|  |
| --- |
| javascript & react |

* تفاوت بین null and udefined در چیست ؟

ما در جاوااسکریپت انواع داده رو داریم که به صورت زیر میباشد :

{string , Number , Boolean , Object , Symbol , Null , undefined}

اولین مورد این که هر ردو عبارت falsy اند یعنی وقتی ما اونارو به Boolean تبدیل میکنیم خروجی برای ما false .

برای این که بفهمیم خروجی یه عبارت falsy هست یا true از Operator Double-Not (!!) استفاده میکنیم و اونارو به Boolean تبدیل میکنیم.

Console.log(!! null) false

Console.log(!! undefined) false

یه روش دیگه که میتونیم بفهمیم یه عبارت false هست یا نه استفاده از تابع Boolean به این صورت:

Console.log(Boolean(null)) false

Console.log(Boolean(undefined)) false

* برسسی Undefinde :

1. اگر متغیری رو تعریف کنیم که بدون مقدار باشد undefined برمیگردونه.

Let X;

Console.log(x) undefined

1. اگر تابعی تعریف کنیم که بدون return باشد اونم undefined برمیگردونه:

Function noReturn() {}

Console.log(noReturn())

1. مورد سوم Property رو از یه آبجکت صدا میکنیم که میدونیم این Property وجود نداره:

Const Person = {}

Console.log(Person.name) undefined

* بررسی Null

در برنامه اگر بخواهیم متغیری رو با مقدار پوچ یا هیچ تعریف کنیم از null استفاده میکنیم.

Const x = null;

برای این که نوع null رو بفهمیم از typeof استفاده میکنیم.

Console.log(typeof null) object

مقایسه null , undefined با == و ===

Console.log(null == undefined) true

Console.log(null === undefined) false

* عملگر && and چیکار میکنه ؟

عملگر منطقی && میاد بررسی میکنه اگر در یک خروجی بین دو یا جند عبارت قرار بگیره آیا همه یاین عبارت ها Truesy هستند یا خیر !

اگر همه ی عبارات Truesy باشند خروجی که به ما برمیگردونه برابر با آخرین عبارت Truesy مونه

اما اگر بین این عبارت ها مقداری falsy باشد خروجی برار اولین عبارت falsy و عبارت های بعد از اون عبارت falsy نادیده گرفته میشن.

Example

Const expr1 = true && “Hello” && 10; 10

Const expr2 = “Jump” && null && false; null

* عملگر || or چیکار میکنه؟

عملگر || اگر بین 2 یا چند عبارت قرار بگیره دنبال اولین عبارت Truesy میگرده و اگر عملگر منطقی || نتونه هیچ عبارت Truesy پیدا کنه خروجی رو برار با آخربن عبارت falsy قرار میده

Csonst expr1 = null || undefined || “hello” hello

Const expr2 = “hello” || true hello

Const expr3 = null || undefined || false false

* سریع ترین راه تبدیل یک رشته به عدد جیه؟

بهترین روش استفاده از Unary Plus میباشد که قبلا از رشته + قرار میدیم . طبق چند مثال زیر :

Const X = “10”

Console.log(+ X) 10

Console.log(typeof X ) string

Console.log(number) number

Console.log(+ ’12.7’) 12.7

Console.log(+ “ “) 0

Console.log(+ true ) 1

Console.log(+ false ) 0

Console.log(+ “Hello”) NaN

* ParseInt & ParseFloat

Console.log(ParseInt(“12.7 Hello”)) 12

Console.log(parseFloat(“12.7 Hello”)) 12.7

* DOM

Document object model

یک رابط کاربری یا API که به ما این اجازه رو میده تا بتونیم سند html مون رو دست کاری کنیم

Document

لیستی از همه ی element هایی که تو اون صفحه قرار داره

* Event Propogation چیکار میکنه؟

در کلمه به معنای گسترش و ترویج میباشد

توی جاوااسکریپت وقتی به یک element یک رویداد ضمیمه میکنیم این رویداد ممکنه فقط برای این element اتفاق نیوفته و ممکنه element های دیگه رو هم تحت تاثیر قرار بده ، به این تحت تاثیر فرار دادن به اصطلاح Event Propogation گفته میشه

Event Bubbling

توی event bubbling رویداد از event داخلی به event بیرونی گسترش پیدا میکنه

Event Capturing

اگر به آرگومان سوم addEventlistener مقدار true رو بدیم Event Capturing اتفاق میافته

درست نقطه مقابل Event Bubbling یعنی رویداد از event بیرونی به درونی اتفاق میافته.

<div>

<p>I’m a Div</p>

<button>I’m a button</button>

</div>

<script>

Document.querySelector(‘div’).addEventListener(‘click’ , function(){

Alert(‘div’);

},true)

Document.querySelector(‘button’).addEventListener(‘click’, function() {

Alert(‘button’)

})

</script>

* فرق بین متد های PreventDefault and StopPropagation چیه؟

preventDefault : اگر method preventDefault روی یک element اعمال بشه مانع عملکرد ذاتی اون element مشه

برای مثال اگر روی یک link اعمال بشه دیگر اون link کار نمیکنه.

