# Javascript

NON-PRIMITIVE TYPES

# Javascript datatypes: Object

- Object is a non-primitive data type in JavaScript.
- An object holds multiple values in terms of properties and methods.
  - Properties can hold values of primitive data types
  - Methods are functions.

```
var person = {
                firstName: "James",
                lastName: "Bond",
                age: 25,
                getFullName: function () {
                    return this.firstName + ' ' + this.lastName
                    }
            };

person.firstName; // returns James
person.lastName; // returns Bond

person["firstName"];// returns James
person["lastName"];// returns Bond

person.getFullName();
```

# Javascript Core Objects

JavaScript provides a number of standard objects, with properties and methods to perform various manipulations. Core objects are independent of the client browser

| Object | Description |
|---|---|
| Array | The Array object as the name suggest, is used to create arrays. It has many methods to add, delete or extract elements from an array and sort them. |
| Boolean | The Boolean object is used to create Boolean values. Elements with two states: true and false. |
| Date | The Date object is used to create dates and time. It also offers methods to manipulate them. |
| Function | The Function object allows you to define custom functions. |
| Math | The Math object allows you to manipulate mathematical functions, such as trigonometric functions. |
| Number | The Number object allows you to perform basic operations on numbers.. |
| RegExp | The RegExp object allows you to create regular expressions. |
| String | The String object provides a variety of methods for manipulating strings. |

# Javascript Arrays

JavaScript arrays are used to store multiple values in a single variable.
There are two ways for creating an array:

- Using an array literal:

```
var cars = ["Saab", "Volvo", "BMW"];
```

- With the array constructor:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

The two examples above do exactly the same. There is no need to use new Array().
For simplicity, readability and execution speed, use the first one (the array literal method).

# Javascript Arrays

It is possible to create an empty array and then add the elements:

- Using an array literal:

```
var fruits=[];
fruits[0]='orange';
fruits[2]='apple';

alert (fruits[2]);
```

- Or with the array constructor:

```
var fruits=new Array();
fruits[0]='orange';
fruits[2]='apple';

alert (fruits[1]); //Undefined
```

# More about Javascript Arrays

Defining the size of an array? No use

```
var fruits=new Array(3);
fruits[0]='orange';
fruits[1]='mandarine';
fruits[2]='apple';
fruits[3]='pear';
fruits[4]='strawberry';

alert (fruits[4]); //strawberry
```

Multidimensional arrays:

```
var board=new Array();
board[0]=new Array('Pepe', 'Juan', 'Belen');
board[1]=new Array('Tom','Mary','Ann');

for (i=0;i<board.length;i++)  {
    for (j=0;j<board[i].length;j++)  {
        document.write(board[i][j]+';');
    }
}
```

And be careful…

```
var fruits=new Array(3);
alert(fruits[0]);  //undefined

var fruits=new Array('3');
alert(fruits[0]);  //3
```

```
var fruits=new Array();
fruits[0] ='orange';
fruits[1] ='mandarine';
fruits[2] ='apple';

alert(fruits[2]); // apple
fruits.length=2;
alert(fruits[2]); // undefined
```

```
var points = new Array(40, 100);  // Creates an array with two elements (40 and 100)
var points = new Array(40);       // Creates an array with 40 undefined elements !!!!!
```

# Javascript Arrays

- Access the Elements of an Array:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

- Access the full array:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

- Array Elements Can Be Objects
  - Because of this, you can have variables of different types in the same Array.

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

# Javascript Arrays

- Adding array elements:
  - The easiest way to add a new element to an array is using the push method:

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

  - New element can also be added to an array using the length property:

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

# Javascript Arrays

- Looping array elements:

```
var fruits, text, fLen, i;

fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;
text = "<ul>";
for (i = 0; i < fLen; i++) {
    text += "<li>" + fruits[i] + "</li>";
}
```

# Array.prototype methods: forEach()

- The forEach() method executes a provided function once for each array element
- (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach)

```
const array1 = ['a', 'b', 'c'];
array1.forEach(element => console.log(element));

const copyArray1 = [];
array1.forEach(item => copyArray1.push(item));
console.log("Copied array: "+copyArray1); //a,b,c

copyArray1.forEach((item, index) => {
  console.log(item);
  if (index === 2) {
    copyArray1.shift(); //Deletes de first element from the array
  }});
console.log("Shifted array: "+copyArray1); //b,c
```

