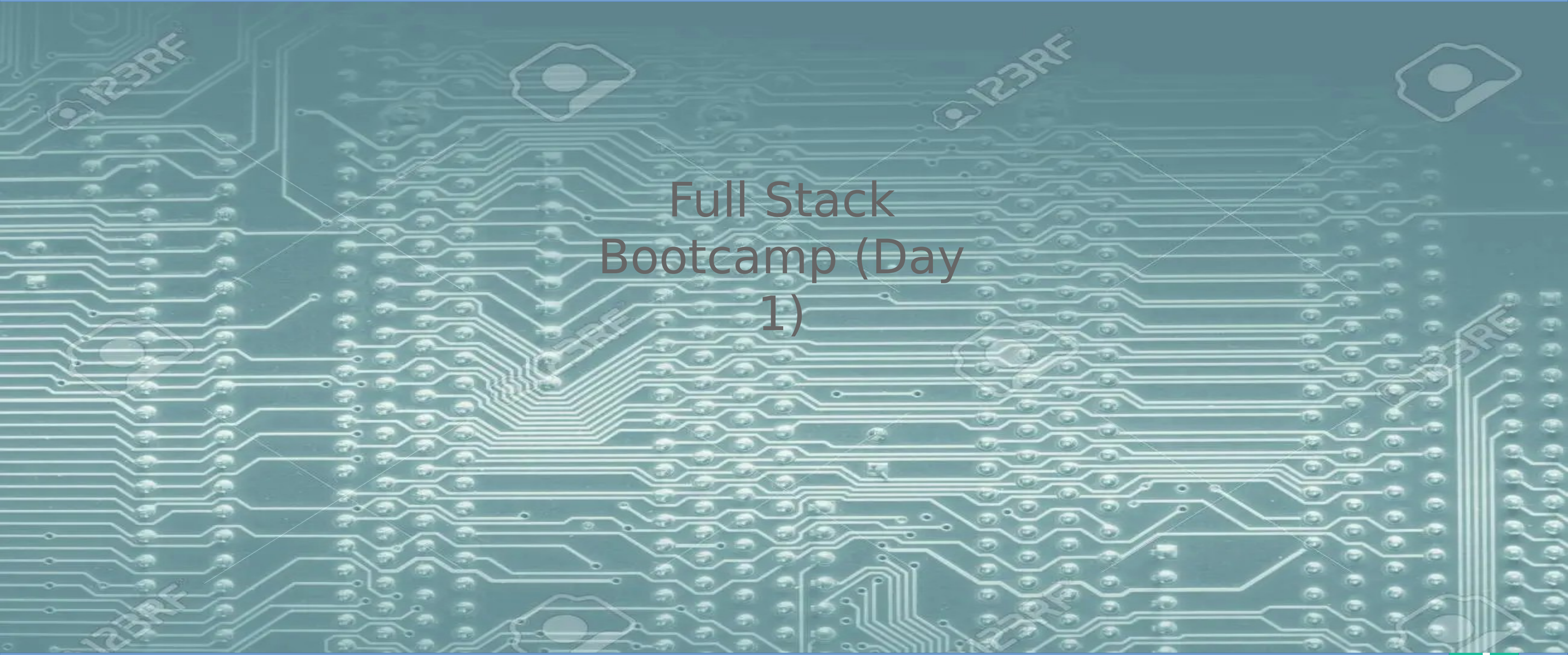
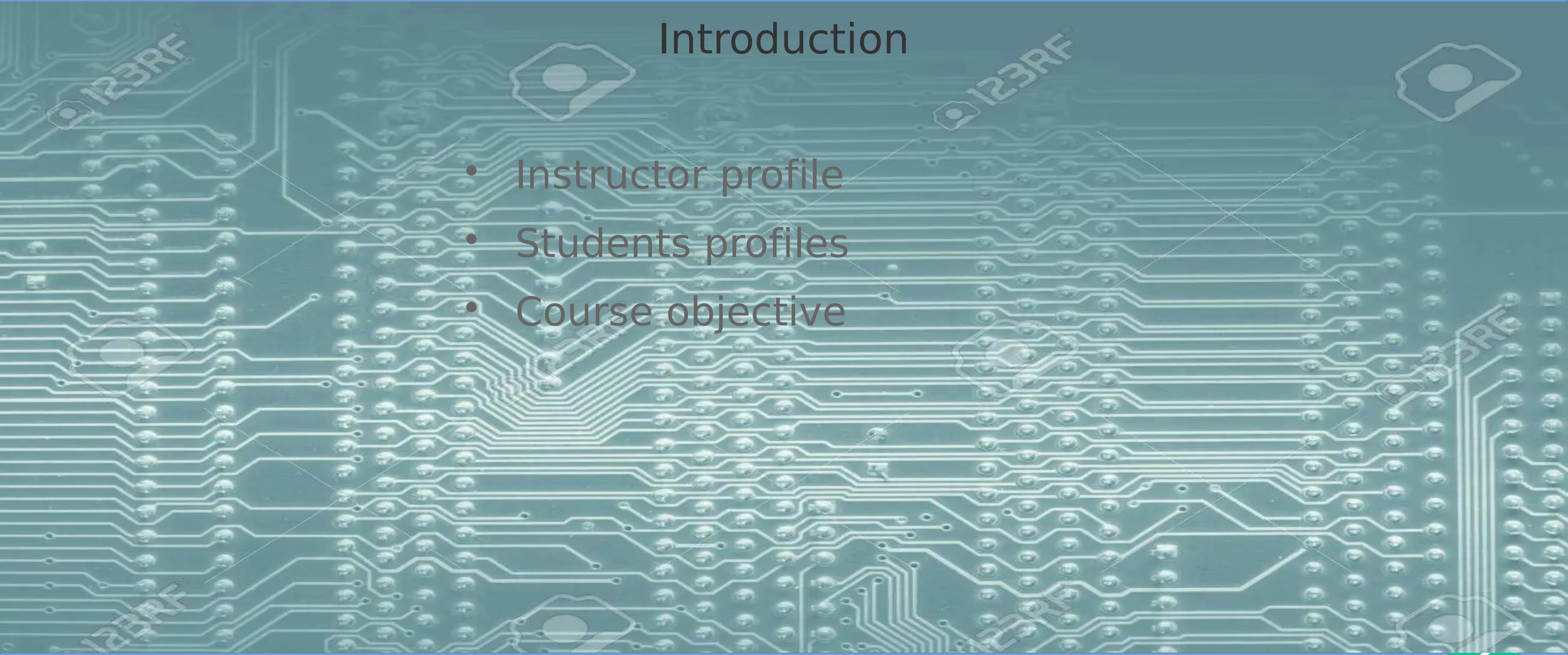
ES6+ Fundamentals



