JavaScript

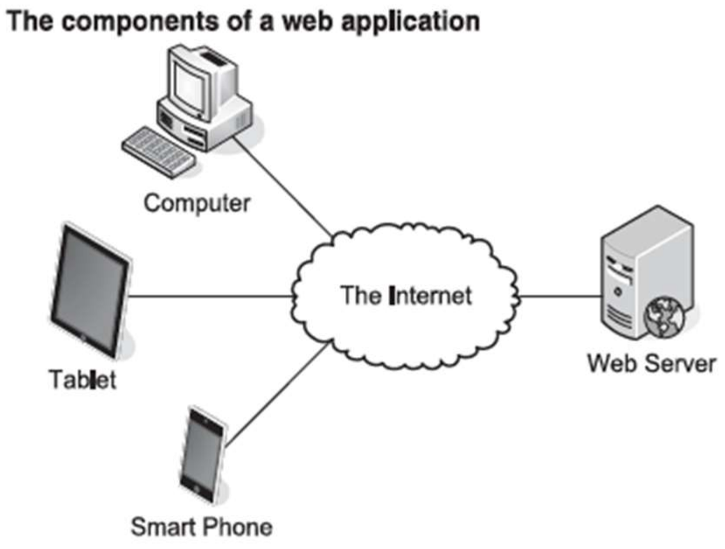
Client Side Web Programming

2021/2022

Jose Socuéllamos

How a web application works?

- Client user programs (web browsers): request web pages from the web server.
- Web server: returns the requested pages to the browser.



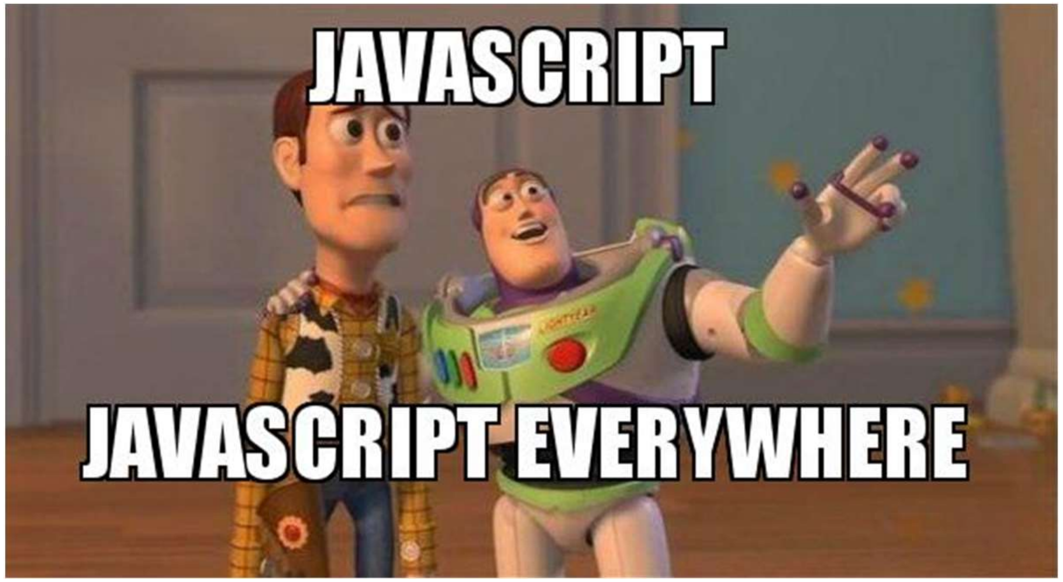
Static vs Dynamic webpages

- Static websites:
  - They are fixed and display the same content for every user, usually written exclusively in HTML.
  - They doesn’t change unless a developer modifies its code.
- Dynamic websites:
  - They can display different content and provide user interaction, by making use of advanced programming and databases in addition to HTML.
  - The final HTML code is created by a script on the browser (client-side) or on the web server (server-side) each time a webpage is requested.
  - The browser doesn’t know if the HTML is created dynamically or not, it just displays the final HTML code.

What is JavaScript???

- Programming language
  - High level
  - Interpreted → We don’t need to compile
  - Weakly typed
  - Born to be web
  - Based in objects, not object oriented → This is changing
  - Event oriented

What is JavaScript???

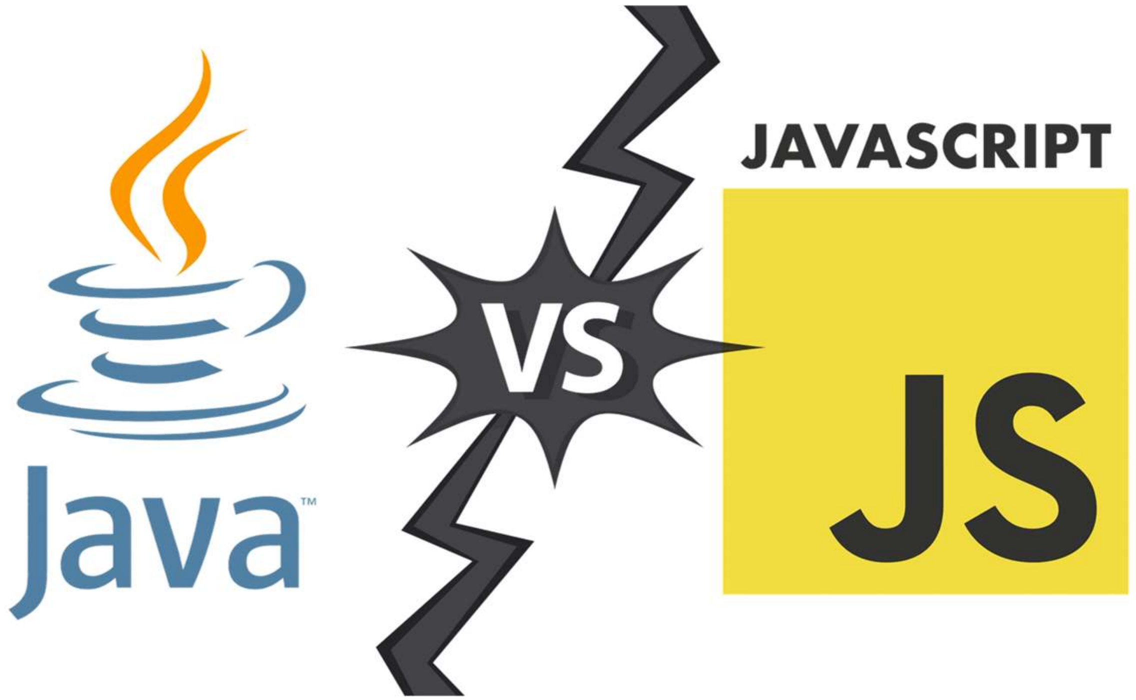


JavaScript Client

- Born to be run into the browser.
- It’s used to perform calculations:
  - Process interactions with the web and respond to events
  - Modify structure or style of a web “on the fly”
  - Server requests
  - Interactive games
  - etc.

JavaScript Frameworks

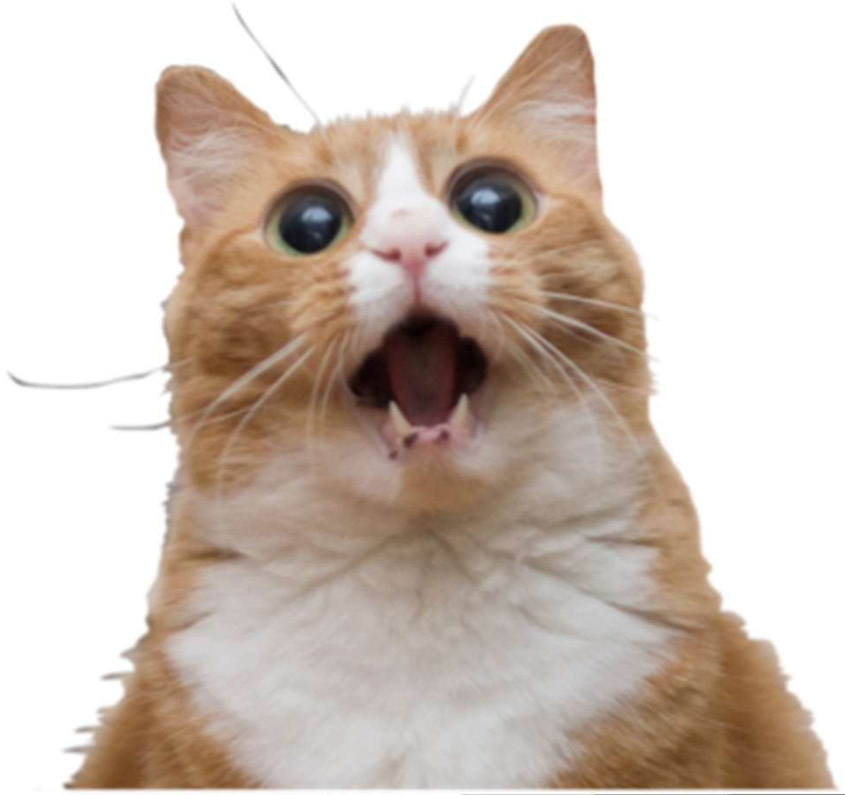
- JavaScript has gradually grown in popularity.
- JS frameworks are collections of JavaScript code libraries that provide developers with pre-written JS code to use for routine programming features and tasks.
- They're used to build websites or web applications around.
- Examples:
  - Responding requests in the server (NodeJS…)
  - Games (Phaser…)
  - Mobile applications (React Native, Ionic…)
  - Databases (MongoDB…)
  - APIs



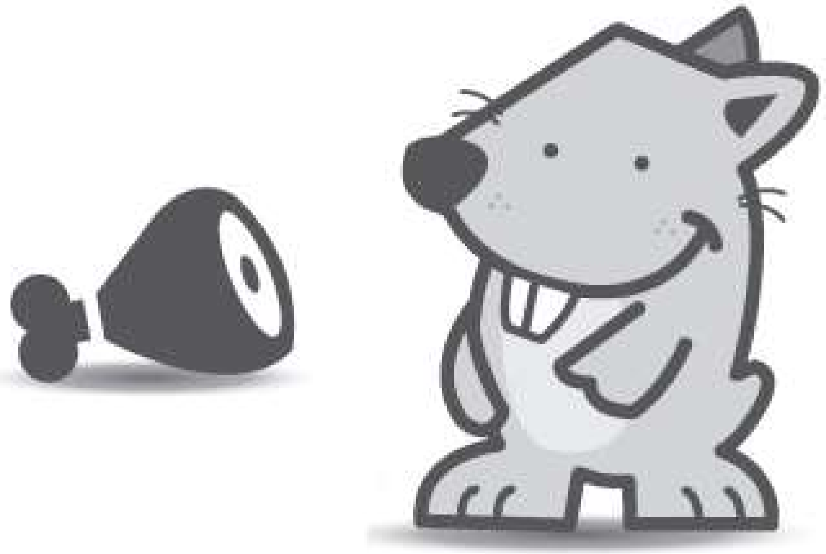
Are Java and JavaScript the same?

