ES6 – Javascript - Udemy

https://www.udemy.com/es6-bootcamp-next-generation-javascript/learn/v4

https://github.com/laczor/JS-ES6-udemy

0. Setup ES6 Babel for Node

https://github.com/babel/example-node-server

Session_3_Classes_Modules

Lesson 25 Exporting modules + Import

Lesson 26 Import Default, Import *

Lesson 28-34 Classes, Static Methods, Getters/Setters

/*STATIC Methods*/

/*GETTERS / SETTERS***/

Session_4_Symbols

Lesson 40 Symbols

Lesson 41 Shared Symbols

Lesson 43 Well Known Symbols

Session_5_Iterators&Generators

Lesson 45 Iterators & Generators

Lesson 47 Iterators In Action

Lesson 48 Custom Iterable Object

Lesson 49 Generators Basic

LESSON 50 Generators in Action

LESSON 52 Controlling Iterators

Lesson 52 * LinkedList Iterator Sample

Session_6_Promises

Lesson 53 Promises

Lesson 56 Chain Promises

Lesson 57 Catch Errors

Lesson 58 All + Race

Session_7_Extesion_Of_BuiltInObjects

It is just a showcase, what new features the default Objects have in ES6.

Session_8_Maps_Sets

Lesson71 Create a Map

Lesson72 Managing Map Items

Lesson73 Looping over the map

Lesson 75 Weakmaps

Lesson 78 Set

Lesson 78 Looping through the sets

Lesson 79 WeakSets

Session_9_Reflect_API

Lesson 83 Reflect_API

Lesson 84 Calling functions with Reflect API

Lesson 85 Reflect and Prototypes

Lesson 86 Reflect and Properties

Lesson 87 Reflect OwnProperties

Lesson 88 Create + Delete properties

Lesson 89 Object Extension

Session_10_Proxy_API

Lesson 92 Proxy Usage

Lesson 93 Proxy handler get method

Lesson 94 Proxy with Reflect

Lesson 95 Proxy with Prototype

Lesson 96 Proxy of proxies

Lesson 97 Wrap functions with Proxy!

Lesson 98 Revocable Proxies

Session 11 Project work

- index.html (traceur + sytem.import)
- Elements.js (DOM selectors)
- app.js (adding evenets)
- http.js (Static helper class with a XMLHttpRequest wrapped in a promise)
- Weather.js (via proxy + handler combo + Reflect API.)
- app.js (executing http + update WeatherObject + html attribut

Lesson 25 Exporting modules + Import

Modules are *always* in Strict Mode

Export: External.js

```
//1. Exporting 1 by 1.
 export let keyValue = 1000;
 export let test = function(){
        console.log('test');
 };

//1.1 Exporting together
 let keyValue = 1000;
 let test = function(){
        console.log('test');
 };

export {keyValue,test};
```

Import: app.js

```
//Lesson25 you can import other exported js files
//They will have a reference! on the exported files
import {keyValue,test} from './external.js';
import {} from './external.js';


console.log(keyValue);
test();
```

Lesson 26 Import Default, Import *

Default export, will export the defined variabel automaticall and it will be imported automatically as well.

external.js

```
        let defaultExport = "Default text";
        export default defaultExport;
```

app.js

```
        import defaultExport from './external.js';
        console.log('Default',defaultExport);
```

Import everything * as will import everything from the js file

```
import * as imported from './external.js';
console.log(imported.test());
```

Lesson 28-24 Classes, Static Methods, Getters/Setters

Classes.js /*****ES6 Classes*/

```
It is almost the same as the prototypal inheritance, only with “java” syntax
class Person {
        constructor(name){
                this.name = name;
        }
        greet(){
                console.log(`Hello this is the ${this.name}\'s greeting`);
        }
}
class Steve extends Person {
        constructor(name){
                super();
                this.name = name;
        }
        greetSteve(){
                console.log('i greet as Steve');
        }
        greetSuper(){
                super.greet();
        }
}
let stevey = new Steve("Istvan");

stevey.greet();         //Calling the parent function
stevey.greetSteve();    //Own function
stevey.greetSuper();    //Own function, calling the parent's super function
```

/***STATIC Methods*/**

If you define a class function as static, you can call them without instantiating the object.

```
class Helper {
        logService(message){
                console.log('Logged: ',message);
        }
}
```

```
let helperObj = new Helper();
helperObj.logService('Initialized Helper Object');

class HelperStatic {
        static logService(message){
                console.log('Logged: ',message);
        }
}

HelperStatic.logService('static function has been used');
```