Full Stack Bootcamp (Day 1)

Introduction

- Instructor profile
- Students profiles
- Course objective



Agenda

- Day 1
  - ES6+
- Day 2
  - React Native
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

What’s ES6+?

- ESC6+ is also known as javaScript or Ecma Script.
- JavaScript was created in May 1995 by Brendan Eich while at Netscape, reportedly in only 10 days.
- Between 1996 and 1997 Netscape used the ecma standars for ECMA SCRIPT 1 https://www.ecma-international.org/
- Ecma 5 included JSON support
- ECMA 6was released on 2015
- ES6+ references to newer versions after ECMA Script6
- Most recent version of the standard isEMCA Script 2021(ECMA 262 Standard) as in January 2021

Language basic

- Let’s start looking the sintax of ES6+
- Every line ends with a ;
- Every block of properties of code is delimited by {}
- Strict mode is to make the variable definition required.
- When in HTML is inside <Script> tag

Language basic

- Exercise 0!!!!
- This exercise is to let you create a new

New ES6 syntax

- let vs var
  - let creates a variable that is blocked scope
  - var adds the variable to the global object list

```
let x = 10;
if (x == 10) {
    let x = 20;
    console.log(x); // 20:  reference x inside the block
}
console.log(x); // 10: reference at the begining of the
script
```

```
for (var i = 0; i < 5; i++) {
    setTimeout(function () {
        console.log(i);
    }, 1000);
}
```

let

```
for (let i = 0; i < 5; i++) {
    setTimeout(function () {
        console.log(i);
    }, 1000);
}
```