I have java
I have script
I have Javascript

JavaScript != Java



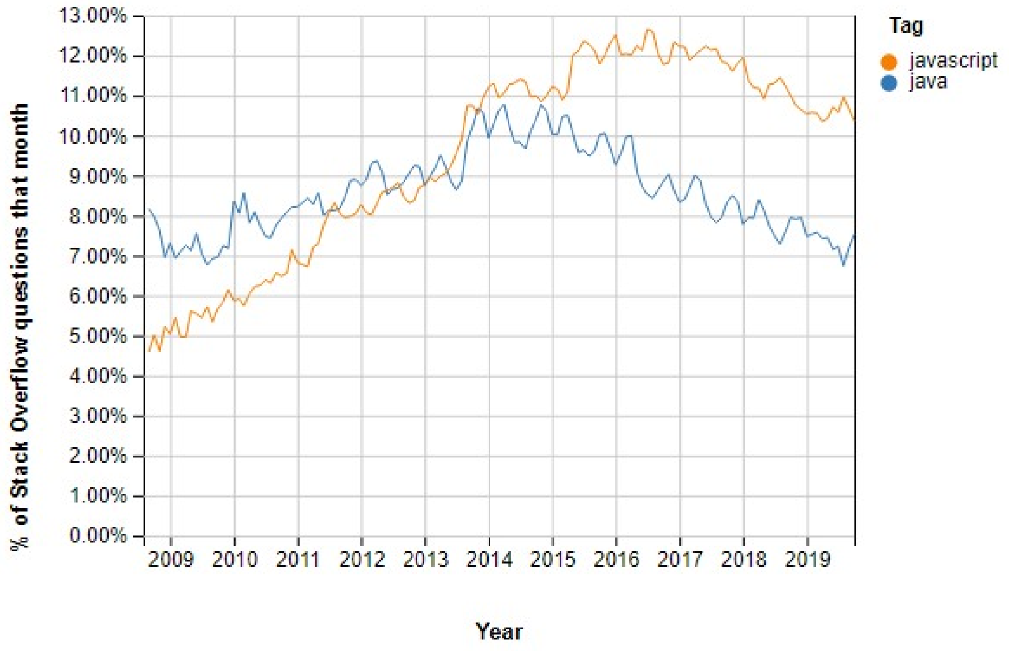
JAVA *is to* JAVASCRIPT *as* HAM *is to* HAMSTER



Java vs JavaScript

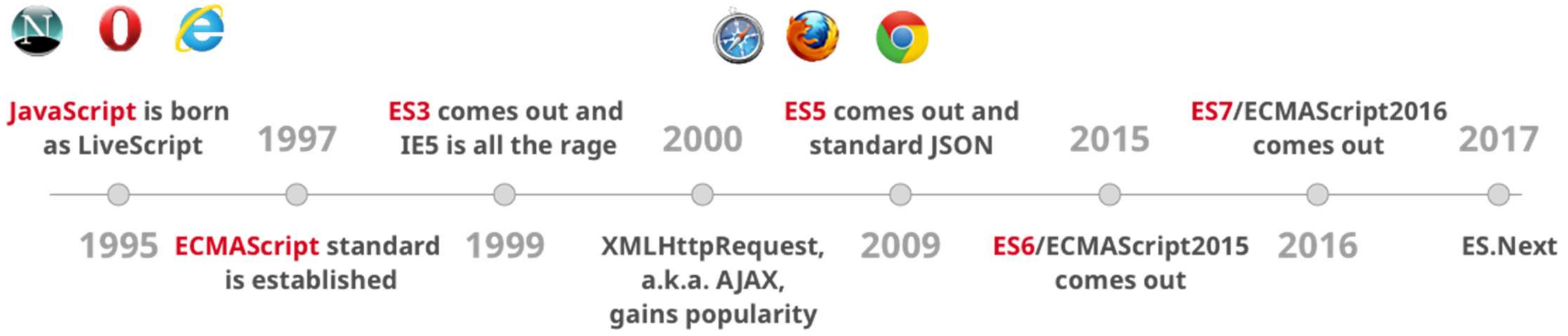
• Static	• Dynamic
• Strongly typed	• Weakly typed (no type)
• Object oriented	• Based/built in objects
• Compiled on the server before execution	• Interpreted by the client
• Strict	• A bit anarchic

Stack Overflow Questions
Java vs JavaScript



Old vs New JavaScript

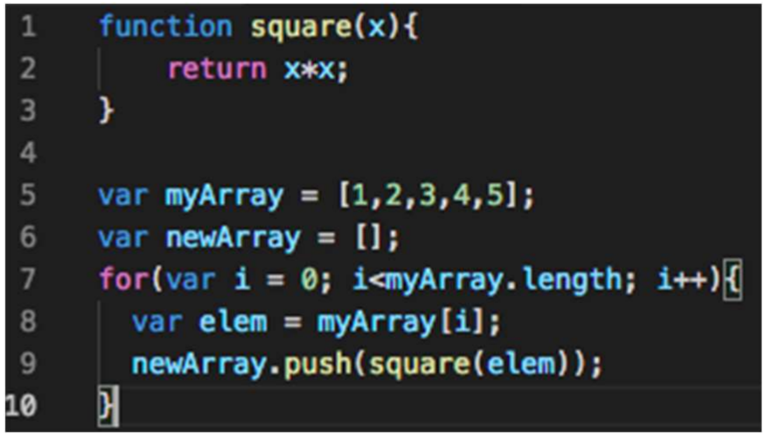
- JS is constantly growing…



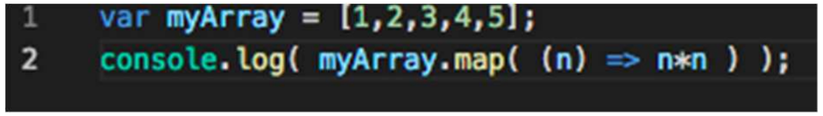
JavaScript is born as LiveScript
1995
ECMAScript standard is established
1997
1999
ES3 comes out and IE5 is all the rage
2000
XMLHttpRequest, a.k.a. AJAX, gains popularity
2009
ES5 comes out and standard JSON
2015
ES6/ECMAScript2015 comes out
2016
ES7/ECMAScript2016 comes out
2017
ES.Next

Old vs New JavaScript

- Problem: browsers don’t grow fast enough to handle these changes.
- Nowadays there are old and new versions of JS coexisting on the same websites.
- Newer versions of JS are closer to other modern scripting languages (e.g., Python) so they are much more productive.



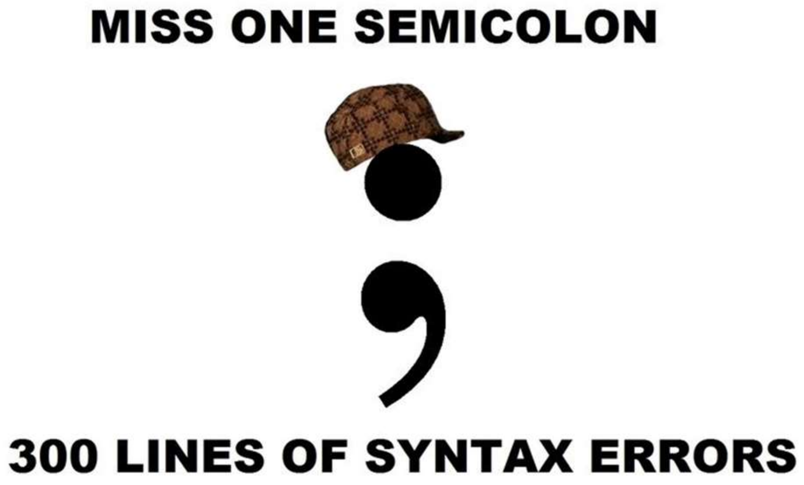
```javascript
function square(x){
    return x*x;
}

var myArray = [1,2,3,4,5];
var newArray = [];
for(var i = 0; i<myArray.length; i++){
  var elem = myArray[i];
  newArray.push(square(elem));
}
```



```javascript
var myArray = [1,2,3,4,5];
console.log( myArray.map( (n) => n*n ) );
```

