of res will be:

Banana

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

**Example**

```
var str = "Apple, Banana, Kiwi";  
var res = str.slice(-12, -6);
```

The result of res will be:

Banana

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

**Example**

```
var str = "Apple, Banana, Kiwi";  
var res = str.slice(-12, -6);
```

The result of res will be:

Banana

If you omit the second parameter, the method will slice out the rest of the string:

**Example**

```
var res = str.slice(7);
```

The result of res will be:

Banana, Kiwi

or, counting from the end:

**Example**

```
var res = str.slice(-12);
```

The result of res will be:

Banana, Kiwi

Negative positions do not work in Internet Explorer 8 and earlier.

## 2.4.6 Copying a substring:

### The substring() Method

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

#### Example

```
<!DOCTYPE html>

<html>

<body>

    <h2>JavaScript String Methods</h2>

    <p>The substring() method extract a part of a string and returns the extracted parts in a
new string:</p>

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

<script>

    var str = "Apple, Banana, Kiwi";

    var res = str.substring(7,13);

    document.getElementById("demo").innerHTML = res;

</script>

</body>

</html>
```

The result of *res* will be:

Banana

If you omit the second parameter, substring() will slice out the rest of the string.

### The substr() Method

substr() is similar to slice().

The difference is that the second parameter specifies the length of the extracted part.

#### Example

```
<!DOCTYPE html>
```



```
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>The substr() method extract a part of a string
and returns the extracted parts in a new string:</p>

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

<script>

    var str = "Apple, Banana, Kiwi";

    var res = str.substr(7,6);

    document.getElementById("demo").innerHTML = res;

</script>

</body>

</html>
```

The result of res will be:

Banana

If you omit the second parameter, substr() will slice out the rest of the string.

### Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7);
```

The result of res will be:

Banana, Kiwi

If the first parameter is negative, the position counts from the end of the string.

### Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(-4);
```

The result of res will be:

Kiwi

## 2.4.7 Converting string to number & number to string:

### Convert String to number:

#### parseInt():-

#### Definition and Usage

The parseInt() function parses a string and returns an integer.

The radix parameter is used to specify which numeral system to be used, for example, a radix of 16 (hexadecimal) indicates that the number in the string should be parsed from a hexadecimal number to a decimal number.

If the radix parameter is omitted, JavaScript assumes the following:

If the string begins with "0x", the radix is 16 (hexadecimal)

If the string begins with "0", the radix is 8 (octal). This feature is deprecated

If the string begins with any other value, the radix is 10 (decimal)

Note: Only the first number in the string is returned!

Note: Leading and trailing spaces are allowed.

Note: If the first character cannot be converted to a number, parseInt() returns NaN.

**Note:** Older browsers will result parseInt("010") as 8, because older versions of ECMAScript, (older than ECMAScript 5, uses the octal radix (8) as default when the string begins with "0". As of ECMAScript 5, the default is the decimal radix (10).

#### Syntax

parseInt(*string*, *radix*)

#### Parameter Values

Parameter	Description
<i>string</i>	Required. The string to be parsed
<i>radix</i>	Optional. A number (from 2 to 36) that represents the numeral system

to be used

**Example:**

```

<!DOCTYPE html>

<html>

<body>

    <p>Click the button to parse different strings.</p>

    <button onclick="myFunction()">Try it</button>

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

    <script>

        function myFunction()

        {

            var a = parseInt("10") + "<br>";

            var b = parseInt("10.00") + "<br>";

            var c = parseInt("10.33") + "<br>";

            var d = parseInt("34 45 66") + "<br>";

            var e = parseInt(" 60 ") + "<br>";

            var f = parseInt("40 years") + "<br>";

            var g = parseInt("He was 40") + "<br>";

            var h = parseInt("10", 10)+ "<br>";

            var i = parseInt("010")+ "<br>";

            var j = parseInt("10", 8)+ "<br>";

            var k = parseInt("0x10")+ "<br>";

            var l = parseInt("10", 16)+ "<br>";

            var n = a + b + c + d + e + f + g + "<br>" + h + i + j + k + l;

            document.getElementById("demo").innerHTML = n;

```

```
    }  
</script>  
</body>  
</html>
```

### **parseFloat():-**

#### **Definition and Usage**

The parseFloat() function parses a string and returns a floating point number.

This function determines if the first character in the specified string is a number. If it is, it parses the string until it reaches the end of the number, and returns the number as a number, not as a string.

Note: Only the first number in the string is returned!

Note: Leading and trailing spaces are allowed.