Example:

<a href=<http://google.com>>click</a>

Document.querySelector(‘a’).onClick=function(event) {

Event.preventDefault()

}

stopPropagation : باعث میشه هیچ ترویج و گسترشی انجانم نگیره.

* چطوری ببینیم preventDefault روی یک element اعمال شده یا نه؟

همه ی المنت ها یه property داذن به اسم defaultPrevented اگر true باشه یعنی اعمال شده ولی اگر false باشه یعنی اعمال نشده.

* چرا این کد به خطا منجر میشه ؟

Catch the Error

برای این که مثلا Error undefined نداشته باشیم باید از عملگر Optionol chaining استفاده کنیم که در Es6 به javascript اضافه شده

در مثال زیر object رو تعریف میکنیم و هیچ داده ای رو بهش نمیدیم.

Const book = {}

Console.log(book.page) undefined

Console.log(book.title) undefined

Console.log(book.author) undefined

Console.log(book ? .page ? .number || 300)

اگر book بود دنبال page بگرد اگر page بود دنبال number بگرد و اگر هیچ کدوم نبودن undefined برنگردون ، 300 رو که مقدار پیش فرض دادیم رو نشون بده.

* Event target چیه؟

شامل element که باعث اجرا شدن یک رویداد خاصی میشه.

Example:

Event.target.tagName

بهمون میگه از چه تگی داخل این element استفاده شده.

* Event.currentTarget چیه؟

شامل element که یک رویداد به اون نسبت داده شده

Example:

Event.currentTarget.tagName

که نشون میده رویداد به کدوم element ضمیر شده.

فرق بین == و === چیه؟

Const X = 10;

Const Y = “10”;

Console.log(X == Y) true

چرا true برمیگردونه ؟ وقتی دو مقدار ظاهر مشابهی دارنولی نوع متفاوتی دارن قبل از این که یه مقایسه واقعی روی آن ها انجام بشه دو مساوی یه تبدیل نوع موقت انجام میده.

1. String => Number
2. Boolean => 1 or 0
3. Object => Primitive using .value of or .toString

Primitive => Strings , Number , Boolean …

در سه مساوی زمانی مقادیر باهم برابرن که نوعشون هم باهم برابر باشن.

* وقتی دو آبجکت مشابه رو باهم مقایسه میکنیم خروجی چیه؟

Const X = {a : 10}

Const Y = {a: 10}

Console.log(X === Y) false

وقتی 2 آبجکت رو باهم دیگر مقایسه میکینم این آدرس یا رفرنس اوناست که باهم دیگه مقایسه میشه نه ظاهر اون ها

* ما چه زمانی میتونیم از مقایسه 2 آبجکت مشابه خروجی True بگیریم ؟

Cosnt X = {a : 10}

Const Y = X

Console.log(X === Y) true

در این حالت مقدار X رو نمیریزیم تو Y بلکه آدرسشو نسبت میدیم به Y

* عملگر Double Not !! چیکار میکنه؟

وقتی !! قبل از مقدار یا عبارتی قرار میگیره ، مقدار یا عبارت اون رو به حالت Boolean تبدیل میکنه یعنی True یا False

Console.log(!! null ) => false

Console.log(!! undefined ) => false

Console.log(!! ‘ ‘ ) => false

Console.log(!! 0 ) => false

Console.log(!! Nan) => false

Console.log(!! {} ) => true

Console.log(!! [] ) => true

Console.log(!! 1) => true

* چجوری توی یک خط یک خط یک عبارت رو محاسبه کنیم ؟

Inline Evaluation

Function addFive (num) {

Return num + 5;

}

Let X = 5;

X++;

X = addFive(X);

X = X \* 2;

X = X – 5;

X = X + 10;

روش دوم => (در یک خط )

X = ( X++ , X = AddFive(X) , X \*= 2 , X-= 5 , X+= 10 )

عبارت هارو داخل پارانتز مینویسیم و با کاما از هم دیگه جدا میکنیم.

* Scope چیست ؟

Scope یا حوزه به جاهایی تو برنامه نویسی گفته میشه که مثلا متغیری تعریف میشه و آن متغیر در آن حوزه قابل استفاده و دسترس که ما 3 مدل scope داریم :

1. Global Scope
2. Local Scope (function Scope)
3. Block Scope
4. Global scope

Var user = ‘sina;

Function Person() {

Console.log(user);

}

Person(); => sina

1. Local scope

Function Person() {

Var user = ‘sina’

}

Console.log(user) => user is not defined

در clocal scope متغیرهایی که داخل function تعریف میکنیم بیرون از function به آن ها دسترسی نداریم .

1. Block Scope

Block scope رو جاوااسکریپت در آپدیت Es6 معرفی کرد و مختص Let , Const میباشد که داخل یک {} نوشته میشن.

{

Let X = 10;

Const Y = 20;

Var Z = 30;

}

به متغیر هایی که داخل این بلاک ها نوشته میشن در حوزه block scope میباشن و خارج از این آبجکت غیر قابل دسترس میباشن و این موضوع var رو در بر نمیگیره و قابل دسترسه.