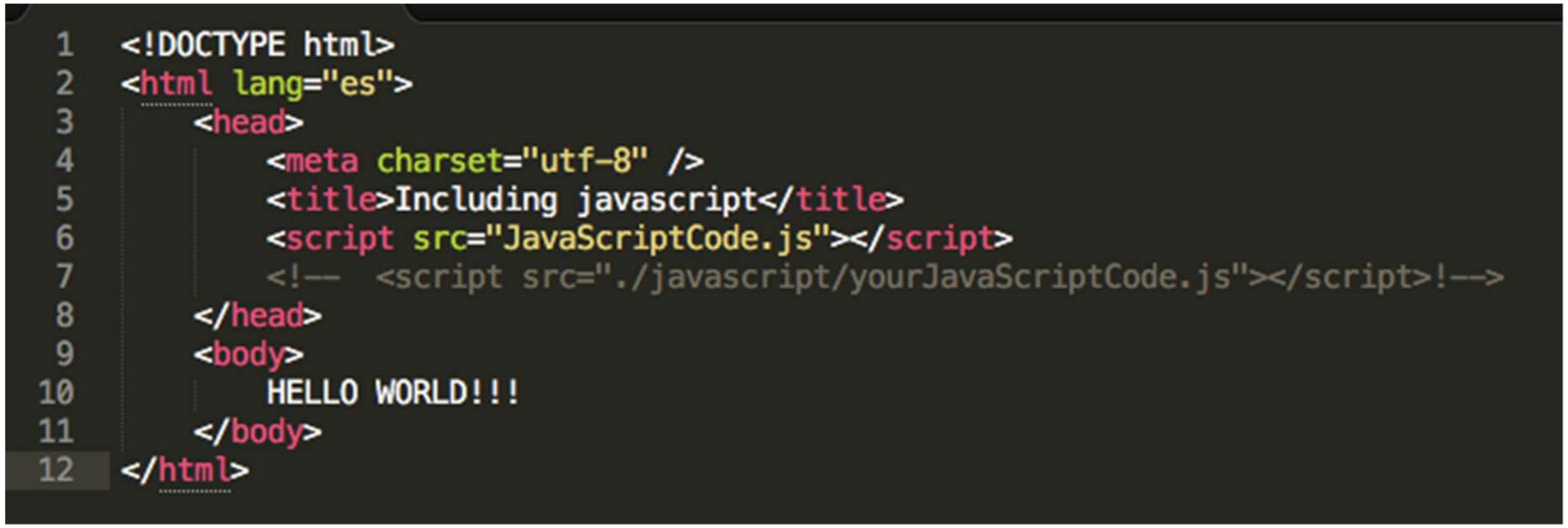
JavaScript syntax

- Some similarities: Rules, comments…
- Case sensitive: Simpsons != simpsons
- Ignores extra whitespaces
- Must end with semicolon



Including scripts in the HTML

- BEST OPTION: Include it as an external reference in the head sectionl



```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="utf-8" />
        <title>Including javascript</title>
        <script src="JavaScriptCode.js"></script>
        <!--  <script src="./javascript/yourJavaScriptCode.js"></script>!-->
    </head>
    <body>
        HELLO WORLD!!!
    </body>
</html>
```

Data types and variables

Data types (basics)

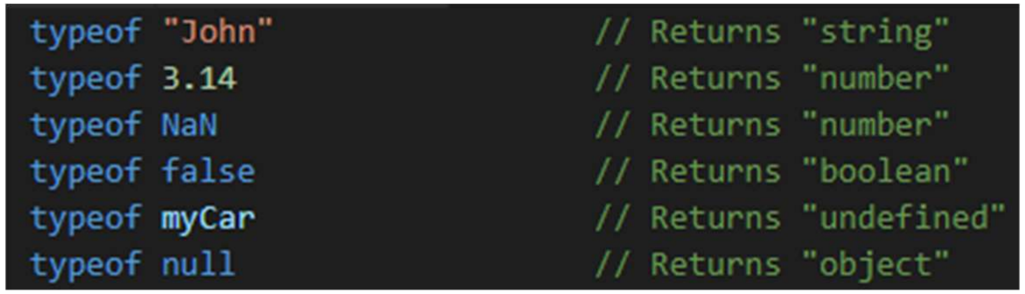
Type	Example(s)

Number	var x = 3.4;
	var x = 2;

String	‘Hi, I’m a text string’ “Hi, I’m a text string”
Boolean	true false
null	It means nothing, empty
undefined	It means not defined yet. Not the same than empty…

Data types (basics)

- In JavaScript there are 5 different data types that can contain values:
  - string
  - number
  - boolean
  - object (we’ll learn it later)
  - function (we’ll learn it later)
- You can use the **typeof** operator to find out the data type of a JavaScript variable.



```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof myCar                  // Returns "undefined"
typeof null                   // Returns "object"
```

Variables

- Global scope:

```
var x = 'Hello';
```

- Local scope:

```
const x = 'Hello';

let x = 'Hello';
```

Using let

```
let i = 43;
for (let i=0;i<10;i++){
    console.log(i);
}
console.log(i);
```

If we use let in a global scope, it will work as a **var** variable.

However, when we use **let** to declare a variable inside a block, this variable and its value only exist inside the block.

Same with **const**.

let vs const

- Both are new ways of declaring variables in JS.
- Specially interesting in local blocks because they don’t overwrite other global values or other visible variables.
- When do I use **let** or **const**?
  - If the value can change → let
  - If the value will never change→ const
    - If we try to change the value of a **const** variable we’ll get an error (TypeError)
    - Useful to debug

Comparators

Use of == and === (!= and !==)

- JavaScript has two equality comparators:
  - Weak equality ==
    - If data types are the same (4 == 3), then the values are compared and returns true only if they are identical..¡
    - If data types are not the same (4 == true) it will try to convert the more complex data into the simpler. After that, if both value are the same it will return true. Any number different than zero will be converted to true.
  - String equality ===
    - Two objects are only equal if they have the same type and value. Don't try to convert.
    - If you are not sure of the behaviour you need, this is your best option.

Working with strings

Working with strings

```
var name = "John Doe";
var age = 28;
console.log( "Hello, my name is ${name} and I am ${age} years old" );
```

Is the same than:

```
console.log( "Hello, my name is " + name + " and I am " + age + " years old" );
```

Data collections

Data types (complex)

Type	Example(s)

Array	let arrayA = [1,2,3,4,5];
	let arrayB = new Array();
	let arrayC = [];

TypedArray	let x = new Int32Arrays(2);
	x[0] = 32;
	x[1] = -1;

Set	let mySet = new Set();
	x.add(4);

Map	let x = new Map();
	x.set('foo',16);

Object	{}

Array methods & properties

Property	Explanation
length	Sets or returns the number of elements in an array

Method	Explanation
indexOf()	Search the array for an element and returns its position
shift()	Removes the first element of an array, and returns that element
pop()	Removes the last element of an array, and returns that element
unshift()	Adds new elements to the beginning of an array, and returns the new length
push()	Adds new elements to the end of an array, and returns the new length
splice()	Adds/Removes elements from an array
slice()	Selects a part of an array, and returns the new array

Conditionals

Conditional sentences (if…else if… else)

```
if (condition) {
        code block
} else if (condition2){
        code block
} else {
        code block
}
```

It is used to perform different actions depending on the condition that is met.

Loops

Loops: For (old)

```
for (let i=0; i<10; i++){
        code block
}
```

Used to repeat a block of sentences.

Loop **for** → Usually I know how many times I need to run the code block

Loops: While

```
while (condition){
        code block
}
```

Used to repeat a block of sentences.

Loop **while** → I don't know hoy many times I need to run the sentences. However I know that I need to run it while a condictions is granted.

Bucles: For-of (new)

```
let myArray = [1,2,3,4,5];

for (let value of myArray){
        code block
}
```

- Used to repeat a block of sentences.
- New type of loop. Requires a collection of objects or an iterable object.
- Requires a local variable (**value** in this case) where each iteration gets the value of one element from the collection.

Functions

Functions in JS

- Parts of a function:
  - Name: Identify what the function does.
  - Input data/Parameters: Information we need to give to get the function working.
  - Output data: The result we get when the function finishes.
  - Body of the function: Instructions/steps to run from the beginning until the end.

```
function square (x){
        let result = x*x;
        return result;
}
```

Functions in JS

- Parts of a function:
  - Name: Identify what the function does.
  - Input data/Parameters: Information we need to give to get the function working.
  - Output data: The result we get when the function finishes.
  - Body of the function: Instructions/steps to run from the beginning until the end.

```
function square (x){
    let result = x*x;
    return result;
}
```

Reserved word use to define de function

Functions in JS

- Parts of a function:
  - Name: Identify what the function does.
  - Input data/Parameters: Information we need to give to get the function working.
  - Output data: The result we get when the function finishes.
  - Body of the function: Instructions/steps to run from the beginning until the end.

```
function square (x){
        let result = x*x;
        return result;
}
```

Function name. Related with the purpose of it.

Functions in JS

- Parts of a function:
  - Name: Identify what the function does.
  - Input data/Parameters: Information we need to give to get the function working.
  - Output data: The result we get when the function finishes.
  - Body of the function: Instructions/steps to run from the beginning until the end.

```
function square (x){
    let result = x*x;
    return resu  ;
}
```

In parentheses and separated by commas, the entry data.

Functions in JS

- Parts of a function:
  - Name: Identify what the function does.
  - Input data/Parameters: Information we need to give to get the function working.
  - Output data: The result we get when the function finishes.
  - Body of the function: Instructions/steps to run from the beginning until the end.

```
function square (x){
        let result = x*x;
        return result;
}
```

In breackets, the body of the function.

Functions in JS

- Parts of a function:
  - Name: Identify what the function does.
  - Input data/Parameters: Information we need to give to get the function working.
  - Output data: The result we get when the function finishes.
  - Body of the function: Instructions/steps to run from the beginning until the end.

```
function square (x){
        let result = x*x;
        return result;
}
```

With return, we indicate what we are going to give to the calling instruction.

