Array, Function & Strings

Unit - II

Array in Java Script

- An *array is an ordered collection of values. Each value is called an element, and each* element has a numeric position in the array, known as its *index.*

- *JavaScript arrays are untyped: an array element may be of any type, and different elements of the same array* may be of different types.

- Array elements may even be objects or other arrays, which allows you to create complex data structures, such as arrays of objects and arrays of arrays.

Array in Java Script

- JavaScript arrays are *dynamic: they grow or shrink* as needed and there is no need to declare a fixed size for the array when you create it or to reallocate it when the size changes.

- JavaScript arrays may be *sparse: the elements* need not have contiguous indexes and there may be gaps.

Creating Array in Java Script

- The easiest way to create an array is with an array literal, which is simply a comma separated list of array elements within square brackets. For example:

```
var empty = []; // An array with no elements
var primes = [2, 3, 5, 7, 11]; // An array with 5 numeric elements
var misc = [ 1.1, true, "a", ]; // 3 elements of various types + trailing comma
```

- Array literals can contain object literals or other array literals:

```
var b = [[1,{x:1, y:2}], [2, {x:3, y:4}]];
```

Creating Array in Java Script

- If you omit a value from an array literal, the omitted element is given the value undefined:

```
var count = [1,,3]; // An array with 3 elements, the middle one undefined.

var undefs = [,,]; // An array with 2 elements, both undefined.
```

- Array literal syntax allows an optional trailing comma, so [,,] has only two elements, not three.

Creating Array in Java Script

- Another way to create an array is with the Array() constructor. You can invoke this constructor in three distinct ways:
- Call it with no arguments:

```
var a = new Array();
```

- This method *creates an empty array with no elements* and is equivalent to the array literal [].

- Call it with a single numeric argument, which specifies a length:

```
var a = new Array(10);
```

This technique creates an array with the specified length.

Creating Array in Java Script

- Explicitly specify two or more array elements or a single non-numeric element for the array:

```
var a = new Array(5, 4, 3, 2, 1, "testing, testing");
```

- In this form, the constructor arguments become the elements of the new array.

- Using an array literal is almost always simpler than this usage of the Array() constructor.

Reading Array in Java Script

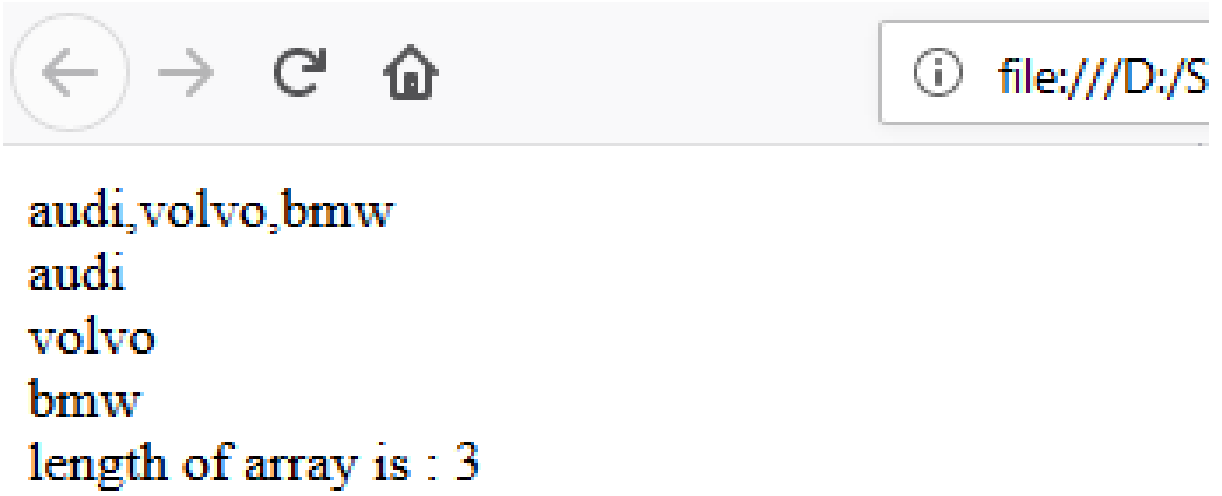
- You access an element of an array using the **[]** operator. A reference to the array should appear to the left of the brackets.

- You can use this syntax to both read and write the value of an element of an array. Thus, the following are all legal JavaScript statements:

```
var a = ["world"]; // Start with a one-element array
var value = a[0]; // Read element 0
a[1] = 3.14; // Write element 1
i = 2;
a[i] = 3; // Write element 2
a[i + 1] = "hello"; // Write element 3
```

Creating Array in Java Script

```
<html>
  <head>
        <title>
              Create an Array
        </title>
  </head>
  <body>
        <script>
              var cars=["audi","volvo","bmw"];
              document.write(cars+"<br>");
              document.write(cars[0]+"<br>");
              document.write(cars[1]+"<br>");
              document.write(cars[2]+"<br>");
              document.write("length of array is : "+cars.length);
        </script>
  </body>
</html>
```



Sparse Array in Java Script

- A *sparse array is one in which the elements do not have contiguous indexes starting at* 0.

- Normally, the length property of an array specifies the number of elements in the array.
- If the array is sparse, the value of the length property is greater than the number of elements.

- Sparse arrays can be created with the Array() constructor or simply by assigning to an array index larger than the current array length.

Sparse Array in Java Script

E.g. a = new Array(5); // No elements, but a.length is 5.

a = []; // Create an array with no elements and length = 0.

a[1000] = 0; // Assignment adds one element but sets length to 1001.

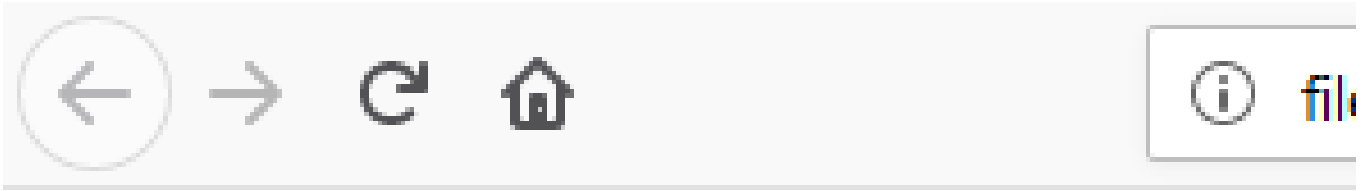
- Note that when you omit value in an array literal, you are not creating a sparse array. The omitted element exists in the array and has the value undefined.

