JavaScript Course

eSolutions Internship 2021

About me

Catalin Daniel Matei

- Bachelor’s degree in Computer Science @ University of Bucharest
- Frontend engineer @ Picnic Technologies in Amsterdam
- 7 years experience with web technologies

/catalin-matei /picnictechnologies
matei.catalin25@gmail.com

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

PIC NiC

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)

Use in HTML

- Inline

```
<script> //JS code </script>
```

- External

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

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
integrity=“sha384-XXXX" crossorigin="anonymous"></script>
```

Good to know

Useful materials

- MDN
- W3schools
- caniuse
- ui.dev
- CSS-Tricks
- The TypeScript Handbook
- StackOverflow

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Agenda by days

Time	Topic
Day 1	A JavaScript syntax
Day 2 1/2	B JavaScript syntax
Day 2 2/2	C JavaScript and the web
Day 3 1/2	D Frontend project architecture
Day 3 2/2	E TypeScript

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Data types & built-in objects

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 be determined automatically while the program is being processed. That also means that you can have the same variable with different types

More info: MDN

PIC NIC

Data types & built-in objects

Primitives

- All primitives are immutable - when you change the value, you change the reference
- null vs undefined

```
const s = 'aaa';
const n = 10;
const b = true;
const n = null;
const u = undefined;
const sy = Symbol();
```

Data types & built-in objects

Objects

- Key-value collection
- Keys are strings - for numeric keys, JS calls *toString()* behind the scenes
- Iterable using *for…in* / *Object.keys()* / *Object.values()* / *Object.entries()*
- Mutable and compared by address, not value

```
const object = {
  foo: 1,
  bar: 2,
  1: 3,
};
const array = [1, 2, 3, 4];
```

Data types & built-in objects

Arrays

- JS Objects with numbered keys
- Have special property length
- Deleting, splicing
- Sorting
- Different ways of iterating through arrays

More info: MDN | W3Schools

Data types & built-in objects

Array iterations

- *For* loop / *for…of*
- Functional methods (*map*, *filter*, *reduce* etc.)
- Functional programming - the process of building software by composing pure functions, avoiding shared state, mutable data, and side-effects

More info: W3Schools

Data types & built-in objects

Functions

- A block of code designed to perform a particular task
- Regular objects with the additional capability of being callable
- Have context
- Arrow functions
- Function expression vs declaration

```
function func(param1, param2, param3) {
  // code to be executed
}
const func = function (param) {
  // code to be executed
};
const arrowFunc = (param) => {
  // code to be executed
};
```

More info: MDN

Data types & built-in objects

Strings

- Length, substring, indexOf
- Single / double quotes
- Escaping
- Template literals

More info: W3Schools

Data types & built-in objects

Regular expressions

- /[pattern]/[flags] format
- Can be instantiated using *RegExp* class
- Regular expression check methods available both on *RegExp* and string instances
- Util: http://regexr.com

```
const regExp1 = /ab+c/i;
const regExp2 = new RegExp('ab+c', 'i');
const regExp3 = new RegExp(/ab+c/, 'i');
```

More info: MDN

Data types & built-in objects

Math

- No express definition for integers
- *Math* object - static methods, no constructor
- Random function
- *BigInt* in ES2020
- Floating point error

More info: Math | Numbers

Data types & built-in objects

Date

- Multiple formats
- No leading 0 on days/months formats may throw exception
- Some formats may return *NaN*
- Specified as the number of milliseconds that have elapsed since midnight on January 1, 1970, UTC

```
const date = new Date('2021-07-19');
const [month, day, year] = [date.getMonth(), date.getDate(), date.getFullYear()];

