**JavaScript**

* In 2009 it became a cross-platform language, which can be used to create programs for desktops, cellphones, and the web.
* In 2009 Node.js was introduced which is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
* developer.mozilla.org/en-US/docs/Glossary/JavaScript برای یادگیری
* jsbin.com برای خود کد زنی
* electron js: Electron is a free and open-source software framework developed and maintained by GitHub. It allows for the development of desktop GUI applications using web technologies: it combines the Chromium rendering engine and the Node.js runtime.
* In JavaScript, you can pass functions around as values.
* We usually write down the first letter of the name of classes and constructor functions in upper case to distinguish them from normal functions.

**ECMAScript**

* for standardization of JavaScript from 6 onward. So instead of having so many JavaScript versions, we can have a standard and if a company wants to add something to JavaScript they have to ask for ECMAScript to confirm it.

**Variables**

* var, let in ecmascript6
* var variable = function() { return x } // Anonymous function functions as variable values is possible, they are like defining a function with the usual method. You call the name of that variable because it is considered to be a function.

var func = function () { return x } == function func { return x}

Func() ; // for both ways to call the function

**Scopes**

* blocks
* Global: If a global var is defined in another block we can't access it, it returns undefined if it is accessed before a local definition in that block
* Local
* ‘use strict’ so we have to use var creating a variable and we can’t miss it

**Conditions**

* If/else if/else
* Switch/case/break/default. when the number of checking something in ifs is more than 3 times and they usually have the same pattern we use switch/case. Every case value is compared by the condition and shows the output like (x>y) == condition, returns true or false

switch(condition){

case (x>y) :

do;

Break;

default:

do;

break;

}

switch(condition){

case x :

case y :

Do;

break;

case z :

Do;

break;

}

**Loops**

* For // used usually for the arrays, defining i with var makes it accessible out of the block which shows the last value. And it’s not good.
* While
* Do while: is executed once at least, for example: making a unique password and compare with DB passwords and create again until it is unique

**Arrays**

* [ ]
* Lists can have duplicate items
* We can add an item to an index that doesn’t exist, it keeps the previous indexes empty

var list = [1,2,3]

list[2] = true

list[6] = 'me'

[5, 2, true, empty × 3, 'me'] // 3 index don’t have any value

**List.properties**

* **List.lenght**
* **Array.forEach()** // takes a list and separates all the items and then gives it to the Anonymous function function

var total = 0

array.forEach(function(item){

Total += item

});

این آیتم ها روforeach خودش می ده به فانکشن بینام اون کاری که می خوایم رو انجام می دیم، روی آیتم ها جلو می ره و دیگه نیاز به استفاده از طول نیست.

* **List.reduce()** // The passed parameters are the previously calculated value and the current value of the list. Previous at the start is the default value, then on the second iteration, it holds the value of the first calculation and so on.

var total = array.reduce(function(total, value){

return total + value // any calculation

}, 10);

این آیتم ها رو reduce خودش یه پردازشی می کنه می ده به فانکشن بینام اون کاری که می خوایم رو انجام می دیم، روی آیتم ها جلو می ره و دیگه نیاز به استفاده از طول نیست. باید برای total یک مقدار اولیه بدیم وگرنه اولین آیتم رو می گیره. یک پارمتر دیگه داره که بعد فانکشن بی نام قرار می گیره که همون دیفالت total هست. تو این مثال 10 .