E.g. var a1 = [,,,]; // This array is [undefined, undefined, undefined]

var a2 = new Array(3); // This array has no values at all

Sparse Array in Java Script

```
<html>
  <head>
        <title>
              Sparse Array
        </title>
  </head>
  <body>
        <script>
              var a=[1];
              //document.write(a.length+"<br>");
              a[10]=2;
              document.write(a.length+"<br><br>");
              document.write(a);
        </script>
  </body>
</html>
```



Array Length in Java Script

- Every array has a length property, and it is this property that makes arrays different from regular JavaScript objects.

- For arrays that are dense (i.e., not sparse), the length property specifies the number of elements in the array. Its value is one more than the highest index in the array:

E.g.

```
[].length // => 0: the array has no elements
['a','b','c'].length // => 3: highest index is 2, length is 3
```

Array Length in Java Script

- The second special behavior that arrays implement in order to maintain the length invariant is that *if you set the length property to a non-negative integer n smaller than its current value, any array elements whose index is greater than or equal to n are deleted from the array*:

```
a = [1,2,3,4,5]; // Start with a 5-element array.
a.length = 3; // a is now [1,2,3].
a.length = 0; // Delete all elements. a is [].
a.length = 5; // Length is 5, but no elements, like new Array(5)
```

Array Length in Java Script

- You can also set the length property of an array to a value larger than its current value.

- Doing this does not actually add any new elements to the array, it simply creates a sparse area at the end of the array.

Array Length in Java Script

- You can make the length property of an array read-only with Object.defineProperty()

e.g.

```
a = [1,2,3]; // Start with a 3-element array.
Object.defineProperty(a, "length", // Make the length property
{writable: false}); // readonly.
a.length = 0; // a is unchanged.
```

Adding & Deleting Array Elements in Java Script

- We've already seen the simplest way to add elements to an array: just assign values to new indexes:

```
a = [] // Start with an empty array.
a[0] = "zero"; // And add elements to it.
a[1] = "one";
```

Adding & Deleting Array Elements in Java Script

- You can also use the *push()* method to add one or more values to the end of an array:

e.g. a = []; // Start with an empty array

a.push("zero") // Add a value at the end. a = ["zero"]

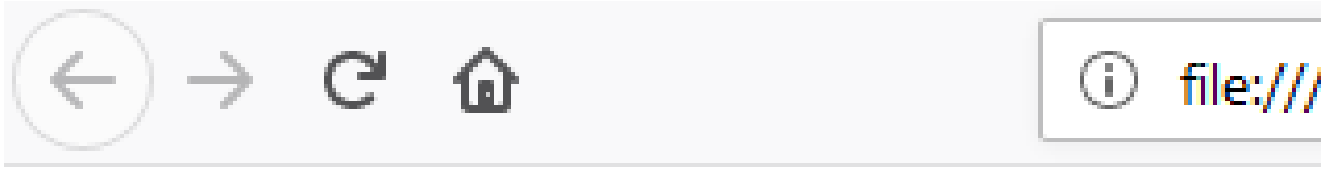
a.push("one", "two") // Add two more values. a = ["zero", "one", "two"]

Adding & Deleting Array Elements in Java Script

- You can use the *unshift()* method to insert a value at the beginning of an array, shifting the existing array elements to higher indexes.

Insert an Element in Array in Java Script

```
<html>
  <head>
        <title>
        Add an element using push and unshift
        </title>
  </head>
  <body>
        <script>
              var a=["red","green","blue"];
              a.push("black");
              document.write(a+"<br>");
              a.unshift("white");
              document.write(a);
        </script>
  </body>
</html>
```



red,green,blue,black
white,red,green,blue,black

Adding & Deleting Array Elements in Java Script

- You can delete array elements with the *delete* operator, just as you can delete object properties:

```
a = [1,2,3];
delete a[1]; // a now has no element at index 1
1 in a // => false: no array index 1 is defined
a.length // => 3: delete does not affect array length
```

- Deleting an array element is similar to (but subtly different than) assigning undefined to that element.
- Note that using delete on an array element does not alter the length property and does not shift elements with higher indexes down to fill in the gap that is left by the deleted property.

Adding & Deleting Array Elements in Java Script

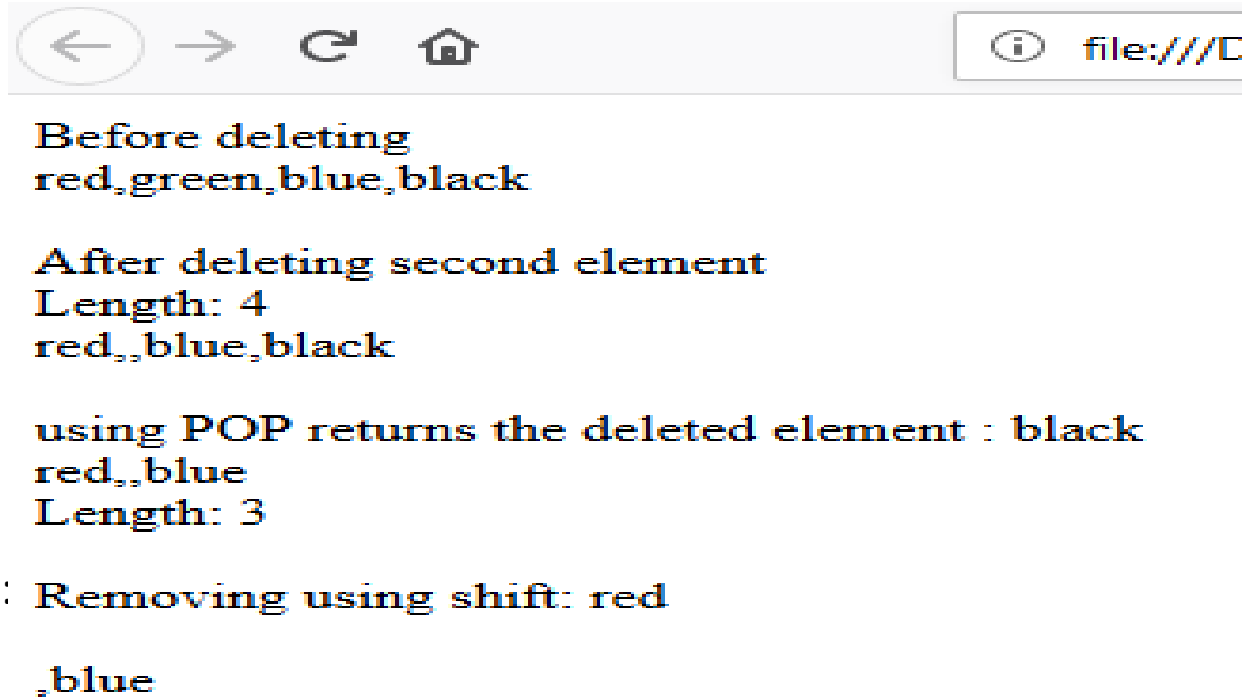
- Arrays have a *pop()* method that reduces the length of an array by 1 but also returns the value of the deleted element.