Functions in JS (Expansion operator)

```
function maximo(...args){
        let maxValue = undefined;
        for(let value of args){
                if(!maxValue || value > maxValue){
                        maxValue = value;
                }
        }
        return maxValue;
}

console.log(maximo(5,1,89));
console.log(maximo(89,23,-102,0));
```

Functions in JS (Expansion operator)

```
function maximo(...args){
        let maxValue = undefined;
        for(let value of args){
                if(!maxValue || value > maxValue){
                        maxValue = value;
                }
        }
        return maxValue;
}

console.log(maximo(5,1,89));
console.log(maximo(89,23,-102,0));
```

A variable number of arguments.

Functions in JS (Expansion operator)

```
function maximo(...args){
        let maxValue = undefined;
        for(let value of args){
                if(!maxValue || value > maxValue){
                        maxValue = value;
                }
        }
        return maxValue;
}

console.log(maximo(5,1,89));
console.log(maximo(89,23,-102,0));
```

Inside, we work with the args as an iterable object.

Functions in JS (Expansion operator)

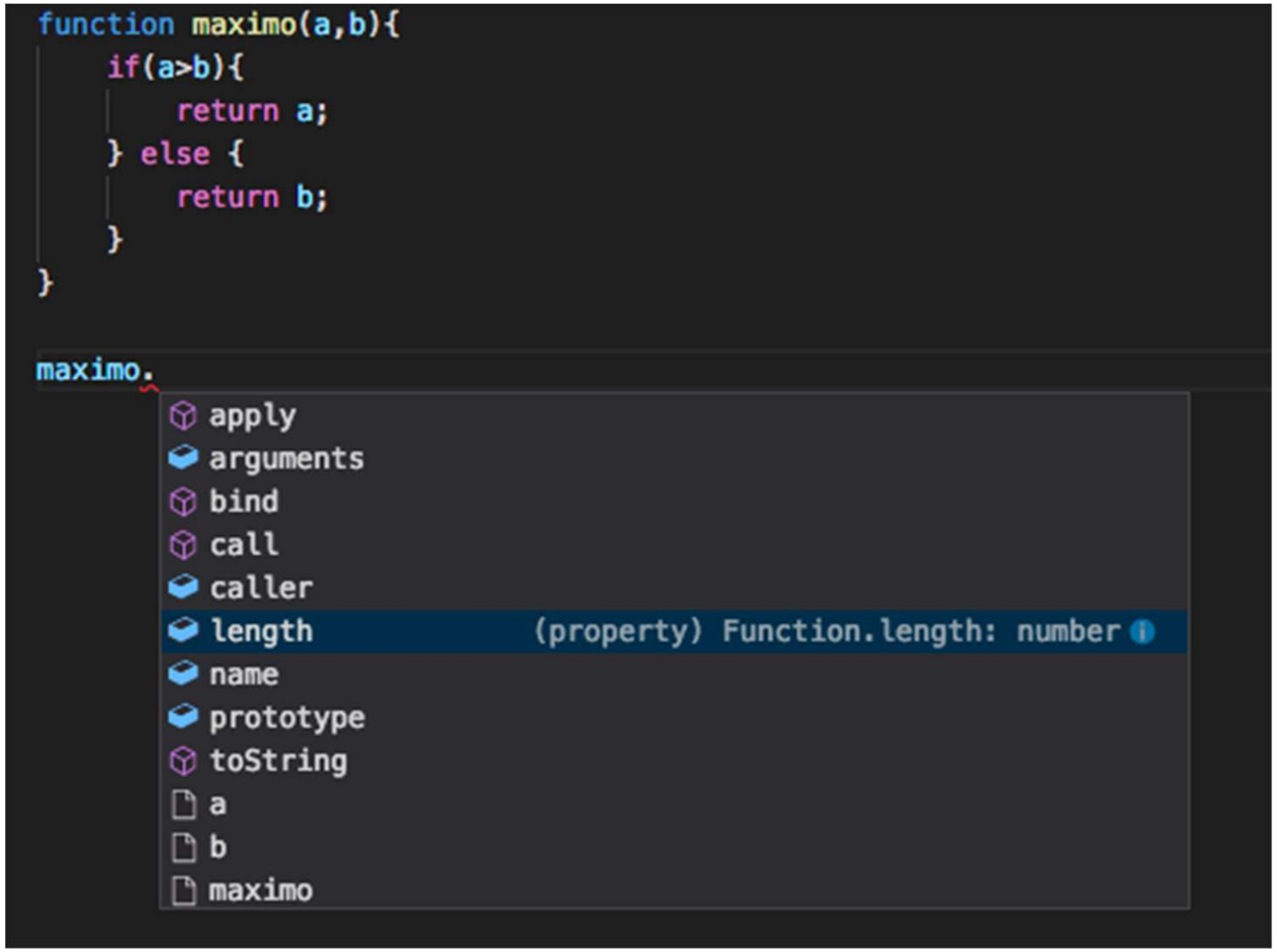
```
function maximo(...args){
        let maxValue = undefined;
        for(let value of args){
                if(!maxValue || value > maxValue){
                        maxValue = value;
                }
        }
        return maxValue;
}

console.log(maximo(5,1,89));
console.log(maximo(89,23,-102,0));
```

The function supports a different number of parameters.

Let’s do an experiment

```
console.log( typeof(maximo) );
```

Functions are objects (in JS)



```
function maximo(a,b){
    if(a>b){
        return a;
    } else {
        return b;
    }
}

maximo.
```

Functions are objects (in JS)

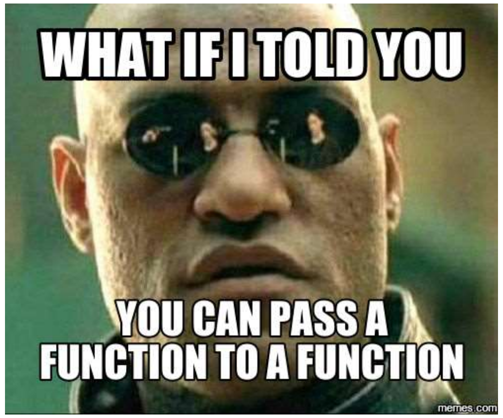
- If a function is an object…
  - Can I store it in a variable? → In fact, the name of the function is the name of the variable
  - Can I assign it to other variable? → Of course!!!
  - And can I declare a variable and assign it to a function object created in the moment→ Sure

Functions are objects (in JS)

```
let x = (param1, param2, param3) => {
    return param1 + param2 + param3;
};

console.log( x(1,2,3) );
```

Functions are objects (in JS)

- If a functions is an object…
  - Can I store it in a variable? → In fact, the name of the function is the name of the variable
  - Can I assign it to other variable? → Of course!!!
  - And can I declare a variable and assign it to a function object created in the moment→ Sure
  - Can I send/pass a function as a parameter to another function→ Why not?
  - Can I return a function from a function→ That's it



First-Class functions

- In JavaScript functions are treated like other variables.
- Functions can be assigned to any other variable or passed as an argument or can be returned by another function.
- Functions are simply a value and are just another type of object.

```
function speak(string) {
    console.log(string);
}

speak("Hello");        // logs "Hello"

var talk = speak;
talk("Hi");                // logs "Hi“

var myFuncs = [talk];
myFuncs[0](“Bye");    // logs “Bye"
```

Higher-order functions

- A function can be called higher-order function if meets one or both of these conditions:
  - Receives another function as an argument.
  - Returns a function as a result.
- Higher-order functions are only possible because of the First-class functions.

```
function applyFunction(fn, val) {
    return function() {
        fn(val);
    };
}

function speak(string) {
    console.log(string);
}

var sayHi = applyFunction (speak, "Hi");
sayHi();                    // logs "Hello";
```

Higher-order functions

```
function sortCriteriaA(a,b){
        if((a%10)<=(b%10)){
                return true;
        }
        return false;
}

function sortCriteriaB(a,b){
        return a<=b;
}
```

```
function sort(array,criteria){
  for(let i=0;i<array.length;i++){
    let minElem = array[i];
    let minIndex = i;
    for(let j=i;j<array.length;j++){
            if(criteria(array[j],minElem)){
              minElem = array[j];
              minIndex = j;
            }
    }
    let auxElem = array[i];
    array[i] = minElem;
    array[minIndex] = auxElem;
  }
}
```

```
let array = [47, 89, 11, 49, 80, 20];
sort(array,sortCriteriaA);
console.log(array);

array = [47, 89, 11, 49, 80, 20];
sort(array,sortCriteriaB);
console.log(array);
```

Higher-order functions

```
function funcGenerator(a){
        return (b) => {
                return a*b;
        };
}

const mult3 = funcGenerator(3);
console.log(mult3(5));
```

Popular higher-order functions in JS

- There are some high order functions in JS already provided by the language and very useful.
- For the arrays we have:
  - map
  - forEach
  - filter
  - some
  - every

Popular higher-order functions (map)

```
function power2(x) {
    return x ** 2;
}

let array = [1, 2, 3, 4, 5];
let newArray = array.map(power2);
document.write(newArray);
```

map returns a new array with the result of applying the function passed as parameter to each element of the original array

Popular higher-order functions (forEach)

```
function createText(item, index) {
    text += index + ": " + item + "<br>";
}

let text = "";
const teachers = ["Jose", "Lorenzo", "Mariluz"];
teachers.forEach(createText);
document.write(text);
```

forEach applies a function to each element of the array, but does not return a new array

Popular higher-order functions (filter)

```
function even(x) {
    return x % 2 == 0;
}

let array = [1, 2, 3, 4, 5];
let newArray = array.filter(even);
document.write(newArray);
```

filter returns a new array filled with all the elements of the original array that pass a test (provided by the passed function)

Popular higher-order functions (some)

```
function even(x) {
    return x % 2 == 0;
}

let array = [1, 2, 3, 4, 5];
document.write(array.some(even));
```

some returns true if at least one of the elements of the original array passes a test (provided by the passed function)

Popular higher-order functions (every)

```
function even(x) {
    return x % 2 == 0;
}

let array = [1, 2, 3, 4, 5];
document.write(array.every(even));
```

every returns true if ALL the elements of the original array pass a test (provided by the passed function)

Arrow functions

- Many of the functions we are using are very simple.
- But, there’s a more concise syntax for writing function expressions.
- Arrow functions allows a developer to accomplish the same result with fewer lines of code and approximately half the typing.

```
// ES5
var add = function (x, y) {
    return x + y;
};
// ES6
var add = (x, y) => { return x + y };
```

Arrow functions

- Many of the functions we are using are very simple.
- But, there's a more concise syntax for writing function expressions.
- Arrow functions allows a developer to accomplish the same result with fewer lines of code and approximately half the typing.

```
var add = (x, y) => { return x + y };
```

Function created in the same sentence.