* Hoisting چیست ؟

در معنی به معنای بالا بردن میباشد . و در جاوااسکریپت به این معناست که قبل از اجرای واقعی که تمام متغیر ها و توابع به بالاترین قسمت حوزه خودشون جا به جا میشن . به این اتفاق میکن Hoisting

Hello()

Function Hello() {

Console.log(‘Hello’); => hello

}

* تفاوت Hoisting در let and var ؟

Console.log(X)

Let X = 10 => cannot acces ‘X’ berore initialization

Console.log(X)

Var X = 10 => undefined

* use strict چیکار میکنه؟

برنامه رو موضف میکنه تا با سخت گیری بیشتری برسیش کنه

* Closure چیه ؟

به سادگی نوشتن یک تابع تو یک تابع دیگست.

این روش مشکلاتی هم دارد:

مشکل اول امنیت داده ست یا Data Encapsulation

طبق مثال زیر :

اول یک متغیر تعریف میکنیم:

Var counter = 0;

Function Jump() {

Counter++;

}

Jump()

Jump()

Jump()

Console.log(counter) => 3

و ما این متغیر رو در یک تابع دیگر هم تعریف میکنیم بدون این که از تابع اولی اطلاع داشته باشد اون رو هم رو خودش اعمال میکنه.

Function fly() {

Counter++;

}

Flu()

Console.log(counter) => 4

مشکل دوم ثبات داده یا persistency هست . مثال:

در این جا امنیت داده داریم ولی ثبات داده نه . و این مشکل با closure حل میشه

Function jump() {

Var counter = 0;

Counter++

Return counter

}

Console.log(jump()) => 1

Console.log(jump() => 1

* مقدار های falsy چی هستند ؟

مقدار های falsy مقدار هایی هستند که وقتی ما آن هارو به حالت Boolean شون تبدیل میکنیم مقدار false رو برمیگردونن.

Const falsyValue = [

“ “,

0,

Null,

Undefined,

NaN,

False,

]

چطور بررسی کنیم یک عبارت falsy هست یا خیر ؟

روش اول استفاده از double not !!

روش دوم استفاده از Boolean

This چیه و چه مقادیری داره ؟

کلمه کلیدی this که ما معمولا اون رو توی توابع استفاده میکنیم به قسمتی از کد اشاره میکنه که الان داره تابع مارو اجرا میکنه.

Example:

Var name = ‘sina’

Function hello() {

Console.log(this.name);

}

Hello() => sina

* Prototype چیست؟

معنی لغوی Prototype به معنای نمونه اولیه میباشد

هر مقداری که ما تعریف میکنیم شامل یه سری ویژگی ها هستند که از والد یا نمونه اولیه به ارث میبره مثل Property length که در آرایه ها و ابجک ها به کار می بریم

به این صورت میتونیم به prototype دسترسی داشته باشیم:

Var o = {}

o.—proto--: => object{…}

* متد apply چیکار میکنه ؟

متد apply کاری میکنه که this به object مورد نظر ما اشاره کنه تحت هر شرایطی .

ویژگی که این متد داره قابلیت استفاده مجدد از تابع رو به ما میده

Const user = {

Name() {

Return `${this.firstname} ${this.lastname}`

}

}

Const sinaghaffari= {

Firstname: ‘sina’

Lastname : ‘ghaffari’

}

Console.log(user.name.apply(sinaghaffari)) => sina ghafari

* متد call چیکار میکنه ؟

تفاوت apply , call در ارسال آرگیومنت به تابع مورد نظره

طبق مثال زیر :

Let car = {

Make: ‘sipa’,

Model: ‘tiba2’,

}

Function displayDetails(year, name) {

Console.log(year, name)

Console.log(‘maker: ‘ + this..make , ‘model : ‘ + this..model)

}

Apply:

displayDetail.apply(car, [ 2020 , ‘sina’] )

به صورت array دریافت میکنه

call :

displayDetail.call(car , 2020 , ‘sina’)

با کاما جدا میکنه

* متد bind چیکار میکنه ؟

تفاوت bind با دو متد قبلی در اینه که نمیاد درجا تابع رو اجرا کند.

دو روش برای اجرای bind وجود داره یکی این که میتونیم داخل یک متغیر بریزیم و اون متغیر رو اجرا کنیم یا این که یه پارانتز خالی جلوی تابع قرار بریم.

به این صورت :

Let a = displayDetails.bind(car , 2020 , ‘sina’ )

Or

displayDetails.bind(car , 2020 , ‘sina’ )()

Asynchronous request درخواست های غیر هم زمان

* Promise ها چی هستند ؟

Promise ها برای هندل کردن async به کار میره

( then , catch )

Let myFirstPromise = new Promise((resolve , reject) => {

setTimeout(() => {

reject(‘this is an error’ );

},2000)

});

myFirstPromise

.then(data => {

Console.log(data)

})

.catch(err => console.log(err))

* Callback

با استفاده از callback میتونیم توابع async مون رو هر جایی که دلمون خواست فراخوانی بکنیم و ازش استفاده کنیم.

یکی از مشکلاتی که به callback ها وارده می شه به روش Handle کردن error ها اشاره کرد.