- There is also a *shift()* method to remove an element from the beginning of an array.

- Unlike delete, the shift() method shifts all elements down to an index one lower than their current index.

Remove an Array Element in Java Script

```
<html>
   <head><title>
            Delete an element using delete, pop and shift
   </title></head>
   <body><script>
   var a=["red","green","blue","black"];
   document.write("Before deleting <br>"+a+"<br><br>");
   delete a[1];
   document.write("After deleting second element <br>Length: "+a.length+"<br>");
   document.write(a+"<br><br>");
   document.write("using POP returns the deleted element : "+a.pop()+"<br>");
   document.write(a+"<br> Length: "+a.length+"<br><br>");
   document.write("Removing using shift: "+a.shift());
   document.write("<br><br>");
   document.write(a+"<br><br>");
   </script></body>
</html>
```



Remove an Array Element in Java Script

- Finally, *splice()* is the general-purpose method for inserting, deleting, or replacing array elements. It alters the length property and shifts array elements to higher or lower indexes as needed.
- Syntax:

array.splice(index,howmany,item1, item2, …., itemx);

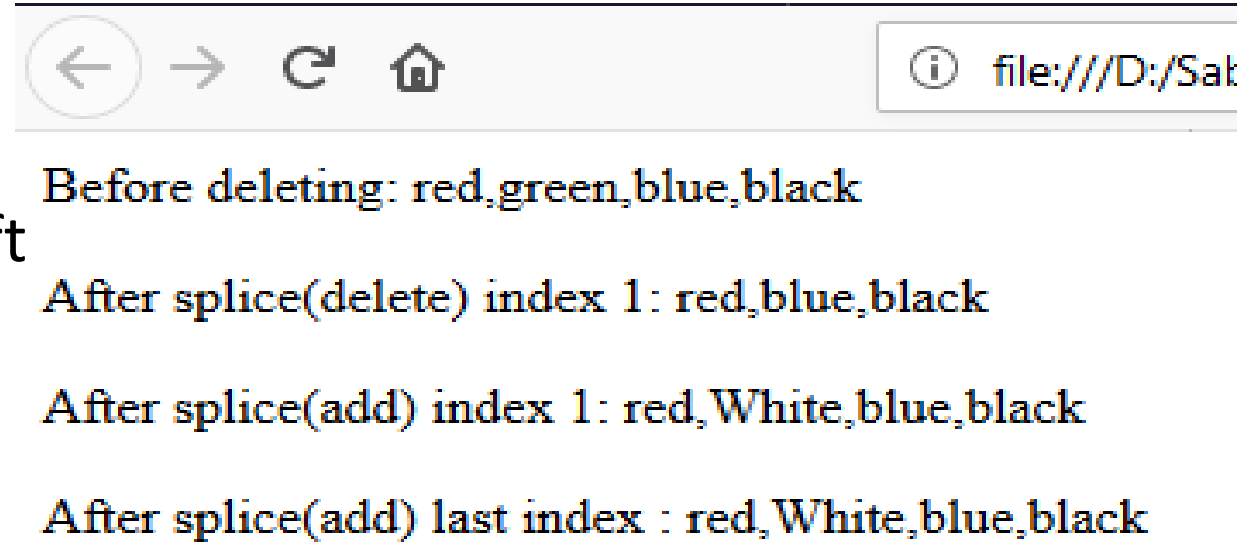
Index: requires an integer value that specifies at what position to add/remove item. use –ve value to specify the position from the end.

Howmany: Optional. Specifies the no. of items to be removed. If set to 0 no items will be removed.

Item: Optional. New items to be added to the array.

Remove an Array Element in Java Script

```
<html>
  <head><title>
  Delete an element using delete, pop and shift
  </title></head>
  <body><script>
  var a=["red","green","blue","black"];
  document.write("Before deleting: "+a+"<br><br>");
  a.splice(1,1);
  document.write("After splice(delete) index 1: "+a+"<br><br>");
  a.splice(1,0,"White");
  document.write("After splice(add) index 1: "+a+"<br><br>");
  document.write("After splice(add) last index : "+a+"<br><br>");
  </script></body>
</html>
```



Sort an Array Element in Java Script

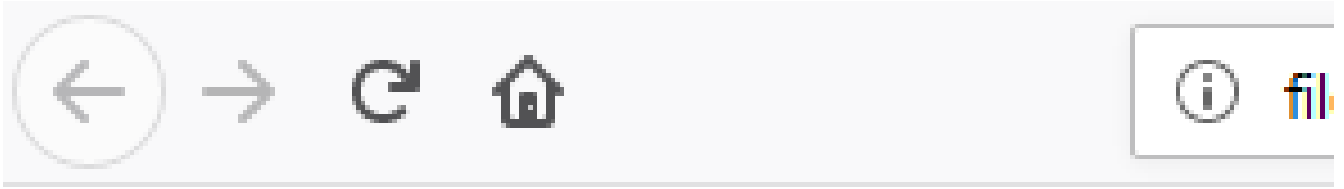
- To sort an array element, *sort()* function is used.
- Syntax:

  array.sort();

- Sorts the array numerically or alphabetically.

Sort an Array Element in Java Script

```
<html>
  <head>
        <title>
              Create an Array
        </title>
  </head>
  <body>
        <script>
              var cars=[6,5,4];
              document.write(cars+"<br>");
              document.write(cars.sort()+"<br>");
              var car1=["john","jim","max"];
              document.write(car1+"<br>");
              document.write(car1.sort()+"<br>");
        </script>
  </body>
</html>
```



6,5,4
4,5,6
john,jim,max
jim,john,max

Functions

- A *function is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times.*

- JavaScript functions are *parameterized:* a function definition may include a list of identifiers, known as *parameters, that* work as local variables for the body of the function.

- Function invocations provide values, or *arguments, for the function's parameters. Functions often use their argument* values to compute a *return value that becomes the value of the function-invocation* expression.

Difference between Functions & Methods

- Java Script is an object oriented scripting language. So everything here works with respect to objects.

- In JavaScript, there's always a global object defined. In a web browser, when scripts create global variables, they're created as members of the ***global object***.

