Javascript

And not only

What is JS

- “JavaScript is a programming language that adds interactivity to your website (for example: games, responses when buttons are pressed or data entered in forms, dynamic styling, animation)” - MDN

History lesson

- Developed by Brendan Eich (co-founder of the Mozilla project) at Netscape

  (initially Mocha or LiveScript)
- The language which implements a standard called ECMAScript.

  (European Computer Manufacturer’s Association)

Good to know

- What we’ll cover: syntax, bottlenecks, principles, web implementation, design patterns, libraries, ES6, TS, Angular, a lot
- Useful : MDN , W3schools, caniuse, StackOverflow

How to insert in HTML pages

- Inline

```
<script></script>
```

- External

```
<script src=“js/index.js”></script>
```

Data types

Loosely typed and a *Dynamic* language

- Does not bother with types too much, and does conversions automatically, i.e. you can easily add string to an integer and get the result as a string
- You don't have to declare the type of a variable ahead of time. The type will get determined automatically while the program is being processed. That also means that you can have the same variable as different types.
- Typeof, instanceof

Primitives

```
var s = ‘aaa’
var n = 10
var b = true
var n = null
var u = undefined

var sy = Symbol() - new in ES6
```

More info: MDN

Objects

- Key-value collections
- Keys are strings
- For numeric keys, js calls toString behind the scenes

```
var array = [1, 2, 3, 4]

var object = {
      foo: 1,
      bar: 2,
      1: 3
}
```

Functions

- a block of code designed to perform a particular task.

```
function name(parameter1, parameter2, parameter3) {

      // code to be executed
}
```

- Are regular objects with the additional capability of being callable.
- Can be self-called
- Have context
- Function methods*

More info: [MDN](#) [Expression vs declarations](#)

Built-in objects

Math

- No express definition for integers
- Math object - static methods/no constructor
- Random
- Floating point error (nicely explained)

More info: Math , Numbers

Strings

- Escaping (\’)
- Length, substring, indexOf
- More on ES6

More info: W3schools

Regular expressions

/[pattern]/[flags]

new RegExp(pattern[, flags])

/ab+c/i;

new RegExp('ab+c', 'i');

new RegExp(/ab+c/, 'i');

- reg.test(str) -> boolean, str.match(reg) -> groups

Full docs: MDN

Util: http://regexr.com

Date

```
var d = new Date("2015-03-25");
```

- Multiple formats
- No leading 0 on days/months formats may throw exception
- Some formats may return NaN

More info: W3schools

Array

- Push / pop, length
- Deleting, splicing
- Sorting
- Do not use arr[newIndex] = val; Do not use delete;
- Functional methods (map, filter, reduce ...)
- Functional* - is the process of building software by composing pure functions, avoiding shared state, mutable data, and side-effects

More info: MDN, W3Schools

Operators

Operators

- Arithmetic operators (+ - / * ++ -- %)
- Assignment operators (= += -= *= /= %=)
- String operators (+ +=)
- Comparison Operators (< > <= >= == != === !==)
- Logical operators (&& || !)
- Unary operators (typeof)
- Relational operators (in, instanceof)
- *Bitwise operators* (& | ~ ^ >> <<)

More info: MDN

Arithmetic operators

- JS tries to convert the operands to a type suitable for the specific operation
- For +/- a string operation is tried
- Interesting with different types

Comparison Operators

- ! and ? operator
- && and ||
- Falsey and Truthy
- == vs ===

More about falsey and truthy here

Mindblown: javascript equality table

Strict mode

Strict mode

- Restricted variant of JavaScript
- Browsers not supporting strict mode will run strict mode code differently

```
  // Global-level strict mode syntax
  'use strict';

function strict() {
  // Function-level strict mode syntax
  'use strict';
  function nested() { return 'And so am I!'; }
  return "Hi!  I'm a strict mode function!  " + nested();
}
```

Converting mistakes into errors

- Using a variable/object, without declaring it
- Deleting with *delete* keyword
- Duplicating a parameter name is not allowed
- The string ‘eval’/’arguments’ cannot be used as a variable (reserved words)
- Global function this value default is undefined