const now = new Date();
const nowMillis = Date.now();
```

More info: MDN | W3Schools

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Operators

Types of operators

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

More info: MDN

Operators

Arithmetic operators

- JS tries to convert the operands to a type suitable for the specific operation
- For +/- a string operation is tried
- For some numeric specific operations (*/) the operators will return *NaN*
- *Interesting* results when using operators with different types

Operators

Comparison operators

- ! and ? operator
- && and ||
- Abstract Equality Comparison (==) vs Strict Equality Comparison (===) - always use ===
- JavaScript equality table

More info: MDN

Operators

Truthy and Falsy

- When evaluated on conditionals they are *true* / *false*
- JavaScript's internal *toBoolean* function returns *true* / *false*
- *toBoolean* operator is called for: ! ? if && ||
- All falsy values in JavaScript

More info: Truthy | Falsy

Operators

New operators - ES2020

- Nullish Coalescing - *null/undefined ?? default*
- Optional chaining - *obj?.prop*

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Context

Scope

- The context in which values and expressions are *visible* or can be referenced
- Javascript is compiled multiple times. Once to initialise the scope (declaration of data) and then to analyse the context
- Scope hierarchy
- Global vs local scope
- Block scope - *var* vs *const*/*let*

More info: W3Schools

Context

Hoisting

- JavaScript's default behaviour of moving all declarations to the top of the current scope
- Don’t rely on it - the code should make sense without assuming the hoisting
- JavaScript in strict mode does not allow variables to be used if they are not declared
- JavaScript only hoists declarations, not initialisations
- Variable declared with *let* or *const* are not hoisted

More info: W3Schools

Context

This

- Reference of the object that executes the current expression (function)
- Default is *window* / *undefined* (strict mode)
- Functions can be used with different *this* - call | bind

Context

IIFE

- Immediately Invoked Function Expression
- Self-calling functions
- Useful for modularisation and encapsulation

```
(function(){
  // your code goes here
  var visibleOnlyHere = 'hidden';
})()
```

More info: MDN

Context

Prototype

- Blueprint of the instantiated type
- Methods that are available on instances of that object
- *instanceof* operator
- Add functionality on built-in objects = bad idea

More info: W3Schools

Context

Classes & prototype chaining

- In ES6 we have the syntax to define a class
- Syntactic sugar over prototyping in ES5
- Inheritance of prototype properties
- Nice article here

Context

Composition over inheritance

- Objects as sets of features
- Using *closures* - lexical environment, *function* factory
- More powerful than inheritance in JS
- Nice article on it on ui.dev

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Other JS concepts

Timing events

- Execution of code at specified time intervals
- Timers instances can be cleared

```
const timeout = setTimeout(() => console.log('Executed afte 2s'), 2000);
clearTimeout(timeout);

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

More info: W3Schools

Other JS concepts

Strict mode

- Restricted variant of JavaScript
- Converting mistakes into errors
  - Using a variable/object without declaring it
  - Deleting with *delete* keyword
  - Duplicating a parameter name
  - The strings *eval* / *arguments* cannot be used variable names (reserved words)
- Global *this* defaults to undefined
- Global level or function level with *'use strict'*
- Browsers not supporting strict mode will run strict mode code differently

More info: W3Schools

Other JS concepts

Older JS vs ES6+

- *var* -> *let* / *const* - block scoped
- Arrow functions -> no *this*
- Spread operator (…)
- *Map*, *Set*
- More built-in functions

PIC NiC

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

JS and the web

HTML interactions

- DOM & DOM nodes
- Finding, Changing, Adding, Deleting
- Styling
- The Render Tree - low-level representation of what will eventually get printed on the screen
- Reflows - recomputes the dimensions and position of the element, and it also triggers further reflows on that element's children, ancestors and elements that appear after it in the DOM
- How the browser renders a web page

More info: W3Schools

JS and the web

DOM Events

- HTML DOM events
- Can be assigned programatically from JS
- Click, hover, change, keyPress etc.
- Bubbling and capturing
- stopPropagation and preventDefault

More info: W3Schools

PRACTICE

JS and the web

Client-side storage

- Cookies - old school
- Web storage (*sessionStorage* / *localStorage*) - new school
- IndexedDB

More info: MDN

JS and the web

Other browser interactions

- BOM
- Window
- Screen
- History

More info: W3Schools

PRACTICE

JS and the web

Progressive Web Apps (PWA)

- Security and performance
- Notifications
- Service workers
- Offline capabilities
- Client storage

More info: MDN

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Server interactions

REST

- Architectural style that defines a form of communication between a client and a server
- The operation you want the server to perform on a resource in the form of a HTTP method (GET / POST / PUT / DELETE / PATCH / etc.)
- Has it's own 6 guiding constraints