- A function in java script does not belong to any object.
- But in JavaScript there is always a default global object. In HTML the default global object is the HTML page itself, so the function above "belongs" to the HTML page.
- In a browser the page object is the browser ***window***. So the function automatically becomes a window function.

Difference between Functions & Methods

- *this*:
  - In JavaScript, the thing called this, is the object that "owns" the current code.

- *Global Object*:
  - When a function is called without an owner object, the value of this becomes the global object.
  - In a web browser the global object is the browser window.

Difference between Functions & Methods

e.g.:

```
function myFunction(a, b)
{
    return a * b;
}

myFunction(10, 2);   or window.myFunction(10,2)       // Will return 20
```

Difference between Functions & Methods

- In java script, an object is a collection of key:value pairs.
- If a value is primitive (int, string, Boolean etc) or another object , the value is considered as property.
- If a value is function, it is a method.
- A *method* is a function, which is a property of an object. Means a method is associated with an object created by programmer where a function is not.

Difference between Functions & Methods

e.g.:

```
var person =
{
        firstName: "John",
        lastName : "Doe",
        id       : 5566,
        fullName : function()
    {
       return this.firstName + " " + this.lastName;
    }
};
```

Defining Functions

- Functions are defined with the *function* keyword, which can be used in a function definition expression or in a function declaration statement.

- In either form, function definitions begin with the keyword function followed by these components:

  - An identifier that names the function. The name is a required part of function declaration statements: For function definition expressions, the name is optional: if present, the name refers to the function object only within the body of the function itself.

Defining Functions

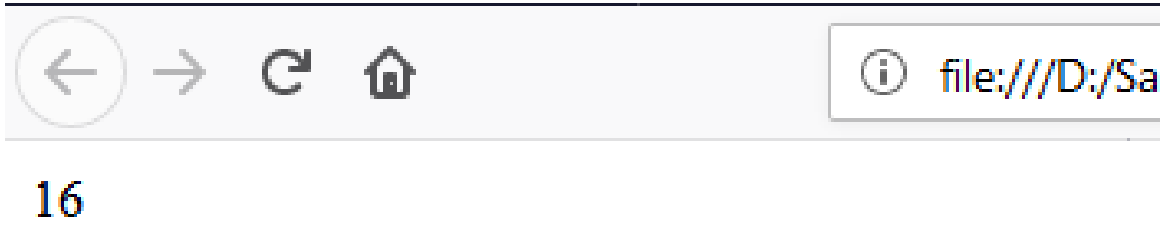
- A pair of parentheses around a comma-separated list of zero or more identifiers. These identifiers are the parameter names for the function, and they behave like local variables within the body of the function.

- A pair of curly braces with zero or more JavaScript statements inside. These statements are the body of the function: they are executed whenever the function is invoked.

Defining Functions

```
<html>
  <head><title>
            Functions
  </title></head>
  <body>
      <script>
            document.write(myfunction(4,4));

            function myfunction(a,b)
            {
                  return(a*b);
            }
      </script>
  </body>
</html>
```



Invoking Functions

- The JavaScript code that makes up the body of a function is not executed when the function is defined but when it is invoked.
- JavaScript functions can be invoked in four ways:
    - as functions,
    - as methods,
    - as constructors

Function Invocation

- A function can be invoked with the function name only.

- Since a function is not associated with any user-defined object but with a global object (window).

- Syntax:

    function_name(parameter list);

Function Invocation

```
<html>
  <body>
  <h2>JavaScript Functions</h2>
  <script>
  function myFunction(a, b) {
    return a * b;
  }
  Document.writemyFunction(10,2)
  </script>
  </body>
</html>
```

JavaScript Functions

20

Method Invocation

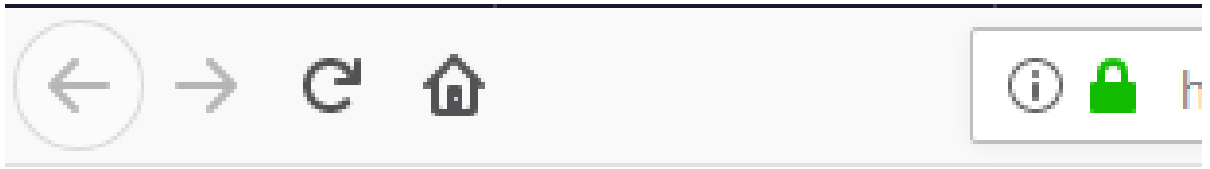
- A method is function, which is a property of an object.

- Syntax:

  object.method_name(parameter_list);

Method Invocation

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



JavaScript Functions

John Doe

Constructor Invocation

- If a function invocation is preceded with the *new* keyword, it is a constructor invocation.

- It looks like you create a new function, but since JavaScript functions are objects you actually create a new object:

- Syntax:

    var name = new function_name(parameter_list);

Constructor Invocation

```
<html>
  <body>
  <h2>JavaScript Constructor</h2>
  <script>
  function myFunction(arg1, arg2) {
    this.firstName = arg1;
    this.lastName  = arg2;
  }
  var x = new myFunction("John","Doe")
  document.write(x.firstName);
  </script>
  </body>
</html>
```

call() Method

- Call() method is predefined Java Script method.