مثال:

Function doSomething(callback, error) {

setTimeOut(() => {

let title = ‘this is article one’;

if(! Title) {error (‘err’) }

setTimeout(() => {

let data = {user: ‘ ‘ }

if(data) {error (‘err data’);return اگر اررور داشت ادامه نده}

callback(data)

},2000);

},2000)

}

doSomething(function(data) {

Console.log(data)

}, function (error) {

Console.log(error)

})

Async and Await

آخرین آپدیت جاوااسکرپت برای فرایند های asynchronous میباشد.

در توابع async , await میتونیم از then , catch نیز استفاده کنیم

Const fetchData = async() => {

Try {

Let response = await axios.get(‘url’)

Console.log(response.data)

} catch(err) {

Console.log(err)

}

}

fetchData()

Higher Order Functions

Filter map reduce

توابع مرتبه بالا

یکی از قابلیت های جاوااسکریپت که اونو به یه زبان برنامه نویس فانکشنال مناسب تبدیل کرده اینه که اجازه میده از توابع مرتبه بالا استفاده کنین.

توابع مرتبه بالا توابعی هستند که یک تابع را به عنوان پارامتر ورودی دریافت و یا به عنوان خروجی ارسال می کنند دانستن این که چه زمانی و چگونه از این توابع استفاده میشود ضروری است ، استفاده از این توابع باعث میشه کدهایی که مینویسیم ساده تر و قابل فهم تر باشد.

* Filter

فیلتر یک متد برای آرایه هاست ، پارامتری که filter میپذیرد یک فانکشن تست است که حتما خروجی آن باید Boolean باشد ، خروجی تابع فیلتر یک آرایه جدید از عضو های تابع اولیه است که در تست فانکشن true را برگردانند.

مثال:

تصور کنید کدی را مینویسید که شامل لیست لیست افرادی است که میخواهید با فیلتر افرادی که 18 سال و یا بیشتر از آن دارد را پیدا کند؟

Const People = [

{name: ‘john Doe’ , age : 16},

{name: ‘sina gh’ , age: 19},

{name: ‘Amin gh’ , age : 20},

{name: ‘Dad’ , age: 18}

]

حال میخواهیم با استفاده مرتبه بالا و filter() کدی بنویسیم که مشابه کار کد بالا رو انجام بده.

Const PeopleAbove18 = (collection) => {

Return collection.filter((person) =>

Person.age >= 18)

}

Map

یکی دیگه از چیزایی که در توابع مرتبه بالا استفاده میشود متد map() است ، که یک فانکشن را دریافت میکند و آن را بر روی تک تک اعضای آرایه اعمال میکند.

برای مثال در یک رشته از اعداد وقتی بخواهیم همه ی اعضا را به توان 2 برسانیم از متد map() استفاده میکنیم.

Const numbers = [1 ,2 , 3 , 4 , 5 , 6, 10 ,20 ]

Const squoes = numbers.map((x) => {

Return Math.pow(x , 2)

)}

Console.log(squars)

و یا برای نمایش نام و سن افراد در مثال قبلی از کد زیر استفاده میکنیم.

Const showInfo = people.map((arr) => {

Return arr.name + “|” + arr.age

)}

Reduce

آخرین و قدرتمندترین متدی که تو این مطلب در موردش حرف میزنیم reduce هست ، با این تابع میتونیم هر گونه تغیری که دلمون میخواد ایجاد کنیم.

Reduce یک تابع و یک مقدار اولیه برای شروع دریافت میکنه .

چند مثال با Reduce :

ضرب کردن اعضای آرایه :

Const numbers = [2,3,4,5]

Const product = numbers.reduce((acc) => {

Return acc \* 2 , 1

)}

پیدا کردن اعداد زوج

Const isEvent = (x) => {

Return x % 2 === 0; اگر یک باشد یعنی فرد

Const numbers = [12,324,213,4,2,3,45,42,34]

Const callback = (acc , x ) => {

If(isEvent(x)) {

Acc.push(x)

}

Return acc

}

Const evenNumbers = numbers.reduce(callback, [] )

Console.log(evenNumbers)

Arguments

توابع جاوااسکریپت یک شی پیش فرض و تعبیه شده به نام arguments دارن .

شی آرگیومنت (arguments ) شامل آرایه ای از آرگومان هاست که تابع زمانی که فراخوانی (اجرا) شد ، تابع از آن ها استفاده میکند.

با این روش شما می توانید به سادگی از یک تابع برای پیدا کردن (برای مثال) بزرگترین مقدار در لیستی از اعداد استفاده کنید.

X = findMax(1,123,500,115,44,88)

Function findMax() {

Var I;

Var Max = -infinity;

For(I = 0 ; i < arguments.length ; i++ ) {

If(arguments[i] > Max) {

Max = arguments[i]

}}

Return Max;

}

Console.log(findMax(4,5,6))

یا میتوانیم یک تابع برای جمع کردن تمامی مقادیر ورودی بسازید

X = sumAll(1,123,500,115,44,88)

Function sumAll() {

Var I ;

Var sum = 0;

For(i=0 ; I < arguments.length; i++ ) {

Sum += arguments[i]

}

Return sum;

}

Document.getElementById(‘demo’).innerHTML = sumAll(1,123,500,115,44,88)

چگون آبجکت بدون Prototype بسازیم ؟

خب همه ی ما با آبجکت ها تیو js آشنایی داریم آبجکت توی js از جفت name and Property ساخته میشه از اونجا که همه چیز توی js از Prototype ارث بری دارند، پس وقتی یک آبجکت لیست یا رشته میسازیم یک سری متد و ویژگی برای استفاده بر روی اون شی در اختیار ما قرار میده.