* **List.push(item)** // adds to the right of the array and returns the index.
* **List.pop()** // deletes the last item from the right side of the list and returns the item.
* **List.unshift(item)** // adds to the left of the list and returns the length of the list.
* **List.shift()** // deletes the last item from the left of the list and returns the item.
* **List.indexOf(item)** // returns the index of the item if it exists
* **List.splice(startindex,howmany)** // removes that part from the main list and returns it as a new list
* **List.slice(startindex,endindex)** // returns that part as a new list and the original list remains the same
* **List.join(‘char’)** // Joins items together and returns it as a string with desired separator char, default char is a comma
* **List.concat(list2)** // returns a new combined list of the two [first list][list2], it doesn’t affect the original lists
* **List.sort()** // sorts the main list, it has many options
* **List.reverse()** // reverses the main list
* **List.filter(func(){});** // returns a new list

list.filter(function(item){

return item % 2 == 0

}

اگر **true** بشه اون آیتم ها رو بر می گردونه یعنی بر اساس شرط فیلتر می کنه

* **List.map(func(){});** // returns a new list

اون **عملیات** رو انجام می ده روی تک تک آیتم ها و به شکل یک لیست بهمون بر می گردونه

list.map(function(item){

return item \* 2

}

**Strings Ecmascript5**

* **String.length**
* **String.charAt(index)** // returns the char
* **String[index]** // returns the char
* **String. cancat(string2)** // returns a new combined string of the two strings, it doesn’t affect the original strings
* **String.toUpperCase()** // returns the string in uppercase, does not effect the original list
* **String.toLowerCase()** // returns the string in lowercase, does not effect the original list
* **String.split(char)** // returns a list of strings splitted by the char
* **String.trim()** // it wipes all the excess spaces, it doesn’t effect the original strings
* **String.indexOf(substring)** // it retruns -1 if it doesn’t find the substring in the string
* **String.substr(strartIndex,endIndex)**
* **String.replace(regexp or string, string2)** // replaces the string2 with found substring in the first passed argument

**Objects**

* **{ }** // to keep data in a variable in the form of key/value
* **Key/value** : the value of keys can be a list, integer, string, object, function

var obj = {

'Key1' : '2'

'key2' : 'this' ,

'detail' : {

D1 : [1,2,3],

D2 : ''

}

'fn' : function (){

Return (this.key1 \*2);

}

}

* **obj.key or obj[key**] // To add to or access and change (get and set) its properties use the dot, for example: obj.fn()
* **Obj[“key-1”]** // They have the same naming rules as var but if you want to use dash char - for the keys use “ ” and for accessing it use [ ]
* **This.key** // to access the proprieties of the obj inside obj
* Another way of defining an object in eccma6 is with a new Object()

var obj = { }

var obj = new Object()

obj.keys ; // for both ways to call the properties

**For in**

* It is used to traverse objects

let obj = {

name: 'foo',

age : 44

}

**for(** let o in obj ) {

console.log( o.**key**, o.**value**)

}

**Constructor functions**

Functions can be treated like objects too.

When you call a function in the usual way it does not recognize *this* keyword. But when you define it as an object the *this* can be accessed. Because objects are alive in memory and we have to distinguish them and their properties from other objects.

\*function func(){

this.prop1 = ‘val’;// default value

this.prop2 = 5;

this.func = function(){

}

}

var obj = new func; // obj is an object with the name of its constructor that is the name the func

console.log(obj.prop1)

* Why should we use them? Because we can make different objects.

\*function func(prop1, prop2){

this.prop1 = prop1

this.prop2 = prop2

this.func = function(){

}

}

var obj = new func(‘val’,2);

var obj = new func(‘val2’,3);

function person(name,family) {

    this.name = name;

    this.family = family;

    this.find\_the\_length = function () {

        // namepars

        return this.name.length;

    }

}

var person1 = new person('Sarah', 'Zit')

var person1 = new person('Jackaline', 'Scott')

console.log(person1.find\_the\_length())

* Instanceof // person instanceof Object returns true, instanceof checks that if smth is an instance of something else.
* Person1.cunstructor // returns the constructor of that object aka the definition of the person const function
* Defining objects with only {} is also using an object constructor which is native to javascript core codes and is interpreted through browser javascript engines which have the native codes like the Object constructor function (var obj = new Object();).