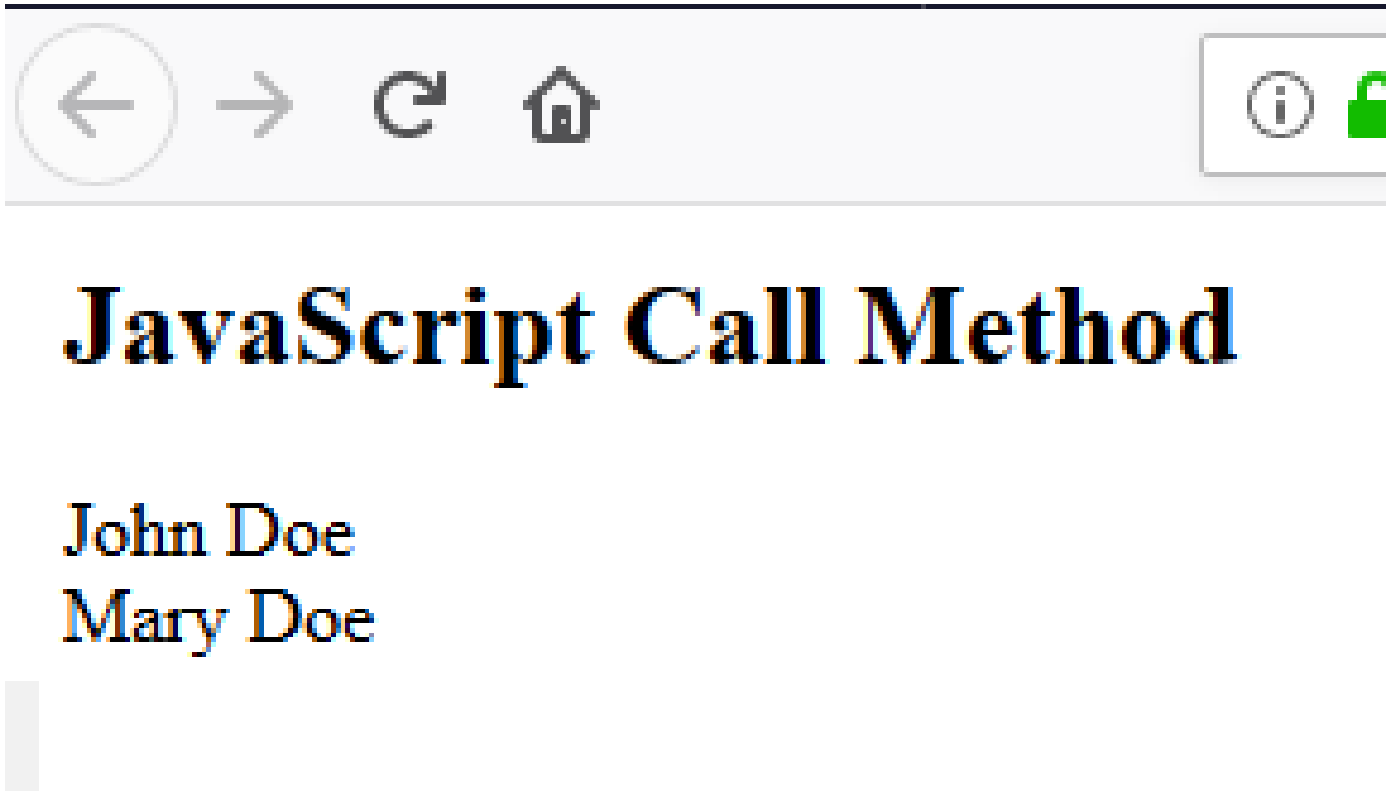
- It can be used to invoke a method with an owner object as an argument (parameter).

- With call(), an object can use a method belonging to another object.
- Syntax:

object.function_name.call(owner_object,parameter_list);

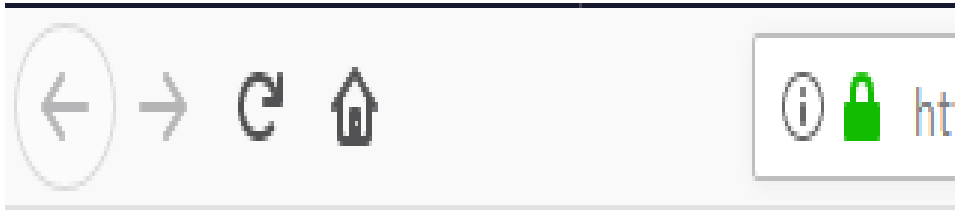
call() Method

```
<html>
   <body>
   <h2>JavaScript Call Method</h2>
   <script>
   var person = {
    fullName: function() {
      return this.firstName + " " + this.lastName;
    }}
   var person1 = {
    firstName:"John",
    lastName: "Doe" }
   var person2 = {
    firstName:"Mary",
    lastName: "Doe" }
   document.write(person.fullName.call(person1)+"<br>");
   document.write(person.fullName.call(person2));
   </script>
   </body>
</html>
```



call() Method with arguments

```
<html>
  <body>
  <h2>JavaScript Functions</h2>
  <script>
  var person = {
   fullName: function(city, country) {
     return this.firstName + " " + this.lastName + "," + city + "," + country;
   }}
  var person1 = {
   firstName:"John",
   lastName: "Doe“ }
  var person2 = {
   firstName:"Mary",
   lastName: "Doe“ }
  document.write(person.fullName.call(person1, "Oslo", "Norway"));
  </script>
  </body>
</html>
```



JavaScript Functions

John Doe,Oslo,Norway

Strings

- Primitive values, like “Hello World", cannot have properties or methods (because they are not objects).

- But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

Concat()

- This method joins two or more strings.
- It can be used instead of + operator. Both will do the same thing.

- Syntax:

  string1.concat(string2,string3,...)

- E.g.:

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
document.write(text3);
```

charAt(x)

- Returns the character at the “x” position within the string.

- Syntax:

  variable = charAt(position)

- E.g.

```
var myString = ‘Hello World!!!';
document.write(myString.charAt(7));
//output: W
```

indexOf()

- This method returns the index of (position of) the first occurrence of a specified text in a string or a character.
- Returns -1 if the text is not found.

- Syntax:

  indexOf(string/character, start_position);

  - Where: string or character is the item to be searched
  - Start_position: starting position to start the search with

- E.g.: var pos = str.lastIndexOf("John");

lastIndexOf()

- This method returns the index of (position of) the last occurrence of a specified text in a string or a character.
- Returns -1 if the text is not found.
- It searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.
- Syntax:

  indexOf(string/character, start_position);

  - Where: string or character is the item to be searched
  - Start_position: starting position to start the search with
- E.g.: var pos = str.lastIndexOf("John");

split()

- The split() method is used to split a string into an array of substrings, and returns the new array.

- If an empty string ("") is used as the separator, the string is split between each character.

- The split() method does not change the original string.

split()

- Syntax: *string.split(separator, limit);*

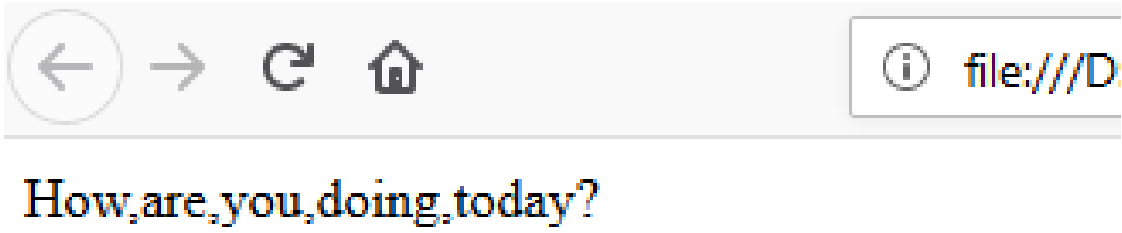
  - Where:

  - **separator** is optional. Specifies the character, or the regular expression, to use for splitting the string. If omitted, the entire string will be returned (an array with only one item)

    Limit: Optional. An integer that specifies the number of splits, items after the split limit will not be included in the array

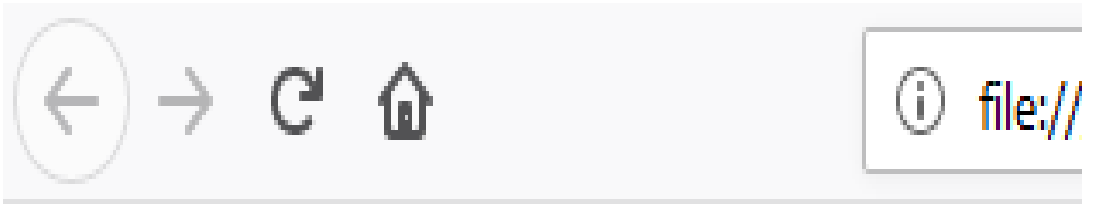
split()