More info: W3Schools

Js and the Web

HTML interactions

- DOM
- DOM nodes
- Finding, Changing, Adding, Deleting
- Styling
- The Render Tree, Reflows

More info: W3Schools

DOM events

- HTML events
- Can be assigned from Js
- Click, hover, change, keyPress ...
- Bubbling and capturing
- stopPropagation and preventDefault

More info: W3Schools

Html DOM events here

Timing events

```
Var timeout = setTimeout(function, milliseconds);
clearTimeout(timeout);

var interval = setInterval(myTimer, 1000);
function myTimer() {
      var d = new Date();
      document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}

clearTimeout(interval);
```

More info: W3Schools

Context

Hoisting

- JavaScript's default behavior of moving all declarations to the top of the current scope
- Don’t rely on it
- JavaScript in strict mode does not allow variables to be used if they are not declared
- JavaScript only hoists declarations, not initializations
- Variable declared with *let* or *const* are not hoisted

More info: W3Schools

Scope

- The context in which values and expressions are "visible," or can be referenced
- Scope hierarchy
- Global/local scope
- Blocks

More info: W3schools

This

- Reference of the object that executes it
- Default is window / undefined (strict mode)
- Functions can be called with different *this* : W3schools

IIFE

- Basically self-calling
- Module design patterns

```
(function(){
      // your code goes here
      var visibleOnlyHere = ‘obviously’;
})()
```

More examples with custom object: W3Schools

Prototype

- Methods that are available on instances of that object
- To add functionality on built-in objects = bad idea

More examples with custom object: W3Schools

Closures

- Lexical environment
- Function factory

More info: MDN

Other browser interactions

- BOM
- Window
- Screen
- History
- Location
- Cookies

More info: W3Schools

Libraries

Useful libraries

- jQuery $
- Lodash _
- Bootstrap

- Angular
- AngularJs

JQuery

- Quick query selectors
- Utility functions
- Implementation for important use-cases (Ajax, animations, promises)

API documentation here

Some examples here

Promises

- The eventual result of an *asynchronous* operation
- A placeholder into which the successful result value or reason for failure will materialize.
- States (pending, fulfilled, rejected)
- Types of asynchronous data
- Avoid callback hell, more functional
- Implementations in many libraries

Nicely explained here

jQuery example here

Ajax

- Server communication without reloading the page
- *Asynchronous*
- Ajax call = promise
- REST

More info: W3Schools, MDN

Frontend project architecture

Build tools

- Development/production build tasks
- Dependency injection

Tools: Gulp, Webpack, Angular CLI

Dependency management: NPM, Yarn

NPM

- Package manager for JS
- Initially introduced for Node
- Modular
- Now used for front-end development

Transpilers

- SCSS/SASS, LESS
- Jade, HAML
- ES6 - Babel
- Typescript

Linting

- ESLint, TSLint
- Code styling
- Pre-defined set of rules

Minifying

- Useful info about minification
- ‘Security’ (not really)
- Optimization
- Use semicolons
- Libraries perks

ES6

What is ES6

- New version of ECMAScript(Javascript)
- Not supported in all browsers
- Optimizations
- New data types and built-in methods
- New functionality

Compile ES6 to ES5: Babel

New stuff

- Const / Let - block-scoped
- Arrow functions
- Classes
- Modules
- Template literals

Complete list here

New stuff

- Spread operator
- Destructuring Assignment
- Generators
- Map/ Set
- New built-in functions

Useful ones here

Typescript

Typing

- Strongly typed
- TS is able to infer the type
- number, string, boolean, any, void, enum
- Generics

Classes

- You can have private members
- Constructor
- Static methods
- JS has classes too

Interfaces

- Allow us to create contracts other classes/objects have to implement
- We can use them to define custom types without creating classes
- ARE NOT compiled to JavaScript! It's just for checking/validation done by our TypeScript compiler.

Modules

- TS is modular, we can divide our code up over several files
- We export a class, interface, variable, ... by adding 'export' keyword in front of it
- We then use "import {} from "" to access the code in these files

Thank you!

catalin.matei@esolutions.ro