**Prototypes**

* Prototypes are objects that let us access properties of the objects we create or are defined in native codes.
* When it comes to inheritance, JavaScript only has one construct: objects. Each object has a private property that holds a **link** to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. By definition, null has no prototype and acts as the final link in this **prototype chain**.
* Nearly all objects in JavaScript are instances of [**Object**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) which sits on the top of a prototype chain.
* The methods and properties are **not** copied from one object to another in the **prototype chain**. They are accessed by walking up the chain through links. It checks the constructor function, if methods and properties are not found there, it checks the prototype of the constructor function, and so on.
* Inheritance is based on prototypes. We can access the properties of the main Object constructor because when we defined our object there is a prototype for our own constructor function and there is a prototype for the main Object constructor that is given to all objects.
* Every object we create has a main prototype that includes these main objects
  + the constructor function that we defined ourselves can override the properties of the main Object constructor. It becomes the main constructor function of that object.
  + Methods we define as a prototype for the constructor function (not as the main methods), using keyword prototype

person.prototype.fun = function() {

    return 'fun';

}

* + the native code Object constructor and all its properties are inherited

They can have their own prototypes.

* By using keyword prototype we can add methods whenever we want to any constructor, even the main Object constructor. So all objects that we make from that constructor have that method.
* We can create a list or a string using new List() and new String() constructors. They are all objects. It’s the same as we define them normally. They inherit the main Object constructor properties.
* Look at it in the browser console by calling the name of an object.

**Inheritance**

**Obj from obj**

* It’s not often used for the frontend but it’s used in the backend like Nodejs
* var obj = Object.create(null) // creates an object with no prototype, not even main Object constructor
* obj.\_\_proto\_\_ // shows the prototypes of the object
* we can make an object inherit from another object with its properties and values

var car4 = Object.create(car3);// car4 has the same properties and

values as car3

// we can define new properties

car4.makeASmile = function() {

    console.log(':)')

}

* Important: All the inherited parts from parents are set as a prototype and its overridden or newly defined properties are shown as the usual properties and not prototypes.

var car4 = Object.create(null)

car4.makeASmile = function() {

    console.log(':)')

}

var car5 = Object.create(car4);

car5.color = 'Yellow';

car5.year = 2010;

In browser inspect: car5

1. *{color: 'Yellow', year: 2010}*
   1. color: "Yellow"
   2. year: 2010
   3. *[[Prototype]]*: Object
      1. makeASmile: *ƒ ()*

**Bind, Apply, Call methods**

* To access an object's properties through the keyword ‘this’ inside an external method. We can use arguments for that function and pass the properties like that, but in specific circumstances, we want to conveniently read the properties of an object inside another function by using the keyword this.
* We can achieve it by using these methods. They are mostly alike have small differences.

**Apply**

* Func.apply(obj) // we don’t call the function with parentheses, Apply firstly processes obj and makes it accessible in func and then calls it.
* Func.apply(obj,[arg1,arg2]) // pass other arguments for func with a list

**Call**

* Similar to apply
* Func.apply(obj, arg1, arg2) // pass other arguments for func with comma

**Bind**

* after passing the obj, it doesn’t call the function automatically, we have to call it

\* func.bind(obj, arg1, arg2)()

\* Var f1 = func.bind(obj, arg1, arg2);

F1();

* We can also use them to pass an object to a method of another object

\*var car = {

make : 'sipa',

model : '111',

displayDetails : function(year , name) {

console.log(year , name)

console.log('Maker : ' + this.make + ' , Model : ' + this.model)

}

}

var car2 = {

make : 'sipa',

model : 'tiba2',

}

car.displayDetails.apply(car2, [2012 , 'hesam'])

**Debugging**

* Using console.log()

    console.log('Start javascript');

* Debugger statement; If no debugging is available, the debugger statement has no effect.

    debugger;

* Debuggers in editors
* Chrome, dev tool

**Visual** **studio**

* In the debugger setting, we chose chrome launch
* For only HTML files in launch.json instead of URL we use this option:

"file": "${workspaceFolder}/index.html",

* we can see the value of variables by hovering over them
* Using the watch section in the visual studio debugger we can track variables by adding their names

**Chrome**

* in console tab in dev tool, it shows the file and the line, example: test.js:17
* In the source tab in the dev tool, set breakpoints, and then reload the page

**Callbacks**

https:// stackoverflow.com/questions/23882149/are-javascript-callbacks-just-anonymous-functions-sent-as-an-argument-in-a-funct

JavaScript callbacks are functions passed as values into an asynchronous function for the purpose of continuation.

A function has another function as a value when something happens, it executes that call back function itself. It calls it and executes it. Like eventListeners callbacks.

**Functions are values:**

So in JavaScript, you can pass functions around as values. You can reference a function in several ways:

* Pass a literal anonymous function as the callback

doSomeWork(function (err, result) {

if (err) {

throw new Error(err);

} else {

console.log(result);

}

});

* Pass a literally named function as the callback

doSomeWork(function magicalCallback(err, result) {

if (err) {

throw new Error(err);

} else {

console.log(result);

}

});

(Naming every function is a smart idea because you can see it in the stack trace)

* Pass in the value of a variable that happens to be storing a function as the callback

var someFunction = function callItWhatYouWant(err, result) {

if (err) {

throw new Error(err);

} else {

console.log(result);

}

}// reference the callback stored in the someFunction variable

doSomeWork(someFunction);

* Pass in the function by referencing the function name as the callback

function callItWhatYouWant(err, result) {

if (err) {

throw new Error(err);

} else {

console.log(result);

}

}

// reference the callback function using the function name

doSomeWork(callItWhatYouWant);

**Try/catch/finally**

* Try some code and if an error has occurred catch it
* The error that is caught is an object with two properties, error. name, and error.message.
* Throw keyword. In a certain context, the throw is used to create custom errors and then send it to catch. It accepts Boolean, numbers and strings, and objects. We can create our own properties for this object so we can do whatever we want with it inside catch.
* Finally is always executed after the try/catch
* Common errors: syntaxError, refrenceError, typeError

**Timer/interval**

* setTimeout(function (){ do what you want},time in millisecond) // this method has two parameters, Do something after a certain time
* setInterval(callbackfunction, time)// Do something periodically, the parameters are the same as timeout
* We write websites sliders with intervals

**Date**

* Mozilla.org, date section,

https:// developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Date

* It’s better to store time in The Gregorian calendar in DB, we can use packages to show it in other calendars
* new Date()
* Unix Timestamp standard is a method that defines the time in milliseconds from the first day of 1970, we can store these milliseconds and then use the language API to convert it to the usual format. Past or present or future.

Var d = new Date();

d.getTime()// returns 1634552143579 for example

var d2 = new Date(1634552143579+ 1000)// add one second, it returns the usual date format

* Date.now() // now in miliseconds Using Date objects
* Date.parse(year/month/day) // returns time in milliseconds
* Set and get methods, getters, and setters, for Date

**Math**

[https:// developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)

* Math.PI // returns the p number
* Math.abs(number) // returns the absolute of the number,  returns a positive or zero form of the number, if the number is ‘null’ it returns 0
* Math.floor(number) // returns the largest integer less than or equal to a given number.
* Math.random(number)

// It returns random numbers from 0 to max (does not include max)

// why? because even if you have (0.9999999999999 \* max) and then floor it it’ll be less than max,

// because it is never 1 to return the max

// random returns a number between 0 and 1: 0.6031627200095537

Math.random()

function getRandomInt(max) {

    return Math.floor( Math.random() \* Math.floor(max))

}

* Math.round(number) // If the decimal is 0.5 or more, it gives the next bigger integer number and if it is less than 0.5, it gives the previous integer number
* Math.pow(number, pow)