- The use of let in this case will create the right result, in order to use all ES6 style you must use the => function

const

JavaScript provides this keyword in order to define constants,variables created by the const keyword are “immutable”. In other words, you can’t reassign them to different values.:

```
const CONSTANT_NAME = value;
const young = { age: 20 };
```

const

```
const vehicules = ['car'];
vehicules.push('train');
console.log(vehicules); // ["car", "train"]

vehicules.pop();
vehicules.pop();
console.log(vehicules); // []

vehicules = []; // TypeError
```

Arrow functions

- Arrow functions
  - An arrow function is an alternative to the function traditional expression, this kind of function is limitated and can’t be used in all situations.
  - Differences and limits:
    - Can’t use **this** or **super** and can’t be use as method.
    - Does not have arguments or **new.target** keyword.
    - It can’t be used for **call**, **apply** or **bind** methods, that usually are used to stabliched an scope or ambit.
    - Can’t be used as constructor.
    - Can’t use **yield** inside of their body.

Default function parameters

Default parameters values can be defined as follow

```
function salute(message='Hi') {
    console.log(message);
}
salute(); // 'Hi'
salute('Hello') // 'Hello'
```

Rest parameters

Rest parameters are called for the way to create a functon with an undetermined number of parameters

```
function fn(a,b,...args) {
   //...
}
```

The args array stores the values after the second parameter

Spread operator

Spread operator are the counterpart of the rest operator (rest parameters) and it’s also represented by three points (…) while the rest parameters “packs” the contents of the refered list into an arry when we use the spread operator we use the spread operator to “unpack” the content of an array.

```
const odd = [1,3,5];
const combined = [...odd, 2,4,6];
console.log(combined);
```

Object literal extentions

Object literal is really a pattern for creating objects in JavaScript and the syntax is very simple to use:

```
function createMachine(name, status) {
    return {
        name: name,
        status: status
    };
}
```

for … of

When an object has an iterator (like the arrays) the for… of loop provides a way to go for every element of the iterator.

```
let scores = [80, 90, 70];
for (let score of scores) {
    score = score + 5;
    console.log(score);
}
```

Exercise 1

```
Go to the exercises and resolve the
exercise 1.
```

Octal and Binary Literals in ES6

You can use the 0o prefix or the 0b in order to stablished binary or ocatal values for a variable

```
let c = 0o51;
console.log(c); // 41
```

Destructing

```
The idea on destructing on arrays is to use only the
needed values for the caller
```

```
function getScores() {
   return [70, 80, 90, 100];
}
let [x, y, z] = getScores();
console.log(x); // 70
console.log(y); // 80
console.log(z); // 90
```

ES6 Modules

Modules are a way to organize the usage of functions and variables. In this case the HTML uses the app.js file and when this module is loaded then the greetings.js.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>ES6 Modules</title>
</head>
<body>
<script type="module" src="./app.js"></script>
</body>
</html>
```

```
// greeting.js
export let message = 'Hi';

export function setMessage(msg) {
  message = msg;
}
```

```
// app.js
import {message, setMessage } from
'./greeting.js';
console.log(message); // 'Hi'
setMessage('Hello');
console.log(message); // 'Hello'
```

Exercise 2

To test our skills to integrate HTML and ES6 let’s do exercise 2.

Classes

Unlike other programming languages such as Java and C#, JavaScript classes are just special functions that use the prototypal inheritance.

```
function Person(name) {
    this.name = name;
}
Person.prototype.getName = function () {
    return this.name;
};
var john = new Person("John Doe");
console.log(john.getName());
```

Classes

Starting on ES6 there are other way to stablish a class.

```
class Person {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}
```

Classes

In order to create subclasses you can use the “extends” keyword.

```
class ScalableCircle extends Circle {
    get radius() {
        return this.scalingFactor * super.radius;
    }
    set radius() {
        throw new Error("ScalableCircle radius is constant." +
                        "Set scaling factor instead.");
    }

    // Code to handle scalingFactor
}
```

Symbols

Symbol is a primitive tyoe that will obtain values that can’t be changed, that is they are immutables.