Arrow functions

- Many of the functions we are using are very simple.
- But, there's a more concise syntax for writing function expressions.
- Arrow functions allows a developer to accomplish the same result with fewer lines of code and approximately half the typing.

```
var add = (x, y) => { return x + y };
```

Parameters of the function

Arrow functions

- Many of the functions we are using are very simple.
- But, there’s a more concise syntax for writing function expressions.
- Arrow functions allows a developer to accomplish the same result with fewer lines of code and approximately half the typing.

```
var add = (x, y) => { return x + y };
```

Output of the function

Arrow functions

- Arrow functions are very used in Angular.
- Syntax: (param1, param2, paramN) => { expression; }

```
var add = (x, y) => { return x + y };
console.log( add( 2, 4 ) );
```

Arrow functions

- Curly brackets aren't required if only one expression is present:

```
var add = (x, y) => x + y;
```

- If there's only one argument, then the parentheses are not required:

```
var squareNum = x => x * x;
```

- If there's no arguments, we must include parenthesis () or underscore _:

```
var hi = () => { return 'Hello World' };
console.info( hi() );
```

Objects

Data types (complex)

Type	Example(s)

Array	let arrayA = [1,2,3,4,5];
	let arrayB = new Array();
	let arrayC = [];

TypedArray	let x = new Int32Arrays(2);
	x[0] = 32;
	x[1] = -1;

Set	let mySet = new Set();
	x.add(4);

Map	let x = new Map();
	x.set(‘foo’,16);

Object	{}

JavaScript Sets

- A JavaScript Set is a collection of unique values, where each value can only occur once.
- A Set is an object and can hold any value of any data type.
- Arrays and Sets are very similar, but Arrays can have duplicate values and Sets cannot.
- Besides, data in an array is ordered by index whereas Sets use keys and the elements are iterable in the order of insertion.

```
const mySet = new Set([1, 2, 3, 3]); // Initializes a set with 3 items
console.log(mySet.values()); // logs 1,2,3
mySet.add(4); // Successfully adds the value
mySet.add(3); // Doesn't add the value. No error thrown!
letters.forEach((value) => {
    document.write(value + "<br>"); // Invokes a function for each Set element
})
mySet.delete(1); // Deletes the element 1
mySet.clear(); // Removes all the elements in the set
```

JavaScript Sets

- An interesting use of Sets…
- Since Sets cannot have duplicate values, we can use them to remove duplicates from an Array:

```
var initialArray = [1, 2, 1, 2, 3, 3, 4, 3, 4];
var set = new Set(initialArray);
const newArray = Array.from(set);
// Or just var newArray = [...new Set(initialArray)];
console.log(newArray); // logs 1,2,3,4
console.log(set.size); // logs 4
console.log(set.has(3)); // logs true since 3 is in the set
```

JavaScript Maps

- Maps holds key-value pairs where the keys can be any datatype (string, numbers, functions or objects).
- Maps remember the original insertion order of the keys.

```
const map = new Map([[1, 10], [2, 20]]); // map = {1=>10, 2=>20}
map.get(1); // returns 10
map.has(1); // returns true
map.set(3, 30); // {1=>10, 2=>20, 3=>30}
map.set(1, 15); // {1=>15, 2=>20, 3=>30}
let isDeleted = map.delete(1); // {2=>20, 3=>30}
console.log(isDeleted); // true
console.log(map.size); // 2
```

JavaScript Objects

- Objects are similar to arrays, except that instead of using indexes to access and modify their data, data in objects is accessed through what are called properties.
- Objects are useful for storing data in a structured way, and can represent real world objects.
- A Literal Object is a set of zero or more pairs *name: value*.
- Example: a cat object…

```
var cat = {
    "name": "Whiskers",
    "legs": 4,
    "tails": 1,
    "enemies": ["Water", "Dogs"]
};
```

JavaScript Maps vs Objects

- Wait… From the definition of Object, you can tell that Map and Objects sound very similar.
- In fact, Objects have been used as maps until Map was added to JavaScript.
- Both are based on the same concept: using key-value pairs to store data.
- How are Maps and Objects different?
  - Keys in an Object can only be simple types (String, Symbol), but in a Map they can be any data-type including Function and Object.
  - In an Object the original order of element isn't always preserved but in a Map it is.
- <u>Objects</u> are better for JSON transfers, and quick and small data exchanges
- <u>Maps</u> are better when working with big data loads

JavaScript Objects - Access

- There are two ways to access the properties of an object:
  - Dot notation (.): You use Dot notation when you know the name of the property you're trying to access ahead of time.

```
var myObj = { prop1: "val1", prop2: "val2" };
var prop1val = myObj.prop1; // val1
var prop2val = myObj.prop2; // val2
```

JavaScript Objects - Access

- There are two ways to access the properties of an object:
  - Bracket notation ([]): it’s similar to an array. You use it when the property of the object you are trying to access has a space in its name.

```
var myObj = {
    "Space Name": "Luke",
    "More Space": "Han",
    "NoSpace": "Millenium Falcon"
};
myObj["Space Name"]; // Luke
myObj["More Space"]; // Han
myObj["NoSpace"]; // Millenium Falcon
```

JavaScript Objects - Update

- After you've created a JavaScript object, you can update its properties at any time just like you would update any other variable.
- You can use either dot or bracket notation to update.

```
var cat = {
    "name": "Whiskers",
    "legs": 4,
    "tails": 1,
    "enemies": ["Water", "Dogs"]
};
cat.name = "Little Whiskers";
cat["name"] = "Little Whiskers";
console.log(cat.name);
```

Using Objects for Lookups

- Objects can be thought of as a key/value storage, like a dictionary.
- If you have tabular data, you can use an object to "lookup" values rather than a switch statement or an if/else chain.
- This is most useful when you know that your input data is limited to a certain range.

```
function phoneticLookup(val) {
    var result = "";
    var lookup = {
        "alpha": "Astoria",
        "bravo": "Boston",
        "charlie": "Chicago",
        "delta": "Denver",
        "echo": "Eagleton",
        "foxtrot": "Fresno"
    };
    result = lookup[val];
    return result;
}
```

JavaScript Objects – Check existence

- It could be useful to check if the property of a given object exists or not.
- We can use the .hasOwnProperty(propname) method of objects to determine if that object has the given property name.
- .hasOwnProperty() returns true or false if the property is found or not.

```
var cat = {
    "name": "Whiskers",
    "legs": 4,
    "tails": 1,
    "enemies": ["Water", "Dogs"]
};

cat.hasOwnProperty("legs");  // true
cat.hasOwnProperty("color"); // false
```

JavaScript Objects - Add Properties

- You can add new properties to existing JavaScript objects the same way you would modify them.

```
var cat = {
    "name": "Whiskers",
    "legs": 4,
    "tails": 1,
    "enemies": ["Water", "Dogs"]
};

cat.color = "gray";
cat["color"] = "gray";
```

JavaScript Objects - Delete Properties

- We can also delete properties from objects.

```
var cat = {
    "name": "Whiskers",
    "legs": 4,
    "tails": 1,
    "enemies": ["Water", "Dogs"]
};

delete cat.tails;
delete cat["tails"];
```

JavaScript Complex Objects

- You may want to store data in a flexible Data Structure.
- A JavaScript object is one way to handle flexible data.
- They allow combinations of strings, numbers, booleans, arrays, functions, and objects.
  - This array contains one object with various pieces of metadata about a film. It also has a nested “actors" array.
  - It’s possible to add more films to the top level array.

```
var myFilms = [
    {
      "title": "Titanic",
      "year": 1997,
      "director": "James Cameron",
      "country": "USA",
      "actors": [
        "Leonardo DiCaprio",
        "Kate Winslet",
        "Billy Zane"
      ],
      "oscars": true
    }
  ];
```

Nested Objects

- The sub-properties of objects can be accessed by chaining together the dot or bracket notation.

```javascript
var ourStorage = {
    "desk": {
        "drawer": "stapler"
    },
    "cabinet": {
        "top drawer": {
            "folder1": "a file",
            "folder2": "secrets"
        },
        "bottom drawer": "soda"
    }
};
ourStorage.cabinet["top drawer"].folder2;  // "secrets"
ourStorage.desk.drawer; // "stapler"
```