/****GETTERS / SETTERS***/

You can control the classes “propert” names

PersonGetSet.name;

When you are trying to get it’s information or modify it.

```
class PersonGetSet {
        constructor(name){
//Due to setter, you have to  define it with this._name, not with
this.name
                this._name = name;
        }
//Will be activaded upon PersonGetSet.name
//
        get name(){
                return this._name.toUpperCase();
        }
        set name(value){
                if(value.length>3){
                        this._name = value;
                }else{
                        console.log("Rejected");
                }
        }
}
let p = new PersonGetSet('TestName');
console.log(p.name);            //Should be TESTNAME, due to built in getter
p.name = "SS";                  //Should "Rejected" due to built in setter
p.name = "Modified name";
console.log(p.name);            //Should be "Modified name"
```

Lesson 40 Symbols

New **primitive type** with a **unique ID**, **not iterable**!

```
/****Lesson 30****/

// Symbol is an object
let symbol = Symbol('name debug');      //name is for debugging
let symbol2 = Symbol('name2 debug');
console.log(symbol);                    //ObjectSymbol(name)
console.log(typeof symbol);             //"symbol"
console.log(symbol == symbol2);         //false, since symbols are unique

/***Perfect for METADATA like timestamp*/
let obj = {
        name: "Max",
        [symbol]:22,
}
console.log(obj);                       //symbol will be hidden
console.log(obj[symbol]);               //symbol still accessible
```

Lesson 42 Shared Symbols

Shared Symbols can be used, **to store and access Metadata** Safely.

```
/*Lesson 31 Shared symbol*/
let symbol3 = Symbol.for('age');
let symbol4 = Symbol.for('age');
console.log(symbol3 == symbol4);        //True; Will be shared, not unique!

//****Accessing Metadata outside of the function
//Also, the symbols are not overwriting the objects properties!!
let symbol0 = Symbol.for('sharedAge');

let person= {
        name : "Steve",
        age:30,
}
function changeAge(person){
        let ageSybmol = Symbol.for("sharedAge");
        person[ageSybmol] = 27;
}
changeAge(person);
console.log(person[symbol0]);                   //27
console.log('Original Property',person["age"]); //30
```

Lesson 43 Well Known Symbols

https://developer.mozilla.org/hu/docs/Web/JavaScript/Reference/Global_Objects/Symbol

```
// For the Clasess, there are quite of few of static fields
class Person {
}
Person.prototype[Symbol.toStringTag] = "Person";


let person0 = new Person();
console.log('Person Object-->',person0);                //Will be Object
Personnsole.log('Original Property',person["age"]);    //30
```

Lesson 45 Iterators & Generators

Generator : Function, which will **"yields"** value.

Iterator: Know how to access value of the collection.

```
// Iterator is an ojbect, which knows how to access the values of the
collection
let arr = [1,2,3];

//It will be a function,
console.log(typeof arr[Symbol.iterator]);

let it = arr[Symbol.iterator]();

//The Symbol iterator Object, has a function, which will return the next
Iterator Object
console.log(it.next());         //value --> 1
console.log(it.next());         //value --> 2
console.log(it.next());         //value --> 3
console.log(it.next());         //value --> nan
```

Lesson 47 Iterators In Action

```
//So we can actually overwrite the built in iterator function of the array
Object,
//And we are returning completely different custom values
let array = [1,2,3];

array[Symbol.iterator] = function(){
let nextValue = 10;
        return {
                next: function(){
                        nextValue++;
                        return {
                                done: nextValue >15 ?true:false,
                                value: nextValue,
                        }
                }
        };
};


for(let el of array){
        console.log(el);
}
```

Lesson 48 Custom Iterable Object

```
// With using custom iteration, we can determine, how and what properties
should be shown
let person = {
        name: "Steve",
        hobbies: ['Sports', 'Cooking'],
        [Symbol.iterator]: function(){
                let i = 0;                                      //To count the
loopings
                let hobbies = this.hobbies;     //To pass the Hobbies array
                return {
                        next:function(){
//reference current hobbies array
                                let value = hobbies[i];
                                i++;
                                return{
                                        done: i>hobbies.length?true:false
,               //condition for our iteration
                                        value: value,
                                };
                        },
                };
        },
}
//We can loop through our person object,
for(let hobby of person){
        console.log('Hobby',hobby);
}
```