```
"use strict";

var arr = [1, 2, 3];
var it = arr[Symbol.iterator]();
console.log(it.next()); // {value: 'a', done:
false}
```

Symbols (cont...)

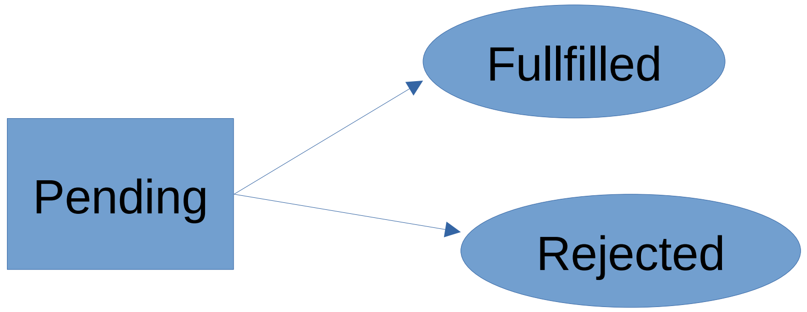
Sometimes sybols are used to simulate object properties encapsulation since the properties that are declared using the Symbol object can’t be accesed outside the object

```
"use strict";

const foo = Symbol();
const myObject = {};
myObject[foo] = 'foo';
myObject['bar'] = 'bar';
console.log(myObject); // {bar: "bar", Symbol(): "foo"}
console.log(foo in myObject); // true
console.log(myObject[foo]); // 'foo'
console.log(Object.getOwnPropertyNames(myObject)); // ['bar']
console.log(Object.keys(myObject));
```

Promises

A promise is an object that encapsulates the result of an asynchronous operation.
A promise object has a state that can be one of the following:

- ✓ Pending
- ✓ Fulfilled with a value
- ✓ Rejected for a reason



Promises

To create a Promise simple use the constructor with the propper methods:

```
const promise = new Promise((resolve, reject) => {
  // contain an operation
  // ...
  // return the state
  if (success) {
    resolve(value);
  } else {
    reject(error);
  }
});
```

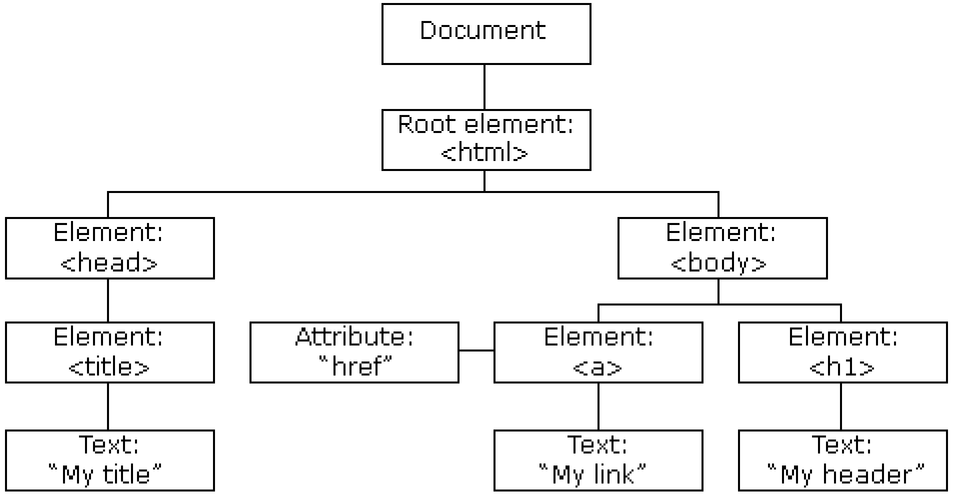
Promises

```
let success = false;

function getUsers() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (success) {
        resolve([
          { username: 'john', email: 'john@test.com' },
          { username: 'jane', email: 'jane@test.com' },
        ]);
      } else {
        reject('Failed to the user list');
      }
    }, 1000);
  });
}

const promise = getUsers();

promise.catch((error) => {
  console.log(error);
});
```

Exercise 3

Let’s make a Promise.

DOM



DOM stands for Document Object Model and is the way ES6 puts all the "document tree" inside the "**document**" object. Document is part of the Document class.
"document" is one of the most used objects inside ES6 in order to interact with webpages.

DOM

document – refers to the HTML documentthatis being presented, methods of this object that are useful are:

- getElementById()
- getElementsByTag()
- document.body
- document.title

A complete reference guide couldbe found on https://developer.mozilla.org/es/docs/Web/API/Document

DOM

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element

DOM

Changing HTML Elements

Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.write(text)	Write into the HTML output stream

Exercise 4

Let’s DOMinate some things.

Collections

ES6 provides a new collection type called **Map** that addresses these deficiencies.

Map object holds key-value pairs where values of any type can be used as either keys or values. In addition, a Map object remembers the original insertion order of the keys.