در واقع یک رشته (و هر شی دیگه توی sj ) به همراه خودش تعداد زیادی ویژگی و متد داره که به اون ها اثر جانبی Side-effects میگیم.

به عنوان یک مثال ساده ، ایست زیر رو در نظر بگیرید این لیست بعد از تعریف ، شامل یک سری از خصوصیت ها میشه ، مثلا همین خصوصیت پرکاربرد length که با گذاشتن یک نقطه بعد از لیست میتونیم بعش دسترسی داشته باشیم و تعداد آرایه های اون لیست رو به ما نشون بده . این ویژگی length از کجا اومده و چطوری به همراه یک شی لیست که همین الان تعریفش کردیم وجود داره.

Const myList = [1,2,3,4,5]

myList.length => 5

خوب همون طور که گفتیم همه چیز از Prototype ساخته میشه.

وقتی توی جاوااسکریپت یک شی جدید میسازیم (یک رشته و لیست و آبجکت ) در واقع داریم یک شی میسازیم که از کلاس Prototype ارث بری داشته پس یک سری ویژگی و خصوصیت ، همراه خودش داره (مثل همون ویژگی length که توی لیست ها مثال زدیم )

وقتی ما یه آبجکت رو به این صورت میسازیم

Const myObject = {}

جاوااسکریپت اون رو به این شکل میبینه

Object.create(object.Prototype)

حالا اگه بخواهیم آبجکتی بسازیم که این ویژگی هارو همراه خودش نداشته باشه.

Const myObj = Object.create(null)

این شکل از ساختن آبجکت باعث میشه ویژگی اه و متد هایی که لازم نداریم اصلا وجود نداشته باشد، بریم همین موضوع رو در عمل ببینیم. (در console )

با قرار دادن null در خروجی آبجکت میتونیم یک آبجکت بدون هیچ side-effects بنویسیم .

این آبجکت بدون اثر جانبی چه استفاده ای دارد؟

معمولا وقتی میخواهیم از آبجکت js برای ذخیره دیتا استفاده کنیم نیازی به متد ها و ویژگی های پیش فرض آبجکت نداریم ، میتونیم با این روش به سادگی آبجکتی بسازیم که هیچ گونه وابستگی به چیزی نداره.

استفاده از localStorage در جاوااسکریپت

به طور خلاصه یه سری اطلاعات رو توی مرورگر کاربر به طور دائمی ذخیره کنیم.

localStorage چیه؟

localStorage یک مکان ذخیره سازی هست که توی مرورگر کاربر وجود داره که این امکان رو به ما میده تا با جاوااسکریپت یه سری اطلاعات رو ذخیره و استفاده کنیم .

در واقع عملیات ثبت ، خواندن، بروز رسانی و حذف (CRUD ) رو میتونیم با فضایی که در اختیار داریم امجام بدیم ، اطلاعاتی که در LOCALsTORAGE ذخیره میشه دائمی است ، یعنی اگر کاربر تب مرورگر و یا حتی خود مرورگر رو ببنده این اطلاعات حذف نمیشن.

متد LocalStorage :

setItem()

getItem()

removeItem()

clear()

بررسی متد های localStorage

setItem()

این متد برای استفاده میشه که میخواییم یه سری اطلاعات رو توی localStorage ثبت و یا اطلاعات قبلی رو ویرایش کنیم ، این متد دو آرگومان میگیره ، اولی کلید اون مقداری که قراره ثبت و یا ویرایش بشه و پارامتر دوم شامل مقدار این کلید هست.

localStorage(‘name’ , ‘sina ghaffari’)

تو این مثال name کلید و sina ghafari مقدار این کلید هست

ما میتونیم یه مقدار پیچیده تر مثلا یک آبجکت رو توی localStorage ذخیره کنیم . باید دقت داشته باشیم که توی LocalStorage باید رشته ذخیره کنیم . پس قبل از ذخیره آبجکت باید اون رو تبدیل به رشته کنیم

Const person = {

Name: ‘sina’,

lastName: ‘ghaffari’

}

Person = JSON.stringify(person)

localStorage.setItem(‘user’ , person)

getItem()

همون طور که از اسمش پیداست این متد برای گرفتن یک آیتماز localStorage استفاده میشه این متد یک پارامتر میگیره که اسم کلید اون آیتم مورد نظر ما هست ، خروجی این متد یک رشته هست

مثال اول

Const name = localStorage.getItem(‘name’)

Console.log(name) => sina ghafari

توی مثال دوم چون آبجک رو به رشته تبدیل کردیم موقع فراخوانی اون آیتم باید دو باره به آبجکت تبریلش کنیم ک

Let user = localStorage.getItem(‘user’)

User = JSON.parse(user)

Console.log(user)

// {“name” : ‘sina’ , “lastname” : ‘ghaffari’}

removeItem()

این متد برای حذف کردن یک آیتم از localStorage به کار میره این متد یک ورودی میگیره که شامل کلید آیتم مورد نظر ما هست:

localStorage.removeItem(‘user’)

console.log(localStorage.getItem(‘user’)

// null

Clear()

این متد همه ی آیتم هایی که توی localStorage وجود داره رو پاک میکنه اینمتد هیچ پارامتری نمیگیره.

localStorage()

\*\*\*

1.حداکثر فضایی که در اختیار داریم 5 مگابایت هست

2. اطلاعاتی که ذخیره میکنیم توسط مرورگر قابل دسترسی هست پس باید حواسمون باشداطلاعات حساس رو ذخیره نکنیم.