Nested Arrays

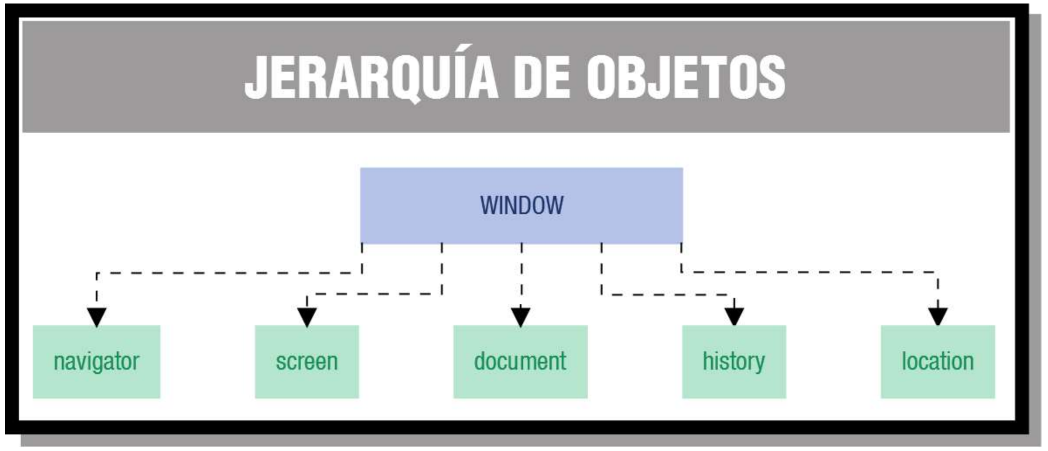
- Objects can contain both nested objects and nested arrays.
- Similar to accessing nested objects, Array bracket notation can be chained to access nested arrays.

```
var consoles = [
    {
        consoleName: "XBox",
        videogames: [
            "Forza Motorsport",
            "Halo",
            "Gears of War"
        ]
    },
    {
        consoleName: "PlayStation",
        videogames: [
            "God of War",
            "The Last of Us",
            "Uncharted"
        ]
    }
];
consoles[0].videogames[1]; // "Halo"
consoles[1].videogames[2]; // "Uncharted"
```

Surfing in the DOM

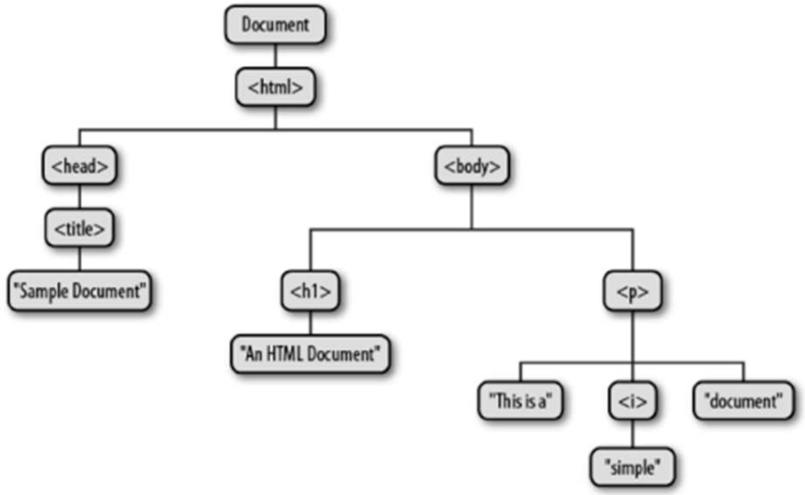
JavaScript High Level Objects

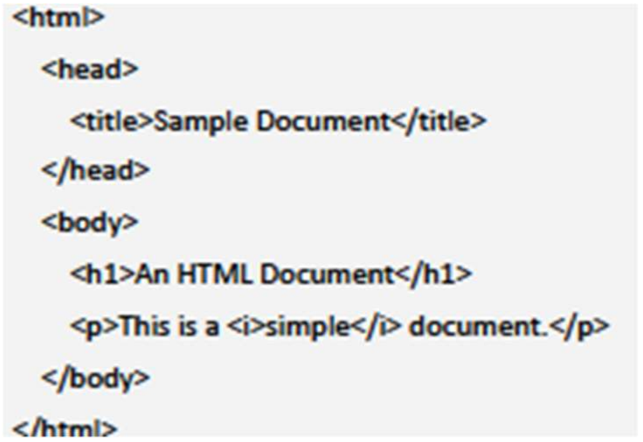
- Now that we've seen the basics of objects, let’s go a little deeper into them.
- We can place Objects in most of our documents.
- This graphic represents the JavaScript high level object model…



What is the HTML DOM?

- When a web page is loaded, the browser creates a Document Object Model of the page.
- The DOM represents a document as a hierarchical tree of Objects.
- The HTML DOM is a standard for how to get, change, add, or delete HTML elements to a webpage via JavaScript.





```
<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.</p>
  </body>
</html>
```

Window Object

- Represents an open window in a browser. For JavaScript, a browser tab is a window too.
- A Window is the main container for all the content that is displayed in the browser.
- It includes the dimensions of the window, scroll bars, toolbar, status bar, etc.
- It has properties and methods, but properties are barely used.

- All the properties and methods in W3Schools.
  - http://www.w3schools.com/jsref/obj_window.asp

Window Object

- We can access properties and methods by using the Dot notation (.):
  - window.propertyName
  - window.methodName (*[parameters]*)
- Other ways:
  - When we access a property from a document loaded on a Window:
    - self.propertyName
  - Since the Window object is always present, we can omit it when accessing objects of the same window:
    - propertyName

Window Object

- A script will never create the main window of a browser.
- It's the user who opens a URL in the browser or a file from the menu.
- Anyway, a script running in one of the main browser windows can create or open new sub-windows.
- The method *window.open()* generates a new window and contains up to three parameters:
  - the URL of the document to open
  - the name of the window
  - its physical appearance (size, color, etc.)

Window Object

```
var subWindow=window.open(“new.html",“new","height=800,width=600");
```

- We assign the new window to the subWindow variable.
- Doing this, we will be able to access the new window from the original script in the main window.

```
subWindow.close();
```

- We cannot use *window.close()*, *self.close()* or *close()*, since we would be trying to close the main window.

Location Object

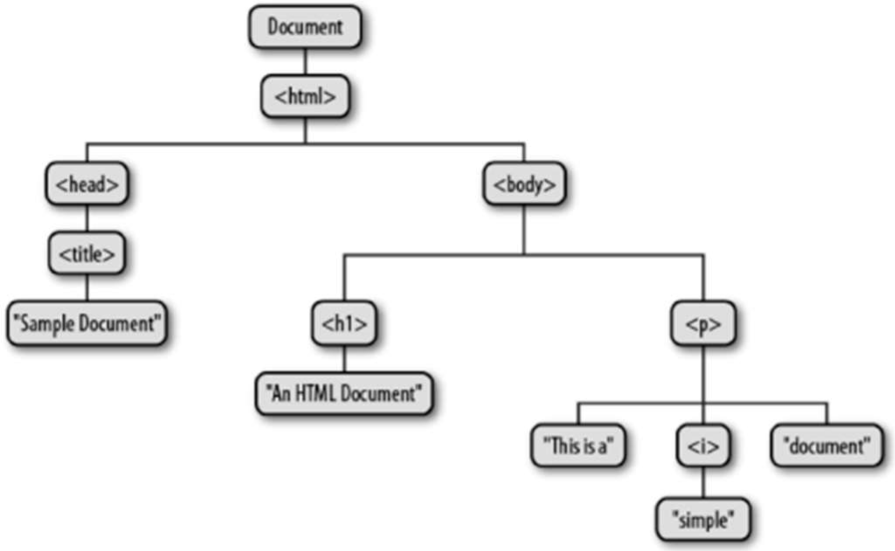
- The Location object contains information about the current URL.
- The Location object is part of the window object and is accessed through the window.location property.
- This object has some useful methods:
  - `location.reload()`: Reloads the current document
  - `location.replace()`: Replaces the current document with a new one
- All the properties and methods in W3Schools.
  - https://www.w3schools.com/jsref/obj_location.asp

Navigator Object

- The Navigator object contains information about the browser.
- This object has some useful properties:
  - `navigator.appName`: Returns the name of the browser
  - `navigator.geolocation`: Returns the user's global position
  - `navigator.language`: Returns the language of the browser
- All the properties and methods in W3Schools.
  - https://www.w3schools.com/jsref/obj_navigator.asp

Document Object

- It’s the highest object in the DOM (Document Objects Model) structure:



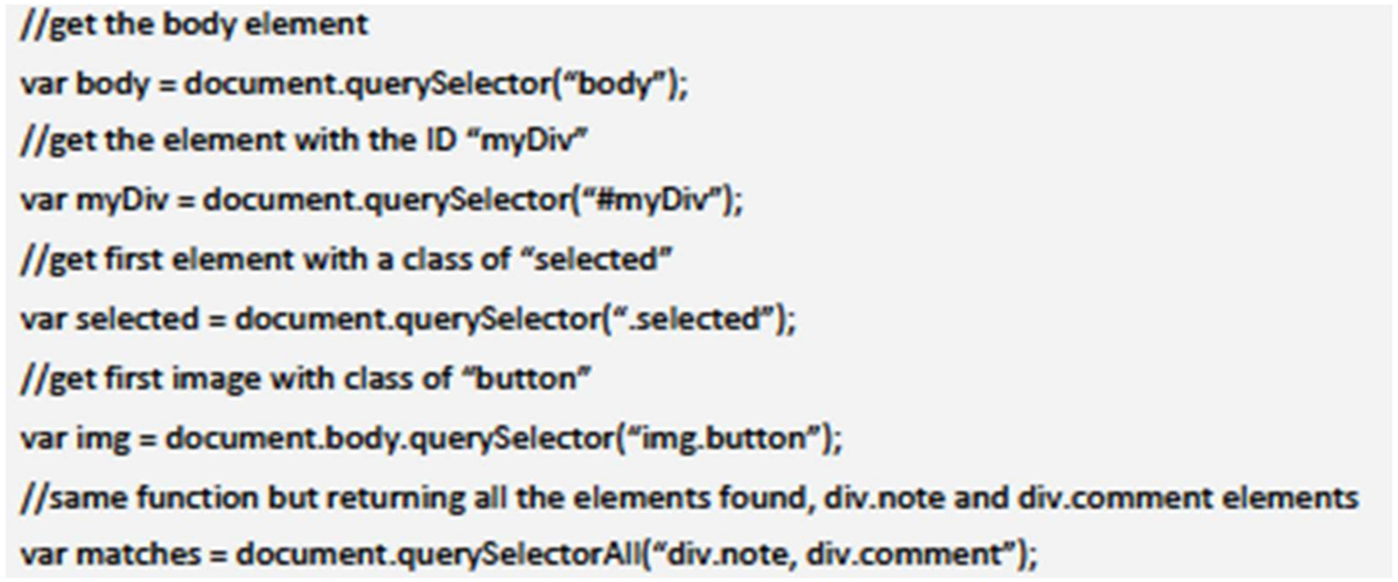
Document Object

- When an HTML document is loaded into a web browser, it becomes a Document object.
- It provides properties and methods to access all node objects from within JavaScript.
- The document object is the root node of the HTML document.
- It’s the object that lets you work with DOM.