**DOM (Document Object Model)**

* document is an object, having all the tags of the page, to make the page elements dynamic. DOM is this object.
* All of those children are also objects
* **document.children** in dev tool, returns the children of the document which has the html too, they are objects with their methods. html-> (head ->(link, meta), body->div)
* document.**head**, document.**body** // accessing special tags
* document.body.**children[0].style.**color =””
* document.body.children[0].**textContent** = “”
* document.body.**firstChild** // usually returns text of that element
* document.body.**firstElementChild**
* document.body.firstElementChild.**nextElementSibling**
* document.getElement(s)By

Id

TagName

ClassName

Name

* Document.querySelector(‘.list’) // returns one item
* Document.querySelectorAll(‘.list’),(‘li’),(‘// list’) // it returns a list that we can loop through with forEach
* Document.querySelectorAll(‘ul li’)[0].style

**Adding elements to DOM**

* Document.createElement(‘ul’)
* el.classlist.add(‘class-name’)
* el.toggle(‘class-name’) // adds the class if it does not exist, removes it if it exists
* el.id = ‘value’
* el.setAttribute(‘id’, ‘value’)
* el.appendChild(el2)
* el.after(el2) // adds el to the after of the el
* el.remove // removes the element
* el.removeChild(‘ul’) // removes the child and returns the it
* el.parentElement

**Dialog**

* It’s better that we don’t use the default javascript dialogs because they don’t have a good style.
* Sweetalert is a package among packages with this purpose that we can use instead, it uses HTML to create dialog
* alert () function // It’s related to the browser, we can’t change it visually that much. at first, it was used for logging.
* confirm() // return true or false, in the past was used to ask the user if they want to quit a form
* prompt() // It has input to fill that return the value of that input

**Events**

[https:// developer.mozilla.org/en-US/docs/Web/API/Event](https://developer.mozilla.org/en-US/docs/Web/API/Event)

* both server-side and client-side

**EventHandler**

* If you look at the prototype in an el, you see some **onX** (onclick for example)methods. These are the methods that work with events as eventHandlers. They are mostly empty on default.
* Some of them are specific to elements like onchange in inputs.
* el.onclick = function () { } // we can add a function to that eventHandler for that element. Like defining function with var x = function() {}. It runs el.onclick()
* browser implement these handlers differently which we have to consider
* The event happens and it is given to the handler.

el.onclick = function (event) { consol.log(event)} // returns the mouseclick object

el.onclick = function (event) { event.target.style.background = 'green'; console.log(event);}

let title = document.querySelector('// title')

title.onmouseenter = function(event) {

event.target.style.backgroundColor = 'red';

}

title.onmouseout = function(e) {

e.target.style.backgroundColor = 'yellow';

}

* We can’t have more than one event handler

**EventListener**

* This is a better method
* We can have more than one event listener for the same el.
* el.addEventListenere(event type, callback function)