# Array.prototype methods: Map()

- The map() method creates a new array populated with the results of calling a provided function on every element in the calling (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)

```
const array1 = [1, 4, 9, 16];
const map1 = array1.map(x => x * 2);
console.log(map1);  //[2, 8, 18, 32]

const array2 = ['1', '4', '9', '16'];
const map2 = array2.map(Number);
console.log(map2); // [1, 4, 9, 16]

const map3 = array1.map((num, index) => {
  if (index < 2) return num;
  else return 0;});
console.log(map3);  //[1, 4, 0, 0]
```

# Javascript Arrays: Methods

| | |
|---|---|
| **concat()** | Returns a new array comprised of this array joined with other array(s) and/or value(s). |
| **every()** | Returns true if every element in this array satisfies the provided testing function. |
| **filter()** | Creates a new array with all of the elements of this array for which the provided filtering function returns true. |
| **forEach()** | Calls a function for each element in the array. |
| **indexOf()** | Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found. |

| | |
|---|---|
| **join()** | Joins all elements of an array into a string. |
| **lastIndexOf()** | Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found. |
| **map()** | Creates a new array with the results of calling a provided function on every element in this array. |
| **pop()** | Removes the last element from an array and returns that element. |
| **push()** | Adds one or more elements to the end of an array and returns the new length of the array. |

# Javascript Arrays: Methods

| | |
|---|---|
| **reduce()** | Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value. |
| **reduceRight()** | Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value. |
| **reverse()** | Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first. |
| **shift()** | Removes the first element from an array and returns that element. |
| **slice()** | Extracts a section of an array and returns a new array. |
| **some()** | Returns true if at least one element in this array satisfies the provided testing function. |

| | |
|---|---|
| **toSource()** | Represents the source code of an object |
| **sort()** | Sorts the elements of an array |
| **splice()** | Adds and/or removes elements from an array. |
| **toString()** | Returns a string representing the array and its elements. |
| **unshift()** | Adds one or more elements to the front of an array and returns the new length of the array. |

Examples:
https://www.w3schools.com/js/js_array_methods.asp

# Javascript Arrays

- Difference between arrays and objects:
  - In JavaScript, objects use named indexes.
  - Arrays are a special kind of objects, with numbered indexes.
    - If you use a named index, JavaScript will redefine the array to a standard object.
    - After that, all array methods and properties will produce incorrect results.

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;
var y = person[0];
```

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;          // person.length will return 0
var y = person[0];              // person[0] will return undefined
```

# Javascript String objects: Properties & methods

| Property | Description |
| --- | --- |
| length | Returns the length of the string. |

| Method | Description |
| --- | --- |
| charAt(position) | Returns the character at the specified position (in Number). |
| charCodeAt(position) | Returns a number indicating the Unicode value of the character at the given position (in Number). |
| concat([string,,]) | Joins specified string literal values (specify multiple strings separated by comma) and returns a new string. |
| indexOf(SearchString, Position) | Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found. |

# Javascript String objects: Methods

| Method | Description |
| --- | --- |
| lastIndexOf(SearchString, Position) | Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found. |
| localeCompare(string,position) | Compares two strings in the current locale. |
| match(RegExp) | Search a string for a match using specified regular expression. Returns a matching array. |
| replace(searchValue, replaceValue) | Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue. |
| search(RegExp) | Search for a match based on specified regular expression. |
| slice(startNumber, endNumber) | Extracts a section of a string based on specified starting and ending index and returns a new string. |
| split(separatorString, limitNumber) | Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator. |

# Javascript String objects: Methods

| Method | Description |
|---|---|
| substr(start, length) | Returns the characters in a string from specified starting position through the specified number of characters (length). |
| substring(start, end) | Returns the characters in a string between start and end indexes. |
| toLocaleLowerCase() | Converts a string to lower case according to current locale. |
| toLocaleUpperCase() | Converts a sting to upper case according to current locale. |
| toLowerCase() | Returns lower case string value. |
| toString() | Returns the value of String object. |
| toUpperCase() | Returns upper case string value. |
| valueOf() | Returns the primitive value of the specified string object. |

Examples: https://www.w3schools.com/js/js_string_methods.asp

# Javascript Math objects: Properties

JavaScript provides 8 mathematical constants that can be accessed with the Math object:

```
Math.E        // returns Euler's number
Math.PI       // returns PI
Math.SQRT2    // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2      // returns the natural logarithm of 2
Math.LN10     // returns the natural logarithm of 10
Math.LOG2E    // returns base 2 logarithm of E
Math.LOG10E   // returns base 10 logarithm of E
```