Document Object

- Some very useful methods (document.xxxxxxxxxx):
  - getElementById("introduction") //returns the element with this id
  - getElementsByName("name") //returns an array with named elements
  - getElementsByTagName("li") //returns an array with all ***li*** elements
  - querySelector(".class") //returns the first .class element
  - querySelectorAll(".class") //returns an array with all .class elements
  - write() //writes text directly to the HTML document
  - createElement("button") //creates a button element

Selecting elements



```
//get the body element
var body = document.querySelector(“body”);
//get the element with the ID “myDiv”
var myDiv = document.querySelector(“#myDiv”);
//get first element with a class of “selected”
var selected = document.querySelector(“.selected”);
//get first image with class of “button”
var img = document.body.querySelector(“img.button”);
//same function but returning all the elements found, div.note and div.comment elements
var matches = document.querySelectorAll(“div.note, div.comment”);
```

- All the properties and methods in W3Schools.
  - http://www.w3schools.com/jsref/dom_obj_document.asp

Element Object

- Represents an HTML element like P, DIV, A, TABLE…
- Can have child nodes, siblings and father.
- It’s the object that lets you work with DOM.

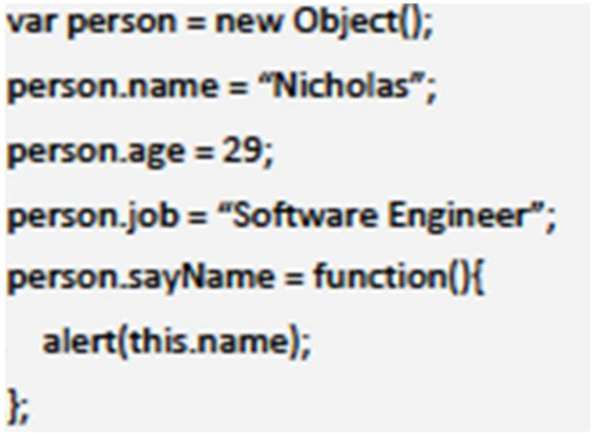
Element Object

- Some very useful methods and properties:
  - innerHTML //Access the HTML content of an element
  - innerText //Access the inner text of an element
  - click() //Simulates a mouse-click on an element
  - blur() //Removes focus from an element
  - appendChild() //Appends a node as the last child of a node
  - parentElement() //Returns the parent element of the specified element
  - remove() //Removes the specified element from the document

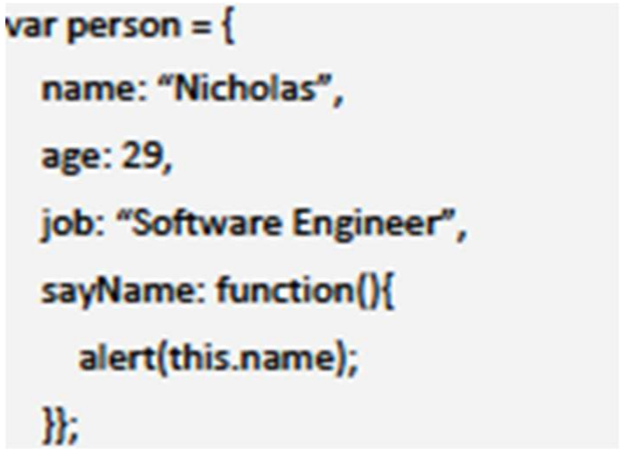
- All the properties and methods in W3Schools.
  - https://www.w3schools.com/jsref/dom_obj_all.asp

User-defined objects

- As we already know, we can create our own objects with properties and functions.



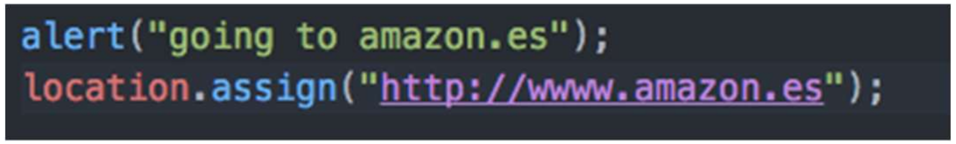
```
var person = new Object();
person.name = “Nicholas”;
person.age = 29;
person.job = “Software Engineer”;
person.sayName = function(){
   alert(this.name);
};
```



```
var person = {
   name: “Nicholas”,
   age: 29,
   job: “Software Engineer”,
   sayName: function(){
      alert(this.name);
   }};
```

More objects

- Location: Redirections



```
alert("going to amazon.es");
location.assign("http://wwww.amazon.es");
```

- Date: Managing dates

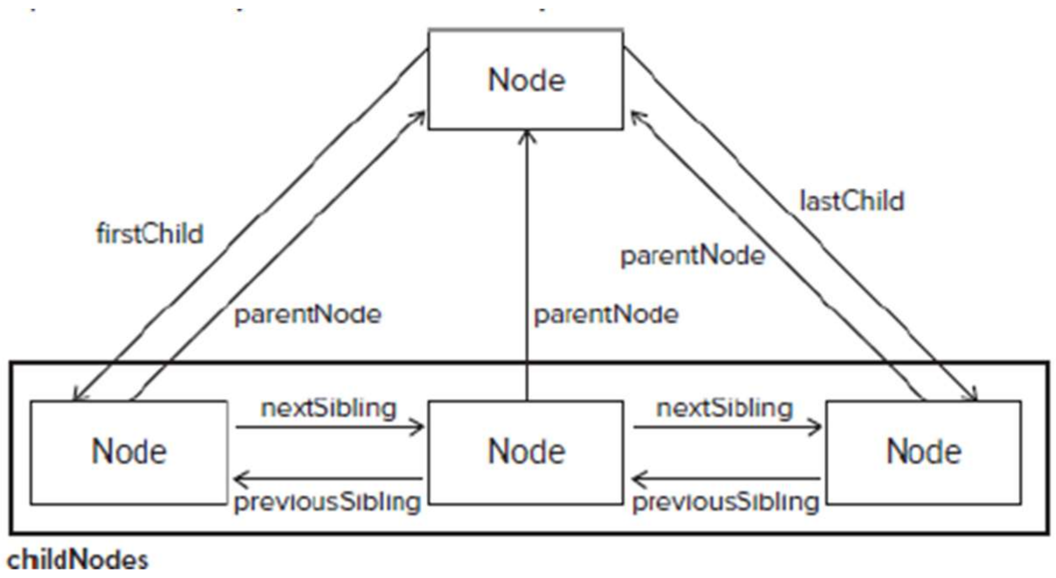
```
var today = new Date();              // creates Date object with current date
alert ( today.toDateString() );      // displays Fri Mar 09 2012 on 3/9/2012
alert ( today.getFullYear() );       // displays 2012
alert ( today.getDate() );           // displays 9
alert ( today.getMonth() );          // displays 2, not 3 for March
```

- And much more…

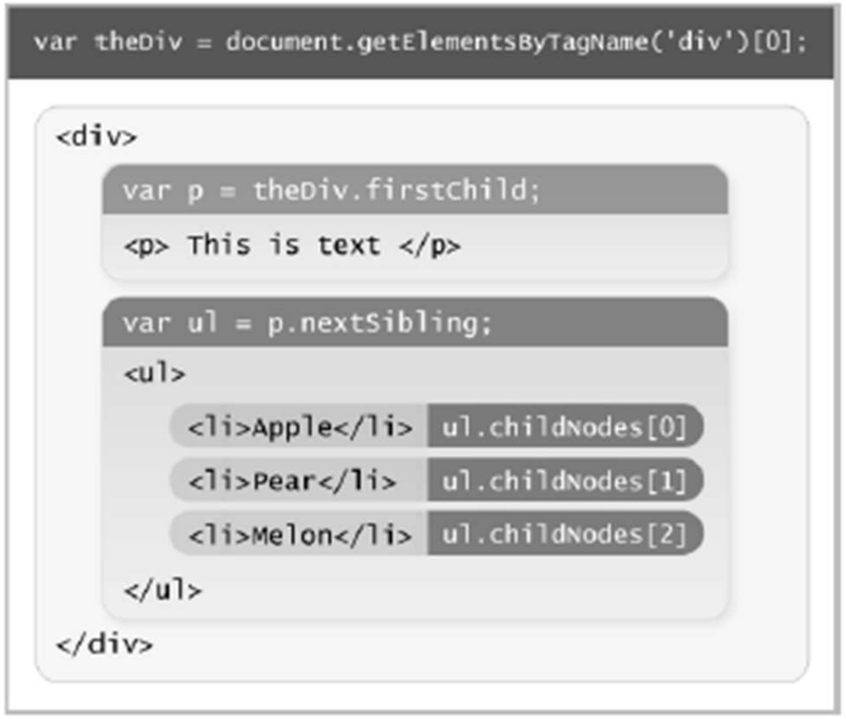
Nodes relationships

- All nodes in a document have relationships to other nodes.
  - Family like (father, sibling, children, etc.)
  - Each node has the following properties:
    - parentNode
    - childNodes
    - firstchild, lastchild
    - nextSibling, previousSibling
    - nodeType
    - nodeValue
    - nodeName

Nodes relationships



Nodes relationships



```
var theDiv = document.getElementsByTagName('div')[0];

<div>
    var p = theDiv.firstChild;
    <p> This is text </p>

    var ul = p.nextSibling;
    <ul>
        <li>Apple</li>   ul.childNodes[0]
        <li>Pear</li>    ul.childNodes[1]
        <li>Melon</li>   ul.childNodes[2]
    </ul>
</div>
```

Changing the dom

- Creating elements
  - A HTML link element:

```
<link rel=”stylesheet” type=”text/css” href=”styles.css”>
```

  - Using JS:

```
var link = document.createElement(“link”);
link.rel = “stylesheet”;
link.type = “text/css”;
link.href = “styles.css”;
var head = document.getElementsByTagName(“head”)[0];
head.appendChild(link);
```

Changing the dom

- Inserting and removing elements
  - As the last child → someNode.appendChild(someNode)
  - In a specific location → parentNode.insertBefore(node, positionNode);
  - Replacing node → someNode.replaceChild(newNode, oldNode);
  - Removing node → someNode.remove();
    parentNode.removeChild(childNode);

Managing events

Events

- JavaScript applications commonly respond to user actions like clicking on a button. These actions are called events, and the anonymous functions that handle the events are called event handlers.