el.addEventListener( ‘click’, function(event) {

// code1

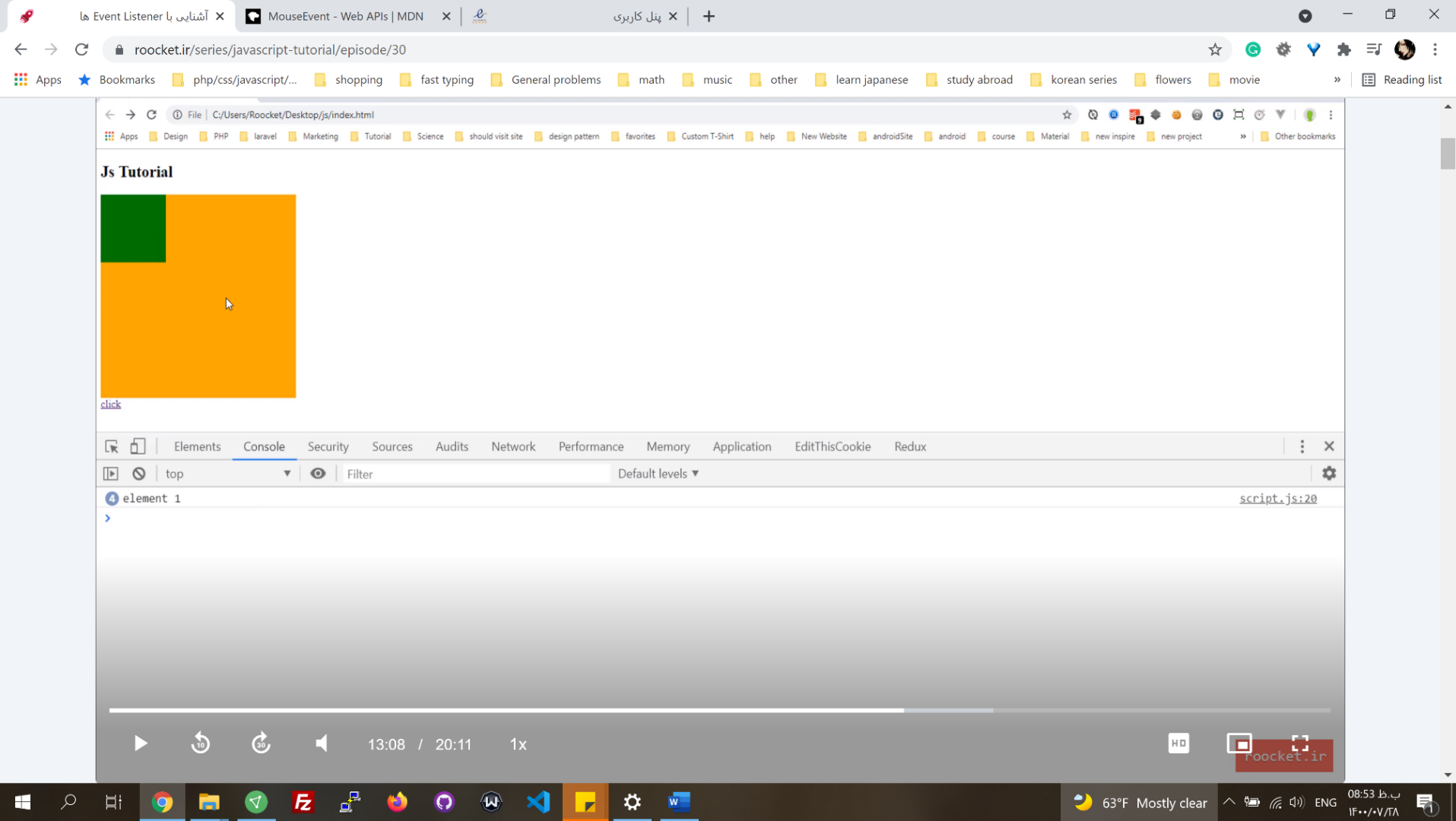
});

el.addEventListener( ‘click’, function(event) {

// code2

});

* event.stopPropagation() // If we have eventListeners both on the child and the parent, the event on child triggers both the child callback and the parent callback. Sometimes this can cause a problem. It can be prevented. We can handle this by event passed to the callback function. It is an object, having methods and properties. One of them is **stopPropagation()** that stops the effects on the parents.



parent

parent

title.addEventListener( ‘click’, function(event) {

event.stopPropagation()

// code

});

|  |  |  |
| --- | --- | --- |
| 0 | UNSENT | Client has been created. open() not called yet. |
| 1 | OPENED | open() has been called. |
| 2 | HEADERS\_RECEIVED | send() has been called, and headers and status are available. |
| 3 | LOADING | Downloading; responseText holds partial data. |
| 4 | DONE | The operation is complete. |

// code

});

* event.preventDefault // prevents event default behavior like clicking on links or submitting on the form. Sometimes we need that. For example when we create a link for submitting a review but we only need to open a modal and work on that. Informs too when we use ajax we use this to prevent sending info to another route.
* el.removeEventListener( event type, callback function) // To remove listeners we have to create a name for the callback functions so we specify what we want to remove. We can put the callback in a var.