Note: If the first character cannot be converted to a number, parseFloat() returns NaN.

#### **Syntax**

parseFloat(*string*)

#### **Parameter Values**

Parameter	Description
<i>string</i>	Required. The string to be parsed

#### **Example:**

```
<!DOCTYPE html>  
<html>  
<body>  
  
    <p>Click the button to parse different strings.</p>
```

```
<button onclick="myFunction()">Try it</button>

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

<script>

    function myFunction()
    {

        var a = parseFloat("10")

        var b = parseFloat("10.00")

        var c = parseFloat("10.33")

        var d = parseFloat("34 45 66")

        var e = parseFloat(" 60 ")

        var f = parseFloat("40 years")

        var g = parseFloat("He was 40")

        document.getElementById("demo").innerHTML =

        a + "<br>" +

        b + "<br>" +

        c + "<br>" +

        d + "<br>" +

        e + "<br>" +

        f + "<br>" +

        g;

    }

</script>

</body>

</html>
```

## Convert Number to String:

**toString() :-**

The toString() method converts a number to a string.

### Syntax

```
number.toString(radix)
```

### Parameter Values

Parameter	Description
radix	Optional. Which base to use for representing a numeric value. Must be an integer between 2 and 36.  2 - The number will show as a binary value  8 - The number will show as an octal value  16 - The number will show as an hexadecimal value

### Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click the button to display the formatted number.</p>
```

```
<button onclick="myFunction()">Try it</button>
```

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

```
<script>
```

```
function myFunction()
```

```
{
```

```
    var num = 15;
```

```
    var n = num.toString();
```

```
        document.getElementById("demo").innerHTML = n;

    }

</script>

</body>

</html>
```

**Example: to display the formatted numbers**

```
<html>

<body>

    <p>Click the button to display the formatted numbers.</p>

    <button onclick="myFunction()">Try it</button>

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

<script>

    function myFunction()

    {

        var num = 15;

        var a = num.toString();

        var b = num.toString(2);

        var c = num.toString(8);

        var d = num.toString(16);

        var n = a + "<br>" + b + "<br>" + c + "<br>" + d;

        document.getElementById("demo").innerHTML=n;

    }

</script>

</body>

</html>
```

### 2.4.8 Changing the case of string:

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase():

#### Example

```
<!DOCTYPE html>

<html>

<body>

    <p>Convert string to upper case:</p>

    <button onclick="myFunction()">Try it</button>

    <p id="demo">Hello World!</p>

    <script>

        function myFunction()

        {

            var text = document.getElementById("demo").innerHTML;

            document.getElementById("demo").innerHTML = text.toUpperCase();

        }

    </script>

</body>

</html>
```

A string is converted to lower case with toLowerCase():

#### Example

```
var text1 = "Hello World!";    // String
var text2 = text1.toLowerCase(); // text2 is text1 converted to lower
```

### 2.4.9 Finding a Unicode of character:

The charCodeAt() Method:



The `charCodeAt()` method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

### Example

```
var str = "HELLO WORLD";
str.charCodeAt(0);    // returns 72
```

### Example

```
var str = "HELLO WORLD";
str[0] = "A";        // Gives no error, but does not work
str[0];              // returns H
```

If you want to work with a string as an array, you can convert it to an array.

## The fromCharCode():

### Definition and Usage

The `fromCharCode()` method converts Unicode values into characters.

Note: This is a static method of the String object, and the syntax is always `String.fromCharCode()`.

### Syntax

`String.fromCharCode(n1, n2, ..., nX)`

### Parameter Values

Parameter	Description
<code>n1, n2, ..., nX</code>	Required. One or more Unicode values to be converted

### Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click the button to display characters of the specified unicode numbers.</p>
```

```
<button onclick="myFunction()">Try it</button>

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

<script>

function myFunction() {

    var res = String.fromCharCode(72, 69, 76, 76, 79);

    document.getElementById("demo").innerHTML = res;

}

</script>

</body>

</html>
```

## Extra String Methods:-

### String.trim()

The trim() method removes whitespace from both sides of a string:

#### Example

```
var str = "    Hello World!    ";
alert(str.trim());
```

## Replacing String Content

The replace() method replaces a specified value with another value in a string:

#### Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

The replace() method does not change the string it is called on. It returns a new string.

By default, the replace() method replaces **only the first** match:

#### Example

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

By default, the replace() method is case sensitive. Writing MICROSOFT (with upper-case) will not work:

**Example**

```
str = "Please visit Microsoft!";  
var n = str.replace("MICROSOFT", "W3Schools");
```