- To make that happen, you have to attach the function to the events.

Events

- There are several categories of events that can occur in a web browser.
- Event groups:
  - User interface events: browser event.
  - Focus events: when gains or loses focus.
  - Mouse events: fired by the mouse
  - Wheel events: fired by the mouse wheel
  - Text event: fired when text is input into the document
  - Keyboard events: when the keyboard is used.
  - Etc…

Events

- Form Events: Events fired by actions inside a form (also applies to almost all HTML elements, but is most used in forms).
  - onfocus
  - onblur
  - oninput
  - onsubmit

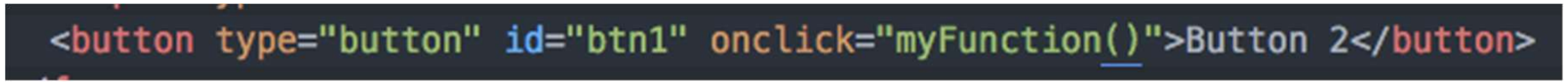
Events

- Keyboard Events
    - onkeydown
    - onkeypress
    - onkeyup

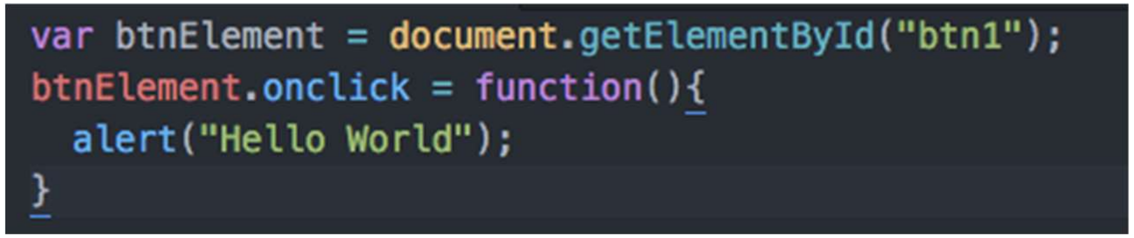
- Mouse Events
    - onclick
    - ondrag
    - ondrop
    - onmouseover

- http://www.w3schools.com/jsref/dom_obj_event.asp

Event handlers

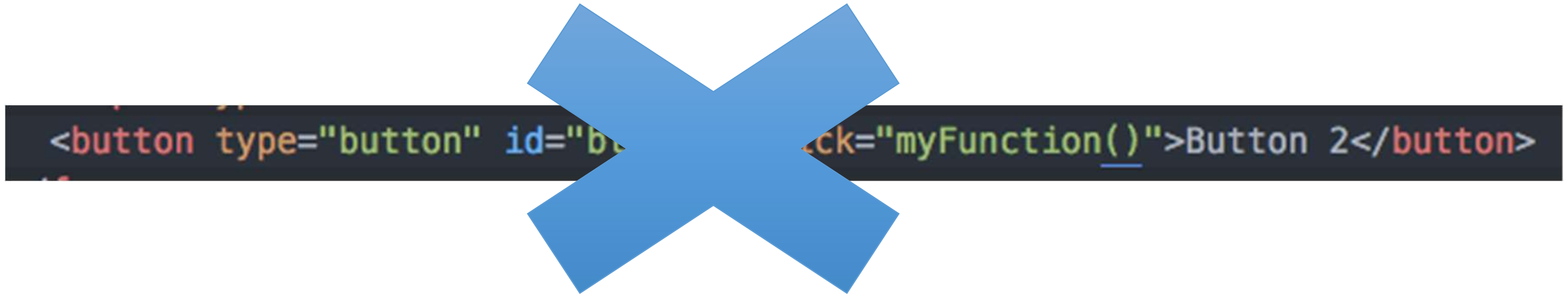


```
<button type="button" id="btn1" onclick="myFunction()">Button 2</button>
```

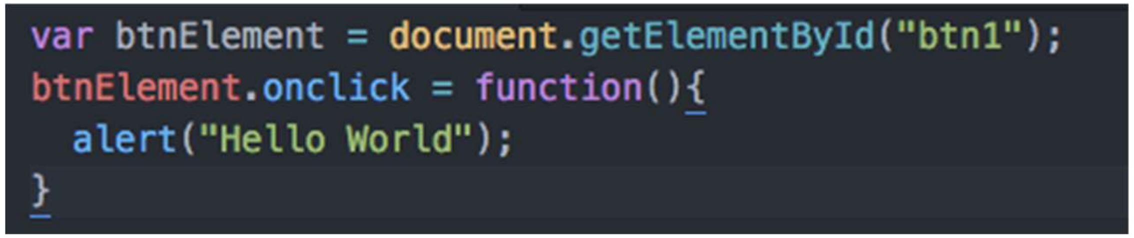


```
var btnElement = document.getElementById("btn1");
btnElement.onclick = function(){
  alert("Hello World");
}
```

Event handlers



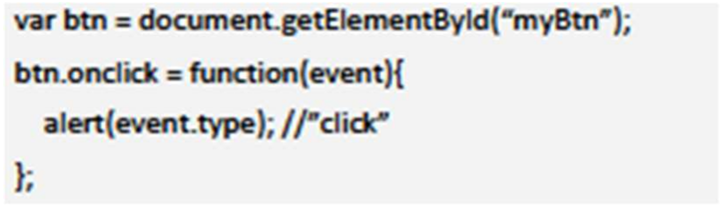
```
<button type="button" id="b         ck="myFunction()">Button 2</button>
```



```
var btnElement = document.getElementById("btn1");
btnElement.onclick = function(){
  alert("Hello World");
}
```

Events

- Event Handlers
  - Can be removed setting the property to null.
    - btnElement.onClick = null;
- There is an object called Event:



```
var btn = document.getElementById("myBtn");
btn.onclick = function(event){
  alert(event.type); //"click"
};
```

- You can get it retrieving it as an arg in your function.
- You can check the event attributes using the console.

Form Validation

- Data validation is the process of ensuring that user input is clean, correct, and useful.

- Typical validation tasks are:
  - has the user filled in all required fields?
  - has the user entered a valid date?
  - has the user entered text in a numeric field?

- Most often, the purpose of data validation is to ensure correct user input.

Form Validation - Example

- When we click the “Submit” button in a form, a JS function can be called to validate the input data:

HTML

```
<form name="loginForm" action="#" onsubmit="return valForm()" method="post">
  Username: <input type="text" name="uname" id="username">
  <input type="submit" value="Submit">
</form>
```

JS

```
function validateForm() {
  var x = document.forms["loginForm"]["uname"].value;
  // var x = document.forms.loginForm.uname.value;
  // var x = document.getElementById("username").value;
  if (x == "") {
    alert("You must enter a username");
    return false;
  }
```

Form Validation - Example

- Besides, we can access all elements in a form as a collection of elements:

HTML

```
<form name="loginForm" action="#" method="post">
  Username: <input type="text" name="uname">
  Password: <input type="password" name="pw">
  <input type="submit" value="Submit">
</form>
```

JS

```
document.getElementById("loginForm").elements.length; #number of elements in the form
document.getElementById("loginForm").elements[0].value; #value of the first element
document.getElementById("loginForm").elements.item(0).value; #value of the first element
document.getElementById("loginForm").elements.namedItem("uname").value; #value of the uname element
```

- This way, we can loop through all elements in a form!

Form Validation

- HTML5 introduced a new HTML validation concept called constraint validation.
  - Constraint validation HTML Input Attributes: max, min, required, type…
  - Constraint validation CSS Pseudo Selectors: :disabled, :optional, :valid…
  - Constraint validation DOM Properties and Methods: validity, checkValidity()…

https://www.w3schools.com/js/js_validation_api.asp

Form Validation – DOM Methods

- If an input field contains invalid data, display a message:

```
<input id="age" type="number" min="18" max="65" required>
<button onclick="checkAge()">OK</button>
<p id="val"></p>
<script>
 function checkAge() {
   var ageValue = document.getElementById("age");
   if (!ageValue.checkValidity()) {
     document.getElementById("val").innerHTML = ageValue.validationMessage;
   }
 }
</script>
```

Form Validation – DOM Properties

- If an input field contains invalid data, display a message:

```
<input id="num" type="number" max="100" min="0" >
<button onclick="check()">OK</button>
<p id="val"></p>
<script>
  function check() {
    var txt = "";
    if (document.getElementById("num").validity.rangeOverflow) {
      txt = "Value too high";
    } else if (document.getElementById("num").validity.rangeUnderflow) {
      txt = " Value too low";
    }
    document.getElementById("val").innerHTML = txt;
  }
</script>
```

Questions?