```
let map = new Map([iterable]);
```

Collections (cont...)

About collections

Two stpecial types of collections:

- Maps. Used for key-value pairs
- Sets. Use for general lists of objects

```
let userRoles = new Map();
```

Collections (cont...)

Useful JavaScript Map methods

clear() – removes all elements from the map object.

delete(key) – removes an element specified by the key. It returns if the element is in the map, or false if it does not.

entries() – returns a new Iterator object that contains an array of [key, value] for each element in the map object. The order of objects in the map is the same as the insertion order.

forEach(callback[, thisArg]) – invokes a callback for each key-value pair in the map in the insertion order. The optional thisArg parameter sets the this value for each callback.

Collections (cont...)

Useful JavaScript Map methods.

get(key) – returns the value associated with the key. If the key does not exist, it returns undefined.

has(key) – returns true if a value associated with the key exists, otherwise, return false.

keys() – returns a new Iterator that contains the keys for elements in insertion order.

Collections (cont...)

Useful JavaScript Map methods

set(key, value) – sets the value for the key in the map object. It returns the map object itself therefore you can chain this method with other methods.

values() returns a new iterator object that contains values for each element in insertion order.

Collections (cont...)

Useful JavaScript Map methods
 set(key, value) – sets the value for the key in the map object. It returns the map object itself therefore you can chain this method with other methods.
 values() returns a new iterator object that contains values for each element in insertion order.

```
let Mariano={name:"Mariano Rivera"},
Julio={name:"Julio Urias"},
Kershaw={"Kershaw Show};
let pitchers=new Map();
pitchers.set(Mariano, "relief");
pitchers.set(Julio, "starter");
```

Collections (cont...)

Set is used to manage unique values of any type

```
let setObject = new Set();

let setObject = new Set(iterableObject);
```

Collections (cont...)

The Set object provides the following useful methods:

add(value) – appends a new element with a specified value to the set. It returns the Set object, therefore, you can chain this method with another Set method.

clear() – removes all elements from the Set object.

delete(value) – deletes an element specified by the value.

entries()– returns a new Iterator that contains an array of [value, value] .

forEach(callback [, thisArg]) – invokes a callback on each element of the Set with the this value sets to thisArg in each call.

has(value) – returns true if an element with a given value is in the set, or false if it is not.

keys() – is the same as values() function.

[@@iterator] – returns a new Iterator object that contains values of all elements stored in the insertion order.

Proxy

Proxy(). Will allow us to intercept operations with objects and redefine the object behaviour. We can use Proxy() to let the programming side to be dynamic , Proxy() will allow us to intercept some operations in parts where this is the only way we could capture them.