Lesson 49 Generators Basic

```
/*Lesson 49. Generators Basic
Looping over the yielded values which will gives us iterable values*/
```

```
function* select(){
        yield 1;
        yield 2;
        yield 3;
}


let generator = select();


console.log(generator.next()); //{ value: 1, done: false }
console.log(generator.next()); //{ value: 2, done: false }
console.log(generator.next()); //{ value: 3, done: false }
console.log(generator.next()); //{ value: undefined, done: true }
```

LESSON 50 Generators in Action

```
let obj = {
        //We are passing the reference to our generator function object
        [Symbol.iterator]: gen,
}
//Simple generator which will provide iterable objects
function *gen(){
        yield 1;
        yield 2;
}
// Now we can loop over elements with the generator
for(let el of obj){
        console.log(el);
}
```

LESSON 52 Controlling Iterators

```
function *gen2(end){
        for(let i=0; i<end;i++){
                try{
                        yield i;
                }catch(err){
                        console.log(err);
                }
        };
}


let it = gen2(2);
console.log(it.next());
console.log(it.throw("error")); //Will catch our errror.
console.log(it.next());
```

Lesson 52 * LinkedList Iterator Sample

Will automatically, generate a generator in the LinkedList Object, which will access the Iterator Symbol

In the Iterator, the node will be determined, and the it will yield the next nodes continuously.

linkedListGenerator.js

```
  *[Symbol.iterator]() {
    let node = this.head;
    while (node) {
      yield node;
      node = node.next;
    }
  }
```

Lesson 53 Promises

Promise: Useful objects, helper to work with async tasks.

They give a promise that they will return back sth, either success, failure, or rejected.

```
/*Promise is an Object, which is created with a constructor function
its has two parameters
resolve function
reject function
*/
/*When we resolve our promise and provide back data
it will be visible in the then() function*/
let promise = new Promise(function(resolve,reject){
                        setTimeout(()=>{
                                // resolve("Done");
                                reject("failed");
                        },1500);
});


promise.then((value)=>{
        console.log('Finished',value);
},(error)=>{
        console.log('Normal Rejected',error);
});


// Promise Object - Resolve - Then (value(),error()) - custom function()
// "Done"
// Promise Object - Reject -  Then (value(),error()) - custom function()
// "Failed"
```

Lesson 56 Chain Promises

```
function waitASecond(seconds){
        return new Promise (function(resolve,reject){
                setTimeout(()=>{
                        seconds++;
                        resolve(seconds);
                },1000)
        });
}


waitASecond(0)
                .then(waitASecond)              //This function, will take the
seconds as arguments
                .then((seconds)=>{              //Just like here;
                        console.log('Calculated Resolve',seconds);
                }
);
```

Lesson 57 Catch Errors

```
function waitASecondCatch(seconds){
        return new Promise (function(resolve,reject){
                if(seconds>2){
                        reject("Rejected");
                }else{
                        setTimeout(()=>{
                                seconds++;
                                resolve(seconds);
                        },1000);
                }
        });
}
waitASecondCatch(3)
                .then(waitASecondCatch)         //This function, will take the
seconds as arguments
                .then((seconds)=>{              //Just like here;
                        console.log('Resolved Conditionally',seconds);
                }).catch((error)=>{
                        console.log('Rejected Conditionally',error);
                }
);
```

Lesson 58 All + Race

```
      // It will only resolve, if every promise in the array is resolved.
// ****ALL****, all promises has to finish!
let promise1 = new Promise(function(resolve,reject){
                        setTimeout(()=>{
                                resolve("Resolved");
                        },1000);
});


let promise2 = new Promise(function(resolve,reject){
                        setTimeout(()=>{
                                reject("Rejected");
                        },200);
});


Promise.all([promise1,promise2])
        .then((success)=>{
                console.log('All Promise',success);
        })
        .catch((error)=>{
                console.log('All Promise',error);
        });


// **RACE*** will wait for the fastest promise to resolve.
Promise.race([promise1,promise2])
        .then((success)=>{
                console.log('All Promise',success);
        })
        .catch((error)=>{
                console.log('All Promise',error);
        });
```