تفاوت var , let , const در چیست؟

تفاوت var , let و const تو این هستش که اگر شما از var استفاده کنین تقریبا در تمام کل یک فایل جاوااسکریپتی می تونید اون متغیر رو درخواست بدید و از مقادیرش استفاده کنین البته استثناهایی هم وجود داره.

ولی let به این صورت نیست، اگر شما بین دو تا آکولاد از let استفاده کنید دیگه خارج ار این آکولاد {} ها نمیتونین به مقدار let دسترسی داشته باشین.

Const هم یه نوع متغیر هست که تغیرات خاصی نمیشه توش انجام داد ولی در کل مجموعه میشه فراخوانیش کرد، در واقع const یه نوع ثابته.

Var x = 10;

// here x is 10

{

Const x = 2;

// here x is 2

}

//here x is 10

Arrow function چیه ؟ و تفاوت اون با regular function در چیست؟

فانکشن arrow یه ویژگی جدید هستش که در Es6 معرفی شد و شکل مختصر برای نوشتن فانکشن ها و regular function ها هر دو کاری یکسان را انجام میدن ولی تفاوت هایی هم با هم دارن.

1. سینتکس

فانکشن arrow به ما اجازه میده تا همون توابع معمولی رو با تعداد کد کمتری بنویسیم ، کروشه ها دیگر توی دستورات خطی نیاز نیستند.

Let add = (x , y ) => x + y

کد بالا تو آرگومان x , y رو میگیره و جمع اون هارو توی متغیر add ذخیره میکنه، اگر فقط یک آرگومان داشته باشیم دیگه حتی به پارانتز ها هم نیاز نداریم.

Let squareNum = x => x \* x

اگر آرگومان هم نداشته باشیم به این شکله:

Let sayHi = \_ => console.log(“hi”)

Let sayHi = () => console.log(“hi”)

1. کار با آرگومان ها

Function arrow نمیتونن در مقدار arguments تغیری ایجاد کنند ولی به آبجکت آرگومان های نزدیکترین تابع non arrow والدشون دسترسی دارن.

در توابع معمولی داریم:

Let myFunc = {

showArgs() {

console.log(arguments);

}

}

myFunc.showArgs(1,2,3,4)

inspect: /

0 : 1

1: 2

2: 3

3: 4

ولی در توابع arrow

Let myFunc = {

showArgs: () => {

console.log(…arguments)

}

}

Myfunc.showArgs(1,2,3,4)

Inspect: /

Error

1. کلید واژه this

بر خلاف توابع معمولی ، توابع arrow چیزی به نام this برای خودشون ندارن ، مقدار this در این توابع خارج از lifecycle اون تابعه و مربوط میشه به نزدیک ترین تابع non-arrow

Me.thisInArrow()

Me.this.InRegular()

1. کلید واژه new

توابع معمولی قابل فراخوانی و construct هستند و میشه از new برای آن ها استفاده کرد.

ولی توابع arrow تنها میتونستن فراخوانی بشن و نمیشه شی جدیدی ازشون ساخت، به همین دلیل نمیتوان ان ها را new کرد.

Let add = ( x , y ) => console.log(x + y)

New add (2,3)

Inspect: / Error

1. پارامتر ها نمیتونن نام تکراری داشته باشن

در توابع Arrow جاوااسکریپت نمی تونیم برای آرگومان ها و پارامترهای مختلف یک نام انتخاب کنید ،

دستور زیر در جاوااسکریپت معتبر است :

Function add (x , x ) {}

که البته وقتی از strict mode استفاده میکنیم معتبر نخواهد بود

“use strict”

Function add (x , x) {}

//syntax Error

اما در Arrow function به هیچ عنوان نیتوانیم از پارامتر هایی با نام یکسان و تکراری استفاده کینم.

(x , x ) => {}

//syntax Error

Object Destructuring چیکار میکنه ؟

Destructuring یک روش ساده و سریع برای ایجاد متغیره ، مقادیر متغیر های ایجاد شده از مقادیر داخل یک آرایه یا مشخصه های داخل یک آبجکت به دست میان ، مزیت این روش خوانایی و بهره وری کد نوشته شده است.

مثال:

Const person = {

firstName: ‘sina’,

lastName: ‘ghaffari’,

phoneNumber: 0098241

}

به کمک object Destructurgin ایجاد متغیر ها به سادگی انجام میشه .

Const { firstName , lastName } = person;

و الان میتونیم هر جا که خواستیم از این مقادیر استفاده کنیم.