**HttpRequest and** **Ajax**

* The pure javascript has a constructor function to send and receive data from a URL, an object can be defined from this constructor function.
* XMLHttpRequest // It’s a built in object in browsers, so it must run on browsers

[https:// developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest)

* HTTP Methods: GET, POST, PUT, HEAD, DELETE, PATCH, OPTIONS
* XMLHttpRequest.readyState, 0, 1, 2, 3 ,4

|  |  |  |
| --- | --- | --- |
| 0 | UNSENT | Client has been created. open() not called yet. |
| 1 | OPENED | open() has been called. |
| 2 | HEADERS\_RECEIVED | send() has been called, and headers and status are available. |
| 3 | LOADING | Downloading; responseText holds partial data. |
| 4 | DONE | The operation is complete. |
| 0 | UNSENT | The client has been created. open() not called yet. |
| 1 | OPENED | open() has been called. |

* Status: An HTTP status code is a server response to a browser’s request.

|  |  |
| --- | --- |
| HTTP Status Code 200 | Get request OK. |
| HTTP Status Code 201 | Post request OK. |
| HTTP Status Code 301 | Permanent Redirect. |
| HTTP Status Code 302 | Temporary Redirect. |
| HTTP Status Code 404 | Not Found. |
| HTTP Status Code 410 | Gone. |
| HTTP Status Code 500 | Internal Server Error. |
| HTTP Status Code 503 | Service Unavailable. |

**Get**

* Receiving data from a URL

var xhttp = new XMLHttpRequest();

// eventHandlers onreadystatechange, we can use just once

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

// Typical action to be performed when the document is ready:

document.getElementById("demo").innerHTML = xhttp.responseText;// plain text

}

};

xhttp.open("GET", "filename", true);

xhttp.send();

var xhttp = new XMLHttpRequest();

// eventHandlers onreadystatechange

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

// Typical action to be performed when the document is ready:

document.getElementById("demo").innerHTML = xhttp.responseText;

}

};

xhttp.open("GET", "filename", true);

xhttp.send();

**Post**

* Send data to a URL
* Headers specify what kind of data type we are sending. XMLHttpRequest.setRequestHeader(header, value)

var xhr = new XMLHttpRequest();

xhr.open("POST", '/server', true);

// Send the proper header information along with the request

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

// XMLHttpRequest.DONE = 4

// event listener, can have many listeners.

xhr.addEventListener(‘load’, function() { // Call a function when the state changes.

if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {

// Request finished. Do processing here.

}

});

// the data sent is in a format that server accepts.

xhr.send("foo=bar&lorem=ipsum");

// xhr.send(new Int8Array());

// xhr.send(document);

// xhr.send(JSON.stringify(data));// to send json

* JSONPlaceholder is a website for test API services. [https:// jsonplaceholder.typicode.com/users](https://jsonplaceholder.typicode.com/users)
* To see the request in the dev tool in chrome, look at the network tab
* The pure XMLHttpRequest is not suggested, because it’s troublesome and the code is long. So it’s better to use libraries.

**Libraries**

* JQuery // API methods to work with its core, single page application
* Axios Ajax Specific library
* React js // Creating the user interface
* React router

**Frameworks**

* Putting together a few libraries
* Vue js // virtual DOM, single-page apps, can be used for normal websites.
* Angular // Only used to create single-page applications
* alpinejs // A rugged, minimal framework for composing JavaScript behavior in your markup

<template **x-for**="post in posts">

<h2 x-text="post.title"></h2>

</template>

* React Bootstrap. React-Bootstrap replaces Bootstrap JavaScript.