Lesson71 Create a Map

```
// Maps and Sets are new type of collections
// Array  +  Ojbects were available until now.
//****Map****//
// - Collection Object, which maps key-value pairs
let cardAce = {
        name:"Ace of Spades",
};
let cardKing = {
        name:"King of Clubs",
};
//1.0 Creating a map object and assigning with set
let deck = new Map();
deck.set('as',cardAce);
deck.set('kc',cardKing);

//2.0 Creating a map object with automatic assigment of an array of key-value
pairs
let deck2 = new Map([['as',cardAce],['kc',cardKing]]);
```

Lesson72 Managing Map Items

```
console.log(deck2);     //Will be a different object than the normal one
console.log(deck2.size);                //2
deck2.set('as',cardAce);                //Overwriting the existing key
console.log(deck2.get('as'));           //get your data
deck2.delete('as');     //you can delete a key with it's value completely
deck2.clear();                  //All the key-value pairs are deleted
```

Lesson73 Looping over the map

```
// Get back the values
for(value of deck.values()){
        console.log('value',value);
}
//Get back the keys
for(key of deck.keys()){
        console.log('key',key);
}
//Will get back the entry wrapped in an array / / [entry]
for(entry of deck){
        console.log('[entry]',entry);
}
```

Lesson 75 Weakmaps

https://stackoverflow.com/questions/29413222/what-are-the-actual-uses-of-es6-weakmap

WeakMaps: Are storing references to it's keys and values as well.

Keys are objects as well, and if they are not used, the weakMap will loose it's connection to the key's value, so it will be dropped as well.

This is actually a loose connections

```
let key1 = {a:1};
let key2 = {b:2};
let weakdeck = new WeakMap();
weakdeck.set('as',cardAce);
weakdeck.set('kc',cardKing);

console.log('WeakDeck',weakdeck.get(key1));
```

Lesson 78 Set

Can be used when you would like to store your uniquely identifiable data in an array, without worrying the duplicates

```
// Similar Arrays
// List of values without duplication
// Unqiue values


let set = new Set([1,1,1]);


set.add(1);             //Will not be executed since 1 is already in the set
set.add(2);             //will be added.
set.delete(2);  //can be deleted.
console.log(set.has(1));        //true, it contains 1
for(el of set){
        console.log(el);
};
console.log(set);
```

Lesson 78 Looping through the sets

```
for(el of set.entries()){
	console.log('[key,value]',el);
};
for(val of set.values()){
	console.log('value',val);
};
for(key of set.keys()){
	console.log('key',key);
};
```

Lesson 79 WeakSets

When you are holding only the objects until they are necessary.

```
// It is only storing objects.
// Unused object, will be garbage collected.
let obj1 = {a:1};
let set2 = new WeakSet([obj1,{b:2},{b:2}]);


// WeakSet is not enuremable!! No looping
//false since we are creating a new object, witj new object reference in this line
console.log(set2.has({b:2}));
console.log(set2.has(obj1));                    //True since the they are pointing to the same
set.delete(obj1);
```

Lesson 83 Reflect_API

(Properties created with Reflect are hidden it is for METADATA!)

https://developer.mozilla.org/en-S/docs/Web/JavaScript/Reference/Global_Objects/Reflect

Reflect is about grouping together the functions, properties, for the Objects

- Construct() -> will create an object, + you can define what should be the prototype
- Apply() -> calling The object functions with passing an other object as "**this**"
- setPrototypeOf() -> will set the prototype to be the passed object
- getPrototypeOf() -> will return the prototype of the passed object
- get() -> Will get the passed object's property
- set() -> Will set the selected Object's property
- ownKeys()-> Will provide a list of the passed Object's list of properties
- defineProperty() → Will let you create readable, writeable property
- deleteProperty() -> will let you delete a selected property from the Object
- preventExtensions() → will check if you can modify the properties or not

```
// // MetaProgramming Code
// // Evaluate our code at runtime.

class Person {
        constructor(name){
                this.name = name;
        }
}
function ProtObj(){
        this.age = 27;
}
//We can create Objects with reflect
// 1. Object to be created
// 2. Arguments to pass to the constructor function
// 3. To Assign a prototype object to the created Object
let person = Reflect.construct(Person,['Steve'],ProtObj);

console.log(person);
```