میتونیم این روش رو روی آرایه هم پیاده کنیم

Array Destructuring

Const brands = [‘ mclaren’ , ‘Renault’ , ‘Ferrari’ ]

بدون استفاده از array destructuring

Const brand = brands[0]

Console.log(brand) => mclaren

با استفاده از array destructuring

Const [ brand ] = brands

Console.log( brand ) => mclaren

همون طور که مشخصه برداشتن آیتم اول از آرایه سادست . فرض کنیم بخواییم ایتم سوم را برداریم ، این آیتم در اندیس 2 قرار داره ولی نمیتونیم از اندیس استفاده کینم . بهمین جهت مجبوریم دستی بشماریم و آیتم هایی که نمی خواییم رو خالی بذاریم

Const brands = [ ‘mclaren’ , ‘Rendault’ , ‘Ferrari’ , ‘Honda’]

Const [ , , brand [ = brands

Console.log(brand) => Ferrari

کلمه کلیدی new چیکار میکنه ؟

کلمه کلیدی new با توابع سازنده برای ساختن اشیاء به کار میرود.

به مثال زیر توجه کنید:

Function Employee(name , position , yearHired ) {

This.name = name;

This.position = position;

This.yearHiren = yearHired;

}

Const evp = new Employee(‘sina ghaffari’ , ‘softWare developer’ , 2021)

کلمه کلیدی new چهار کار را انجام میدهد.

1. Creats an empty object یک شیء خالی درست میکند
2. Assigns that empty object to this value آن شیء خالی را به this اختصاص میدهد
3. The function Will inherit from function Name.prototype این تابع از functionName.prototype به ارث می رسد.
4. Returns the this if no explict return statement is used .

در صورت عدم استفاده از دستور بازگشت صحیح ، this را برمیگرداند

در کد بالا ابتدا یک شیء خالی ایجاد میکند {} سپس مقدار this را به شئ خالی میدهد و ویژگی هایی را به آن شئ اضافه میکند ، از ان جا که ما یک دستود بازگشت صریح نداریم به طور خودکار برای ما برمیگرداند.

Object Set در جاوااسکریپت چیکار میکنه؟

یکی از ویژگی هایی که در Es6 به جاوااسکریپت اضافه شده object set هست که کاربرد اصلی اون نگهداری مجموعه ای از مقادیر به صورت یکتا هست یعنی توی set هیچ آیتم تکراری وجود ندارد.

نحوه ساختن یک آبجکت set به صورت زیر هست :

New set()

کافیه از کلمه کلیدی new استفاده کنیم. خب اینجا یک آبجکت خالی درست شد که در حال حاضر هیچ مقداری نداره . اگر هنگام ساختن این آبجکت به اون یک مقدار قابل پیمایش (iterable ) پاس بدهیم مجموعه ی یکتایی از اعضا خواهیم داشت.

Const numbers = [1,2,2,2,3]

Cosnt uniqenumbers = new set(numbers)

For(num of uniqenumbers) {

Alert(num) // 1,2,3

// 2 های تکراری حذف شدن

}

چند تا عضو داریم ؟

اگر بخواهیم تعداد اعضای آبجکت رو متوجه بشیم کافیه پراپرتی size رو فراخوانی بکنیم :

Const uniqeNumbers = new set([1,2,2,2,3]);

Alert(uniqeNumbers.size) // 3

اضافه کردن به set اگر یک آبجکت set از پیش تعریف شده داشته باشیم بخواییم مقدار به اون اضافه کنیم ، از متد add استفاده میکنیم:

Const numbers = new set()

Numbers.add(1)

Numbers.add(2)

Numbers.add(2)

Numbers.add(2)

Numbers.add(3)

Alert(numbers.size) // 3

همون طور که میبینیم مقدار های تکراری وارد این آبجکت نشدن.

آیا آبجکت چنین عضوی دارد ؟

برای این که بررسی کنیم یک آبجکت set دارای یک مقدار خواص هست یا خیر ، از متد has استفاده میکنیم:

Const numbers = new set()

Numbers.add(1)

Numbers.has(1) => true

Numbers.has(2) => false

حذف کردن از آبجکت

برای حذف کردن یک مقدار از object از متد delete استفاده میکنیم.

Const numbers = new set([1])

Numbers.delete(1)

Numbers.has(1) => false

و اگر بخواییم همه مقادیر رو پاک کنیم از متد clear استفاده میکنیم.

Const numbers = new set([1,2])

Numbers.clear()

Numbers.size() => 0

آبجکت set شبیه به یک آرایه هست اما همون طور که دیدید برای اضافه کردن آیتم به آبجکت نمی تونیم از متد هایی مثل .push() استفاده کنیم همچنین از property length برای گرفتن تعداد عضو ها . با استفاده از روش زیر میتونیم یک آبجکت set رو تبدیل کنیم به آرایه.

Const numbers = new set([1,2,2,2,3])

Const toArray = […numbers] // [1,2,3]

این یک روش رایج برای حذف کردن مقدار های تکراری از آرایه هست ، اینجا ابتدا از یک آرایه موجود ، یک آبجکت set ساختیم تا مقادیر یکتایی داشته باشیم و سپس با عملگر spread ، آبجکت رو بخش کردیم توی یک آرایه جدید این طوری آرایه ای با مقادیر یکتا داریم.

