```
var jsDataTypes = ["string", "number", "boolean", "undefined", "null",
  "symbol", "object"]
/*
String - anything wrapped in single or double quotes
Number - whole number or decimal number
Boolean - true or false
undefined - variable that is declared but not assigned a value
null - absense of any real value
symbol - data type to create unique keys in objects // not gonna learn
object - store multiple things with property value pairs
  >> array = special type of object to store many things in order
*/

```

String Properties and Methods

Primitive values, like "John Doe", 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.

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a new joined strings
endsWith()	Checks whether a string ends with specified string/characters
fromCharCode()	Converts Unicode values to characters
includes()	Checks whether a string contains the specified string/characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
repeat()	Returns a new string with a specified number of copies of an existing string
replace()	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced
search()	Searches a string for a specified value, or regular expression, and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

JavaScript Numbers

JavaScript has only one type of number.

Numbers can be written with, or without, decimals:

Property	Description
constructor	Returns the function that created JavaScript's Number prototype
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)
prototype	Allows you to add properties and methods to an object

Method	Description
isFinite()	Checks whether a value is a finite number
isInteger()	Checks whether a value is an integer
isNaN()	Checks whether a value is Number.NaN
isSafeInteger()	Checks whether a value is a safe integer
toExponential(x)	Converts a number into an exponential notation
toFixed(x)	Formats a number with x numbers of digits after the decimal point
toPrecision(x)	Formats a number to x length
toString()	Converts a number to a string
valueOf()	Returns the primitive value of a number

Boolean Properties

Property	Description
constructor	Returns the function that created JavaScript's Boolean prototype
prototype	Allows you to add properties and methods to the Boolean prototype

Boolean Methods

Method	Description
toString()	Converts a boolean value to a string, and returns the result
valueOf()	Returns the primitive value of a boolean

JavaScript Object Properties

JavaScript Properties

Properties are the values associated with a JavaScript object.

A JavaScript object is a collection of unordered properties.

Properties can usually be changed, added, and deleted, but some are read only.

Accessing JavaScript Properties

The syntax for accessing the property of an object is:

```
objectName.property       // person.age
```

or

```
objectName["property"]    // person["age"]
```

or

```
objectName[expression]    // x = "age"; person[x]
```

The **this** Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the **fullName** function.

In other words, **this.firstName** means the **firstName** property of **this object**.

Read more about the `this` keyword at JS this Keyword.

JavaScript Methods

JavaScript methods are actions that can be performed on objects.

A JavaScript **method** is a property containing a **function definition**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

Banana,Orange,Apple,Mango

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

Popping

The `pop()` method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();              // Removes the last element ("Mango") from fruits
```

Pushing

The `push()` method adds a new element to an array (at the end):

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits
```

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();            // Removes the first element "Banana" from fruits
```

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

The `unshift()` method returns the new array length.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");     // Returns 5
```

Changing Elements

Array elements are accessed using their **index number**:

> Array **indexes** start with 0. [0] is the first array element, [1] is the second, [2] is the third ...

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";        // Changes the first element of fruits to "Kiwi"
```

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator `delete`:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];          // Changes the first element in fruits to undefined
```

Splicing an Array

The `splice()` method can be used to add new items to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Try it Yourself »

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);         // Removes the first element of fruits
```

Try it Yourself »

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys
```

Try it Yourself »

> The `concat()` method does not change the existing arrays. It always returns a new array.

The `concat()` method can take any number of array arguments:

Example (Merging Three Arrays)

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);   // Concatenates arr1 with arr2 and arr3
```

Try it Yourself »

The `concat()` method can also take values as arguments:

Example (Merging an Array with Values)

```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
```

Try it Yourself »

Slicing an Array

The `slice()` method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

Try it Yourself »

Automatic toString()

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Try it Yourself »

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
```

Try it Yourself »