Lesson84 Calling functions with Reflect API

```
// // Good idea to use Reflect and use centralized approach
class Person2 {
        constructor(name,age){
                this.name = name;
                this.age = age;
        }
        greet(arg){
                console.log(arg +'Hello i am' + this.name);
        }
}
let person2 = Reflect.construct(Person2,['Steve',25],ProtObj);
Reflect.apply(person.greet,person,[]);
Reflect.apply(person.greet,{name:Anna},['...']);
//Will log out ...Hello i am Anna, since we passed an argument + an object to
refer as this
```

Lesson 85 Reflect and Prototypes

```
// You can access the prototype with reflect without usin __proto__
class Person3 {
        constructor(name){
                this.name = name;
        }
}

let person3 = new Person();
console.log(Reflect.getPrototypeOf(person));
console.log(Reflect.getPrototypeOf(person) == Person.prototype);
console.log(person.__proto__== Person.prototype);
// You can modify the prototype as well.
let proto = {
        age:30,
}

Reflect.setPrototype(person3,proto);
console.log(Reflect.getPrototypeOf(person));
```

Lesson 86 Reflect and Properties

You can **get** the object's properties, works even with a ES6 Class **getter/setter**

Also you can pass the object which properties you would like to modify

```
// Really useful for the dynamic part referencing object properties
class Personx {
        constructor(name,age){
                this.name = name;
                this._age = age;
        }
        get age(){
                return this._age;
        }
}
let mum = {
        _age:30,
}
let personx = new Personx("Steve",25);
//Can get property
console.log(Reflect.get(personx,"name")); //person.name;
//Getters can be used as well.
console.log(Reflect.get(personx,"age")); //person.name;
//Modifying object properties
Reflect.set(personx,"name","Istvan"); //person.name = "Anna";
//Object,Property,NewProperty,Object to refer as "this."
Reflect.set(personx,"name","mum2",mum); //person.name = "Anna";
console.log(Reflect.get(personx,"name")); //person.name;
//Using Apply for the getter
console.log(Reflect.get(personx,"age",mum)); //person.age.apply(mum);
```

Lesson 87 Reflect OwnProperties

```
//Core properties
console.log(Reflect.ownKeys(personx));
```

Lesson 88 Create + Delete properties

You can **defineProperty** of a passed object, can be **readOnly** or **Writable.**

In case of readOnly you can't modify it.

deleteProperty will just delete the selected property on the passed Object.

```
// 1.0 Creating properties
Reflect.defineProperty(personx, "hobby",{
});

// personx.hobby  = "Cooking";                  //can't assign to readonly
console.log('hobby not modified',personx);
// 1.1 Creating writable properties
Reflect.defineProperty(personx, "hobbies",{
        writable: true,
        value: ["Sports","Cooking"],
});

console.log('hobbies is hidden!',personx);
console.log('hobbies before',personx.hobbies);
personx.hobbies  = "Cooking";
console.log('hobbies after',personx.hobbies);
Reflect.deleteProperty(personx,'age');
```

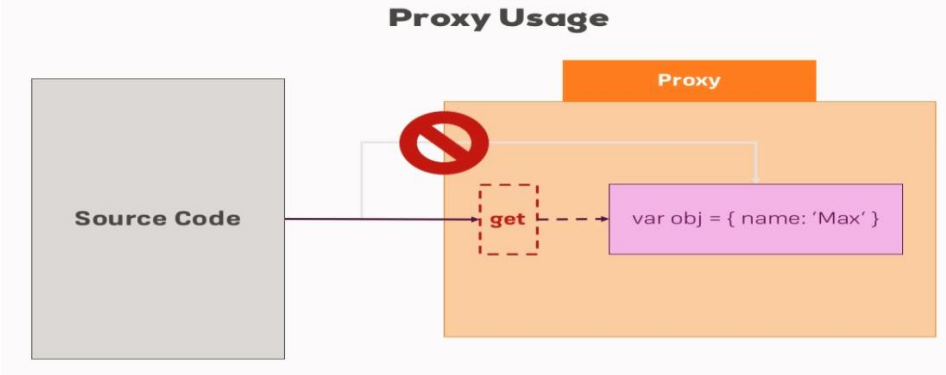
Lesson 89 Object Extension

```
// We can lock our object from extension
Reflect.preventExtensions(personx);
//We will see false, since it won't be extensible
console.log(Reflect.isExtensible(personx));
```

Lesson 92 Proxy Usage

Basically, you wrap your object in an outer object, which will process any request prior reaching the inner object.