```
var obj = {a: 1, b: 2};
var obj2 = new Proxy(
    obj,    {
        get : function(target, propertyKey) {
            console.log('get:', propertyKey);
            return Reflect.get(target, propertyKey);
        }
    }); // Referencia original
var c = obj.a;          // no message shown
// Supervised reference
var d = obj2.a;         // get: a
 // No existing propertie
var e = obj2.no_exist;  // get: no_exist
```

Exercise 5

Another DOMain exercise to relax.

Reflection

Reflection is the ability of a program to manipulate variables, properties, and methods of objects at runtime.
Prior to ES6, JavaScript already has reflection features even though they were not officially called that by the community or the specification. For example, methods like Object.keys(), Object.getOwnPropertyDescriptor(), and Array.isArray() are the classic reflection features.
ES6 introduces a new global object called Reflect that allows you to call methods, construct objects, get and set properties, manipulate and extend properties.
The Reflect API is important because it allows you to develop programs and frameworks that are able to handle dynamic code.

Reflection (cont...)

Reflect.apply() – call a function with specified arguments.
Reflect.construct() – act like the new operator, but as a function. It is equivalent to calling new target(...args).
Reflect.defineProperty() – is similar to Object.defineProperty(), but return a Boolean value indicating whether or not the property was successfully defined on the object.
Reflect.deleteProperty() – behave like the delete operator, but as a function. It’s equivalent to calling the delete objectName[propertyName].
Reflect.get() – return the value of a property

Reflection (cont...)

Reflect.getOwnPropertyDescriptor() – is similar to *Object.getOwnPropertyDescriptor().* It returns a property descriptor of a property if the property exists on the object, or undefined otherwise.

Reflect.getPrototypeOf() – is the same as Object.getPrototypeOf().

Reflect.has() – work like the in operator, but as a function. It returns a boolean indicating whether an property (either owned or inherited) exists.

Reflect.isExtensible() – is the same as Object.isExtensible().

Reflect.ownKeys() – return an array of the owned property keys (not inherited) of an object.

Reflect.preventExtensions() – is similar to Object.preventExtensions(). It returns a Boolean.

Reflect.set() – assign a value to a property and return a Boolean value which is true if the property is set successfully.

Reflect.setPrototypeOf() – set the prototype of an object.

Other usefull types and objects

Date() - Allows you to get the current date and time as well as manipulate dates inside the logic
XMLHttpRequest – Stablishes an AJAX request, this object is usually encapsulated to use microservices that will interact with the page
DOM objects – DOM refers to the Document Object Model that is the way to find elements inside a “document” created by an HTML page. The “root” element in the DOM is the **document** object.
Navigatorobject – Gives information about the navigator or app (browser) that is being used
Location object – Gives information about the curren user location

Other usefull types and objects
Date

Date() - Is the costructor, without parameters creates the actual date
Date(milliseconds) – Creates a new Date object with the value of a date since midnight Jan 1 1970
Date(dateString) – Uses Strings different formats that represents a Date and converts it to the object, for example: “Tue Mar 24 2015 18:00:00 GMT-0600” or "2015-03-25" are valid dateStrings.
Date(year, month, day, hours, minutes, seconds, milliseconds) – Creates a new Date with the given information

Methods

now() – Returns the number of milliseconds since midnight Jan 1, 1970
getYear(), getMonth(), getDay(), getDate() - Returns the values from the object, setters are also available

Exercise 6, 7 and 8

```
Let’s play with objects.
```

Other usefull types and objects XMLHttpRequest

This object is used for AJAX (HTTP) requests, methods and properties such as open(), send() and status arethemos common used

```
let request = obj => {
    return new Promise((resolve, reject) => {
        let xhr = new XMLHttpRequest();
        xhr.open(obj.method || "GET", obj.url);
        if (obj.headers) {
            Object.keys(obj.headers).forEach(key => {
                xhr.setRequestHeader(key,
obj.headers[key]);
            });
        }
        xhr.onload = () => {
            if (xhr.status >= 200 && xhr.status < 300) {
                resolve(xhr.response);
            } else {
                reject(xhr.statusText);
            }
        };
        xhr.onerror = () =>
reject(xhr.statusText);
        xhr.send(obj.body);
    });
};
```

Exercise 9, 10 and 11

```
You are themaster.
```

Day 1 summary

<u>Default function parameters</u> – learn how to set the default value for parameters of a function.
<u>Rest parameter</u> – introduce you to the rest parameter and how to use them effectively.
<u>Spread operator</u> – learn how to use the spread operator effectively.
<u>Object literal syntax extensions</u> – provide a new way to define object literal.
<u>for…of</u> – learn how to use the for...of loop to iterate over elements of an iterable object.
<u>Octal and binary literals</u> – provide support for binary literals and change the way to represent octal literals.
Template literals – learn how to substitute variables in a string.
Modules – learn how to substitute variables in a string.
Classes – Learn how classes are creted and how sublssing works in ES6
Symbol – Learn hw to create symbols and use them
Promises – Learn about asynchronos calls using Promises
Collections – Learn about Maps and Sets
Proxy & Reflection – Learn about how proxies and reflections works