```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "How are you doing today?"
          var res = str.split(" ");
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



split()

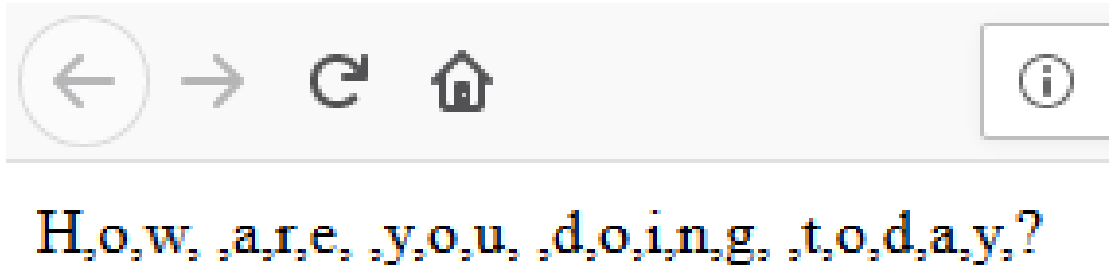
```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "How are you doing today?";
          var res = str.split(" ", 3);
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



How,are,you

split()

```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "How are you doing today?";
          var res = str.split(“”);
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



substring()

- The substring() method extracts the characters from a string, between two specified indices, and returns the new sub string.

- This method extracts the characters in a string between "start" and "end", not including "end" itself.

- If "start" is greater than "end", this method will swap the two arguments, meaning str.substring(1, 4) == str.substring(4, 1).

- If either "start" or "end" is less than 0, it is treated as if it were 0

substring()

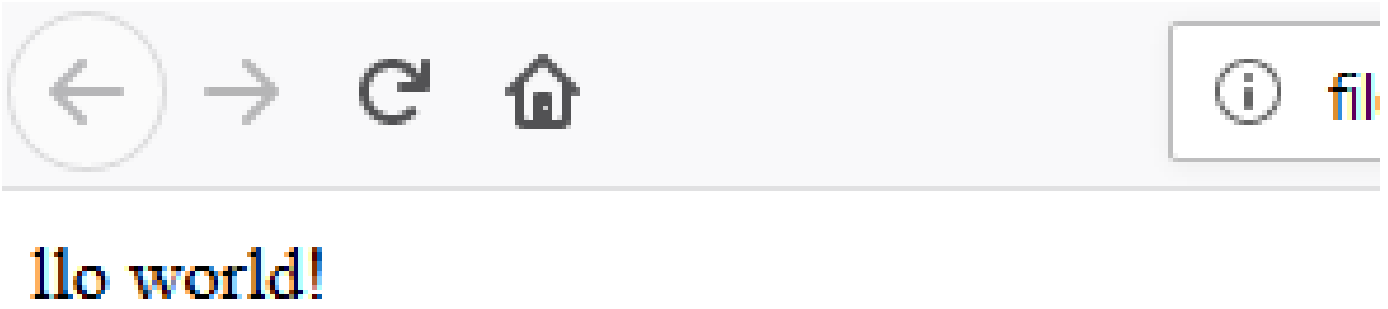
- Syntax: *string.substring(start, end)*

  - *Where:*
    start: Required. The position where to start the extraction. First character is at index 0

  - *End:* Optional. The position (up to, but not including) where to end the extraction. If omitted, it extracts the rest of the string

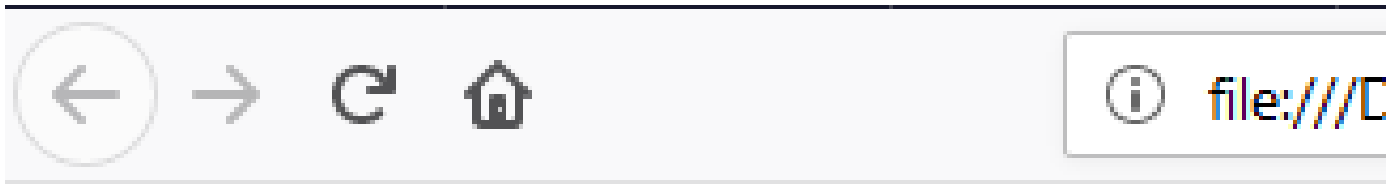
substring()

```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "Hello world!";
          var res = str.substring(2);
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



substring()

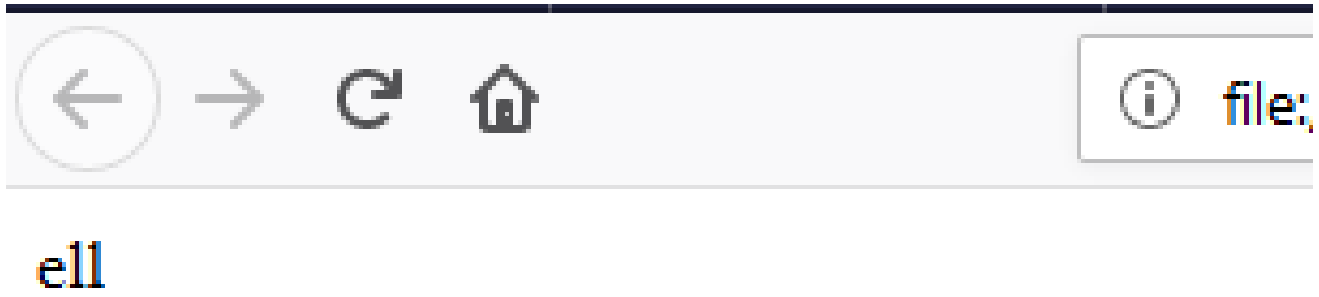
```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "Hello world!";
          var res = str.substring(1,4);
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