More info: MDN | restfulapi.net

Server interactions

Promisses

- The eventual result of an asynchronous operation
- A placeholder into which the successful result value or reason for failure will materialise
- States: pending | fulfilled | rejected
- Types of asynchronous data: timing event, event listeners, Ajax calls
- Avoid callback hell
- Implementations in many libraries + native support in newer versions of JS
- Graceful asynchronous programming with Promises

More info: MDN

Server interactions

Ajax

- Newer way to interact with a server and updating resources
- Asynchronous JavaScript and ~~XML~~ (JSON)
- Server communication without reloading the page
- Ajax call = promise
- Can be used to consume REST APIs
- Different implementations using old and new technologies

More info: MDN | W3Schools

Server interactions

XMLHttpRequest & Fetch API

- Objects used to interact with the servers
- Fetch API is the more modern implementation

```
var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET", "http://www.example.org/
example.txt");
oReq.send();
```

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

More info: XMLHttpRequest | Fetch API

Server interactions

async and await

- Syntactic sugar over asynchronous programming in JS

```
async function myFetch() {
  let response = await fetch('coffee.jpg');
  if (!response.ok) {
    throw new Error(`HTTP error! status: $
{response.status}`);
  }
  return await response.blob();
}
```

```
myFetch().then((blob) => {
  let objectURL = URL.createObjectURL(blob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
}).catch(e => console.log(e));
```

More info: MDN

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

Frontend project architecture

Popular libraries

- React
- Angular
- Vue
- Others

- jQuery $
- Lodash _
- Bootstrap

Frontend project architecture

JQuery

- Quick DOM query selectors
- Utility functions
- Implementation for important use-cases (Ajax, animations, promises)
- You might not need jQuery

More info: API documentation

Frontend project architecture

Node.js

- JavaScript runtime built on Chrome's V8 JavaScript engine
- Install using NVM
- Local development environment for FE projects
- Enables JavaScript for the backend

More info: Node.js

Frontend project architecture

npm

- Package manager for JS
- Default package manager for Node.js
- Command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry
- Modular
- Now used for front-end development

More info: npm

Frontend project architecture

Build tools

- Execute workflows for you code
- Define development / production build tasks
- Run by the package manager: npm / yarn
- packaje.json scripts, Webpack, React scripts, Angular CLI

More info: npm

Frontend project architecture

Transpilers

- Source to source compilers
- TypeScript
- LESS, SASS (SCSS)
- Jade, HAML
- Babel (ES6+ -> ES5)

Frontend project architecture

Code style and formatting

- Opinionated vs unopinionated
- Code linting: ESLint / TSLint (deprecated) - defined set of code style rules
- Code formatting: Prettier (opinionated)
- EditorConfig - helping the editor produce code in line with the style guide
- EditorConfig vs. ESLint vs. Prettier

PIC NiC

Frontend project architecture

Other build tasks

- Code minification
- Ensure browser support: browserlist, caniuse
- Polyfills
- Module optimisation (tree shaking)
- Running tests

SETUP

Agenda

- Data types & built-in objects
- Operators
- Context
- Other JS concepts
- JS and the web
- Server interaction
- Frontend project architecture
- TypeScript

PIC NiC

TypeScript

Typing

- Strongly typed
- Vey popular language with great integration with the biggest frontend frameworks
- Enables
  - Better code style with less unexpected behaviour
  - Better code completion (Intellisense) by IDEs
  - Better build optimisation by build tools
- TS is able to infer the type on initialisation
- Compiled into vanilla JS code. Some concepts are just for type checking and will be translated in 0 lines of JS
- TS is modular, we can divide our code up over several files and export/import multiple things from them

More info: TypeScript | TS Handbook

PIC NIC

TypeScript

Interfaces

- Most commonly used to declare DTOs
- Allows 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 the TypeScript compiler

More info: Object types

PIC NIC

TypeScript

Classes

- Enhancement over normal JS classes
- Can define static methods
- Can have private/protected/pubic modifiers
- Cat define properties on a class using the constructor

More info: Classes

TypeScript

Enums

- TS specific concept - they are translated to JS classes
- Can be used a type and as a value
- Different types for enum values

More info: Enums

TypeScript

Common types

- Template literals
- Union types
- Tuple types
- Generics
- *keyof* type operator
- *typeof* type operator
- *any* / *unknown* / *never*

PRACTICE

Thank you 🎉