https://developer.mozilla.org/hu/docs/Web/JavaScript/Reference/Global_Objects/Proxy



Lesson 93 Proxy handler get method

```
let person = {
        name: 'Istvan',
        age : 25,
};

//You can define all of the available traps, which you would like to process
prior reaching the inner object
let handler = {
//target: Object
//name: is the property to access
        get:function(target,name){
//**1**Check if the name exist in the target, if yes, it will return the
object property, else some taxt
                return name in target ? target[name]:"Doesn't exists";
        },

};
//Target object, + the handler object, with the set traps
var proxy = new Proxy(person,handler);

console.log(proxy.age);       //Will return the object, since it is there
console.log(proxy.surname);   //will return text, instead of undefined
```

Lesson 94 Proxy with Reflect

You can combine the Reflect API to modify objects, with proxy.

```
set: function(target,property,value){
                if(value.length >2){
//**2**We can combine it with Reflect as well.
// Reflect.set(target,property,value);
                        Reflect.set(target,property,value);
                }
        }
proxy.name = "hi";
console.log(proxy.name);
//Will not overwrite the name since we have a setter handler


proxy.name = "Istvan Man!";
console.log(proxy.name);
//Will not overwrite the name since we have a setter handler
```

Lesson 95 Proxy with Prototype

You can make the target object, to inherit the proxy's methods, to add extra protection if you are trying to access a property, which is not there.

Proxy_prototype.js

```
let person = {  name: 'Istvan', age : 25 };
//You can define all of the available traps, which you would like to process
prior reaching the inner object
let handler = {
        get:function(target,name){
                return name in target ? target[name]:"Doesn't exists";
        }
};
var proxy = new Proxy({},handler);
//We will get back   person's name property
console.log(person.name);
//We set proxy to be the prototype of person Object
Reflect.setPrototypeOf(person,proxy);
//Since person, inherited all of the proxy's properties, functions, getters,
setters ..
// it will log out the text from our handler.
console.log(person.hobbies);
```

Lesson 96 Proxy of proxies

```
let handler_proto = {
};

//We create a proxy object, with an empty handler
let proto_proxy = new Proxy({},handler_proto);
//For create empty proxy, we add a new proxy, with an additional handler
let proxy2 = new Proxy(proto_proxy,handler);


//We set the person's prototype to be the proxy2 object
Reflect.setPrototypeOf(person,proxy2);
```

Lesson 97 Wrap functions with Proxy!

Wrapper_function.js

```
//As a result, the following will be the prototypal chain
// handler_proto --> handler -> person
// proto_proxy --> proxy2 -> person
console.log(person.hobbies);

// You can wrap function objects as well!!
function log(message){
        console.log(message);



}

let handler = {
        apply : function(target,thisArg,argumentsList){
                if(argumentsList.length ==1){
                                        return
Reflect.apply(target,thisArg,argumentsList);
                }
        }
}

let proxy = new Proxy(log,handler);
proxy("Hello",10);
```

Lesson 98 Revocable Proxies

After you set a proxy to be a wrapper, you can deactivate it!

```
let person2 = {
        name:"Istvan"
};


let handler2 ={
                get:function(target,name){
                return Reflect.get(target,property);
        },
};


//Destructure the created Proxy object, to store the proxy + the revoke
property of the created object in seperate objects.
let {proxy3,revoke} = Proxy.revocable(person2,handler2);
// revoke will be a function object, of the proxy's objectproperty
console.log(proxy3.name); //okay
revoke();   //will remove the proxy wrapper
console.log(proxy3.name); //no proxy wrapped around the object
```

Session 11 Project work

Index.html

Using traceur + sytem.import for modularized js files.

```
  <!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Project</title>
    <!-- traceur is an es6 compiler -->
    <script src="https://google.github.io/traceur-
compiler/bin/traceur.js"></script>
    <!-- ajax is to make async data request -
<script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.31/system.js">
</script>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<header>
    <h1>The Super Awesome Weather App</h1>
</header>
<div class="main">
    <div id="search">
        <label for="city">City Name</label>
        <input type="text" id="city">
        <button type="button">Show Weather!</button>
    </div>
    <div id="load">Loading...</div>
    <div id="weather">
        <h1 id="weatherCity">City Name</h1>
        <div id="weatherDescription">Weather Description</div>
        <div id="weatherTemperature">Temperature</div>
    </div>
</div>
<!-- <script src="app.js"></script> -->
<!-- instead of including the js file, we use modularized approach to import
it. -->
<script>
     System.import('app.js');
</script></body></html>
```