ell

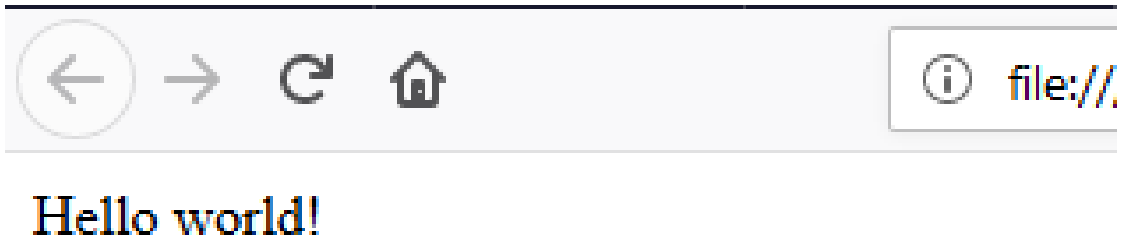
substring()

```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "Hello world!";
          var res = str.substring(4,1);
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



substring()

```
<html>
  <body>
        <script>
        function myFunction() {
          var str = "Hello world!";
          var res = str.substring(-3);
          document.write(res);
        }
        myFunction();
        </script>
  </body>
</html>
```



slice()

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

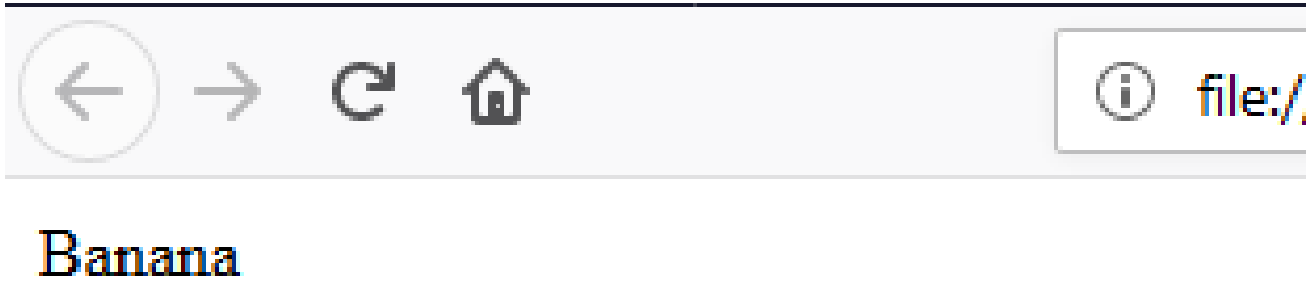
- The method takes 2 parameters: the start position, and the end position (end not included).

- Syntax: *slice(start, end)*
  - *Where*
    start: Required. The position where to start the extraction. First character is at index 0

  - *End:* Optional. The position (up to, but not including) where to end the extraction. If omitted, it extracts the rest of the string

slice()

```
<html>
  <body>
        <script>
        var str = "Apple, Banana, Kiwi";
        var res = str.slice(7,13);
        document.write(res);
        </script>
  </body>
</html>
```



slice()

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

- For *start* as negative, it is treated as *strLength* + (*beginIndex*) where *strLength* is the length of the string (for example, if *beginIndex* is -3 it is treated as *strLength* - 3).

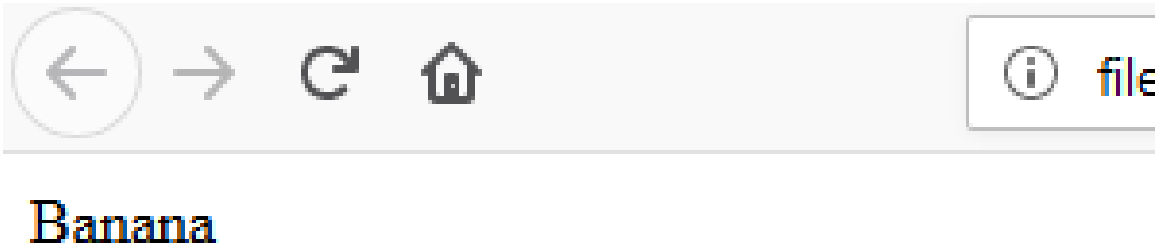
- For *end* negative, it is treated as *strLength* + *endIndex* where *strLength* is the length of the string (for example, if *endIndex* is -3 it is treated as *strLength* - 3)

- e.g. var str = 'The quick brown fox jumps over the lazy dog.';

  document.write(str.slice(-9, -5));// expected output: "lazy"

slice()

```
<html>
  <body>
        <script>
        var str = "Apple, Banana, Kiwi";
        var res = str.slice(-12,-6);
        document.write(res);
        </script>
  </body>
</html>
```



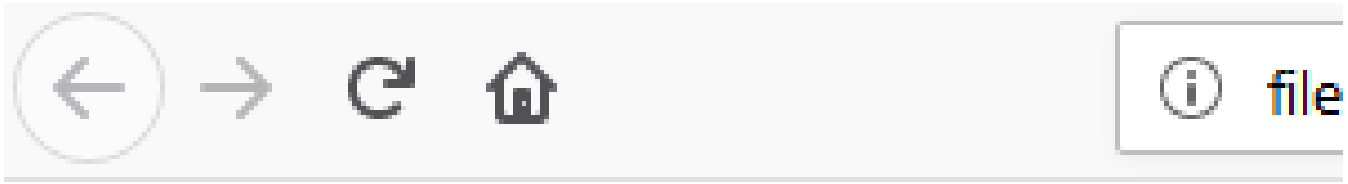
String to number

- Number():
  - The Number() function converts the object argument to a number that represents the object's value.
  - If the value cannot be converted to a legal number, NaN is returned.
  - Syntax: *Number(object)*
    - Where: object Optional. A JavaScript object. If no argument is provided, it returns 0.

String to number

Number():