# Javascript Math objects: Methods

| Method | Description |
|---|---|
| abs(x) | Returns the absolute value of x |
| acos(x) | Returns the arccosine of x, in radians |
| asin(x) | Returns the arcsine of x, in radians |
| atan(x) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| atan2(y, x) | Returns the arctangent of the quotient of its arguments |
| ceil(x) | Returns the value of x rounded up to its nearest integer |
| cos(x) | Returns the cosine of x (x is in radians) |
| exp(x) | Returns the value of $E^x$ |
| floor(x) | Returns the value of x rounded down to its nearest integer |

# Javascript Math objects: Methods

| Method | Description |
|---|---|
| log(x) | Returns the natural logarithm (base E) of x |
| max(x, y, z, ..., n) | Returns the number with the highest value |
| min(x, y, z, ..., n) | Returns the number with the lowest value |
| pow(x, y) | Returns the value of x to the power of y |
| random() | Returns a random number between 0 and 1 |
| round(x) | Returns the value of x rounded to its nearest integer |
| sin(x) | Returns the sine of x (x is in radians) |
| sqrt(x) | Returns the square root of x |
| tan(x) | Returns the tangent of an angle |

Examples: https://www.w3schools.com/js/js_math.asp

# Javascript Date objects

- A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.
- Date objects are created with the **new Date()** constructor.
- There are **4 ways** of initiating a date:

```
new Date() //Creates a new date object with the current date and time
new Date(dateString) //Creates a new date object from the specified date and time
new Date(milliseconds) //Creates a new date object as zero time plus the number
                       //Zero time is 01 January 1970 00:00:00 UTC
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

- Examples:

```
var d1 = new Date(); //Sun Apr 02 2017 10:46:03 GMT+0200
var d2 = new Date("October 13, 2014 11:13:00"); //Mon Oct 13 2014 11:13:00 GMT+0200
var d3 = new Date(86400000); //Fri Jan 02 1970 01:00:00 GMT+0100
var d4 = new Date(99, 5, 24, 11, 33, 30, 0); //Thu Jun 24 1999 11:33:30 GMT+0200
```

# Javascript Date objects: Format

- There are generally 4 types of JavaScript date input formats:

| Type | Example |
|---|---|
| ISO Date | "2015-03-25" (The International Standard) |
| Short Date | "03/25/2015" |
| Long Date | "Mar 25 2015" or "25 Mar 2015" |
| Full Date | "Wednesday March 25 2015" |

- Some other options:

```
var d1 = new Date("2015-03"); //Sun Mar 01 2015 01:00:00 GMT+0100
var d2 = new Date("2015"); //Thu Jan 01 2015 01:00:00 GMT+0100
var d3 = new Date("2015-03-25T12:00:00Z"); //Wed Mar 25 2015 13:00:00 GMT+0100
var d4 = new Date("JANUARY, 25, 2015"); //Sun Jan 25 2015 00:00:00 GMT+0100
var d = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (W. Europe Standard Time)"
        //Wed Mar 25 2015 09:56:24 GMT+0100
```

# Javascript Date objects: Methods

| Method | Description |
| --- | --- |
| getDate() | Get the day as a number (1-31) |
| getDay() | Get the weekday as a number (0-6) |
| getFullYear() | Get the four digit year (yyyy) |
| getHours() | Get the hour (0-23) |
| getMilliseconds() | Get the milliseconds (0-999) |
| getMinutes() | Get the minutes (0-59) |
| getMonth() | Get the month (0-11) |
| getSeconds() | Get the seconds (0-59) |
| getTime() | Get the time (milliseconds since January 1, 1970) |

# Javascript Date objects: Methods

| Method | Description |
|---|---|
| setDate() | Set the day as a number (1-31) |
| setFullYear() | Set the year (optionally month and day) |
| setHours() | Set the hour (0-23) |
| setMilliseconds() | Set the milliseconds (0-999) |
| setMinutes() | Set the minutes (0-59) |
| setMonth() | Set the month (0-11) |
| setSeconds() | Set the seconds (0-59) |
| setTime() | Set the time (milliseconds since January 1, 1970) |

Examples: https://www.w3schools.com/js/js_date_methods.asp