Elements.js

It is a good practice to select the elements in a separate JS file

```
export const ELEMENT_SEARCH_BUTTON = document.querySelector('button');
export const ELEMENT_SEARCHED_CITY = document.querySelector('#city');
export const ELEMENT_LOADING_TEXT = document.querySelector('#load');
export const ELEMENT_WEATHER_BOX = document.querySelector('#weather');
export const ELEMENT_WEATHER_CITY = ELEMENT_WEATHER_BOX.firstElementChild;
```

app.js

You can import all of the selected elements, **an assign event listeners to them**

```
import * as ELEMENTS from 'elements.js';

ELEMENTS.ELEMENT_SEARCH_BUTTON.addEventListener('click', searchWeather);
```

http.js

Static helper class with a **`XMLHttpRequest`** wrapped in a **promise**

```
// Helper class, static methods can be used even if you do not instanciate it.
// 1. We wrap everything in a promise
// then we make an XMLHttpRequest GET request, with the URl
// This when request Done && 200 for XMLHttpRequest we parse the request to be
JSOn
// or resolve the error // then we send the http request.
export class Http {
    static fetchData(url) {
        return new Promise((resolve, reject) => {
            const HTTP = new XMLHttpRequest();
            HTTP.open('GET', url);
            HTTP.onreadystatechange = function() {
           if (HTTP.readyState == XMLHttpRequest.DONE && HTTP.status == 200) {
                    const RESPONSE_DATA = JSON.parse(HTTP.responseText);
                    resolve(RESPONSE_DATA);
                } else if (HTTP.readyState == XMLHttpRequest.DONE) {
                    console.log('jo');
                    reject('Something went wrong');
                }
            };
            HTTP.send();
        });
    }
}
```

Weather.js

We create a separate class, with a handler. So we have an object, with the weather properties, and we can modify it's properties only via **proxy** + **handler** combo + **Reflect API.**

So it is quite protected!

```
// So we are creatin a weather data class, which will store the weather
information
// We also use a handler, which will inferfere when we would like to modify
it's properties using Proxy.
// WIth Reflect Api we modify the created object properties


export class WeatherData {
    constructor(cityName, description) {
        this.cityName = cityName;
        this.description = description;
        this.temperature = '';
    }
}


//This is the handler which will get passed to the proxy.
export const WEATHER_PROXY_HANDLER = {
    get: function(target, property) {
        return Reflect.get(target, property);
    },
    set: function(target, property, value) {
        const newValue = (value * 1.8 + 32).toFixed(2) + 'F.';
        return Reflect.set(target, property, newValue);
    }
};
```

App.js

Executing our http request then updating the data with our Object,proxy,handlerReflectAPI combination, then modify the css properties.

```
// So, it is actually an event, which will be triggered on clicking the search
button
ELEMENTS.ELEMENT_SEARCH_BUTTON.addEventListener('click', searchWeather);
// 0. the cityname will be trimmed + checked
// 1. The elements will get their css properties
// 2. URL string is made
// 3. our XMLHttpRequest wrapped in a promise is executed.
// 4. when it is successfully done, we modify the WeatherData object, via
proxy + handler + Reflect APIT
function searchWeather() {
    const CITY_NAME = ELEMENTS.ELEMENT_SEARCHED_CITY.value.trim();
    if (CITY_NAME.length == 0) {
        return alert('Please enter a city name');
    }
    ELEMENTS.ELEMENT_LOADING_TEXT.style.display = 'block';
    ELEMENTS.ELEMENT_WEATHER_BOX.style.display = 'none';
    const URL = 'http://api.openweathermap.org/data/2.5/weather?q=' +
CITY_NAME + '&units=metric&appid=' + APP_ID;
    Http.fetchData(URL)
        .then(responseData => {
            const WEATHER_DATA = new WeatherData(CITY_NAME,
responseData.weather[0].description.toUpperCase());
            const WEATHER_PROXY = new Proxy(WEATHER_DATA,
WEATHER_PROXY_HANDLER);
            WEATHER_PROXY.temperature = responseData.main.temp;
            updateWeather(WEATHER_PROXY);
        })
        .catch(error => alert(error));
}
```