```
<html><body><script>
function myFunction() {
  var x1 = true;
  var x2 = false;
  var x3 = new Date();
  var x4 = "999";
  var x5 = "999 888";
  var n = Number(x1) + "<br>" +
  Number(x2) + "<br>" +
  Number(x3) + "<br>" +
  Number(x4) + "<br>" +
  Number(x5);
  document.write(n);} myFunction();
</script></body></html>
```



1
0
1563461152160
999
NaN

String to number

- parseInt():
  - 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)

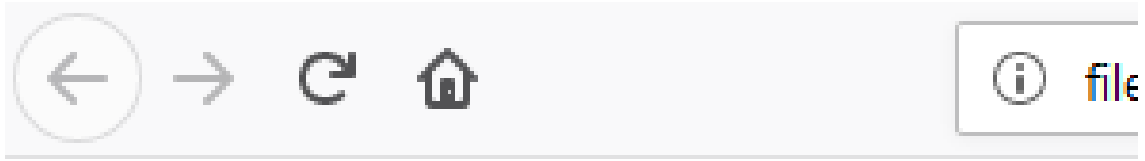
String to number

- parseInt():
  - Only the first number in the string is returned!
  - Leading and trailing spaces are allowed.
  - If the first character cannot be converted to a number, parseInt() returns NaN.
  - Syntax: *parseInt(string, radix)*
    - Where: *string* Required. The string to be parsed
    - *radix* Optional. A number (from 2 to 36) that represents the numeral system to be used

String to number

```
<html>
<body><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.write(n);}
    myFunction();
</script></body>
</html>
```



10
10
10
34
60
40
NaN

10
10
8
16
16

String to number

- parseFloat():
  - 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.
  - Only the first number in the string is returned!
  - Leading and trailing spaces are allowed.
  - If the first character cannot be converted to a number, parseFloat() returns NaN.

String to number

- parseFloat():
  - Syntax: parseFloat(*string*)
  - Where: *string* Required. The string to be parsed

String to number

```
<html>
<body><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.write(
  a + "<br>" +  b + "<br>" +
  c + "<br>" +  d + "<br>" +
  e + "<br>" +  f + "<br>" +
  g);}myFunction()
</script></body>
</html>
```

10
10
10.33
34
60
40
NaN

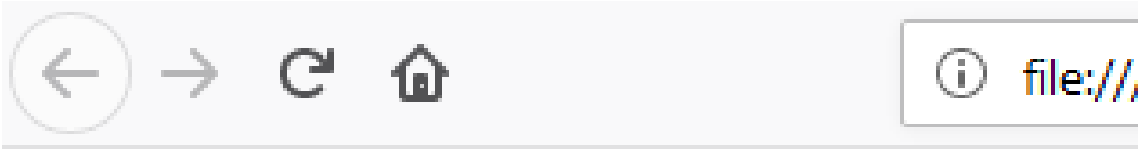
Number to String

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

- Syntax: *number.toString(radix)*
  - Where:
    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

Number to String

```
<html>
<body>
<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.write(n);
}
myFunction();
</script>
</body>
</html>
```



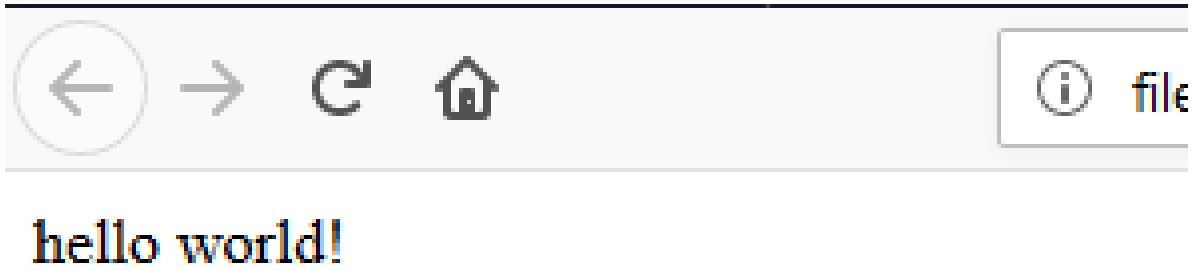
15
1111
17
f

Changing to Lower Case

- The toLowerCase() method converts a string to lowercase letters.
- It does not change the original string.
- Syntax: *string.toLowerCase()*
  - Method accepts no parameter.

Changing to Lower Case

```
<html>
  <body>
  <script>
        function myFunction() {
          var str = "Hello World!";
          var res = str.toLowerCase();
          documentwrite(res);
        }
        myFunction();
  </script>
  </body>
</html>
```



Changing to Upper Case

- The toUpperCase() method converts a string to uppercase letters.
- The toUpperCase() method does not change the original string.
- Syntax: *string.toUpperCase()*
  - It accepts no parameter

Changing to Upper Case

```
<html>
  <body>
  <script>
        function myFunction() {
          var str = "Hello World!";
          var res = str.toUpperCase();
          document.write(res);
        }
        myFunction();
  </script>
  </body>
</html>
```

Unicode of the Character

- The charCodeAt() method returns the Unicode of the character at the specified index in a string.
- The index of the first character is 0, the second character 1, and so on.
- You can use the charCodeAt() method together with the length property to return the Unicode of the last character in a string.
- The index of the last character is -1, the second last character is -2, and so on (See Example below).

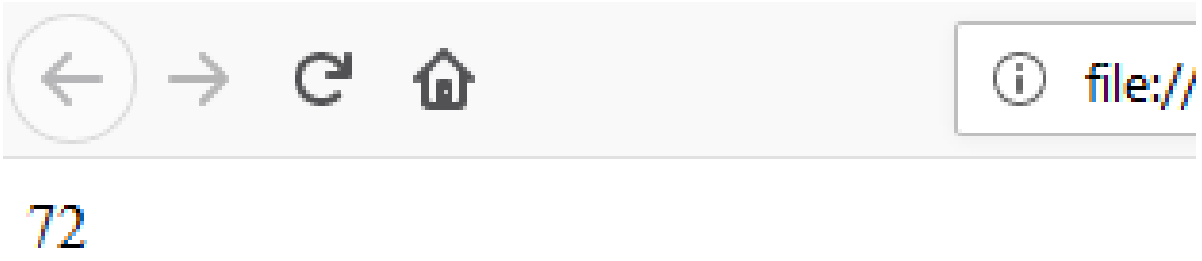
Unicode of the Character

- Syntax: *string*.charCodeAt(*index*)

- Where:
  index: Required. A number representing the index of the character you want to return

Unicode of the Character

```
<html>
  <body>
        <script>
              function myFunction() {
                var str = "HELLO WORLD";
                var n = str.charCodeAt(0);
                document.write(n);
              }
              myFunction();
        </script>
  </body>
</html>
```



Character of a Unicode

- The fromCharCode() method converts Unicode values into characters.
- This is a static method of the String object, and the syntax is always String.fromCharCode().
- Syntax: *String.fromCharCode(n1, n2, ..., nX)*
- Where:
  n1, *n2*, ..., *nX:* Required. One or more Unicode values to be converted

Character of a Unicode

```
<html>
  <body>
        <script>
              function myFunction() {
                var res = String.fromCharCode(65,70);
                document.write(res);
              }
              myFunction();
        </script>
  </body>
</html>
```