Module ماژول ها چی هستند ؟

ماژول ها در Es6 به جاوااسکریپت اضافه شدن.

یک جمله ی معروف درباره ماژول ها یا به طور کلی اصل separation of concerns (جدا سازی وابستگی ها ) وجود داره .

درست مثل یک کتاب که به فصل های مجزا تقسیم بندی میشن یک برنامه میتونه به قسمت های مجزا تقسیم بندی بشه به هر یک از این قسمت ها میگن ماژول (modul )

یک ماژول معمولا مستقل از بخش های برنامه عمل میکند توی جاوااسکریپت هر ماژول به صورت یک فایل جداگونه ذخیره میشه و می تونه بدون مشکل به یک ماژول دیگر و یا هستند برنامه اضافه یا برداشته بشه.

اعضای یک ماژول (متغیر ها و توابع ) فقط توی خود مازول قابل دسترسی هستند و global نیستن.

چرا از ماژول ها استفاده میکنیم؟

1.قابلیت نگهداری

از خوبی های استفاده از ماژول اینه که قابلیت نگهداری (maintainability ) برنامه ما بالاتر میره و بخش های مختلف رو میشه راحت تر توسعه داد .

2.استفاده مجدد

یک تکه کد از برنامه که جاهای زیادی داره تکرار میشه میتونیم جدا و به یک فایل مجزا منتقل کنیم هر وقت که خواستیم از این ویژگی استفاده کنیم کافیه فقط این فایل رو وارد کنیم.

ماژول ها رو با کلمه کلیدی import and export به هم متصل میکنن.

عملگر های spread و rest در جاوااسکریپت؟

عملگر های spread and rest که در Es6 معرفی شدن عملگر هایی هستند که با ویژگی های خاصشون بعضی مسائل برنامه نویس رو به زبان جاوااسکریپت برای ما به سادگی قابل حل کردن.

عملگر Rest :

عملگر rest بر آرگومان های ورودی تابع جاوااسکریپت عمل می کند و بر ورودی های توابع تاثیر میذاره.

بررسی Arguments :

تابعی می نویسیم که تعدادی نامعلوم ورودی بگیره و حاصل جمعش رو چاپ کنه برای حل این مسئله بدون کمک گرفتن از rest operators باید مفهوم arguments رو بررسی کنیم قبل از معرفی rest operators چطوری مسائل حل میشدن.

Function sum() {

Console.log(arguments)

// {‘0’ : 1 , ‘1’ : 2 , ‘2’ : 3 , ‘3’ , 4 , ‘4’ , 5 }

}

Sum(1,2,3,4,5)

کلید واژه arguments هر ورودی که به تابع میدیم رو برامون به شکل یک آبجکت ولی شبه آرایه در میاره. چرا میگیم شبه آرایه به این دلیل هستش که به روش خیلی ساده مثل آرایه میتونیم عناصرش رو صدا بزنیم.

Function sum() {

Console.log(arguments[0]) // 1

Console.log(arguments[1]) // 2

}

Sum(1,2,3,4,5,6)

حتی میتونیم با Array.from به آرایه تبدیلش کنیم چون که ترتیب key هاش با عناصر یک آرایه یکیه.

( جلوتر وقتی با عملگر spread آبجکت رو به آرایه تبدیل میکنیم بیشتر درک میکنید )

Function sum() {

Console.log(Array.from(arguments))

// [1,2,3,3,4,5,6]

}

Sum(1,2,3,4,5,6)

حال مسئله ای که در اول تعریف کردیم رو به کمک arguments حل میکنیم.

Function sum() {

Let sum = 0 ;

For(const iterator of Array.from(arguments)) {

Sum += iterator

}

Console.log(sum) // 21

}

Sum(1,2,3,4,5,6)

حلق for of اکثرا برای آرایه ها استفاده میشه که هر iterator به جای یکی از عناصر آرایه قرار میگیره.

اما rest operator اومد که زندگی ما رو راحت تر کنه و بتونیم ابعاد دیگری هم به تابعمون بدیم . دیگه نوبتیم باشه نوبت rest operator هستش:

نمونه کد شکل کلی استفاده از rest operator رو نشون میده:

Function sum(…numbers) {

Console.log(numbers)

// [1,2,3,4,5,6]

}

Sum(1,2,3,4,5,6)

میبینیم که نماد rest operator به صورت 3 نقطه هستش و قبل از اسم متغیری میاد که قراره آرایه ساخته شده درونش ریخته بشه ، حالا بیایید این بار با rest operator مسئله رو حل کنیم.

Function sum(…numbers) {

Let sum = 0;

For( const iterator of numbers ) {

Sum += iterator

}

Console.log(sum) // 21

}

Sum(1,2,3,4,5,6)

Rest operator زمانی به کار میاد که میخواییم ورودی هامون رو بخش بندی کنیم و مثلا بگیم آرگومان اول بره تو a و دومی بره تو b و بقیه رو تو آرایه ای به اسم numbers بریزه.

Function sum(a,b, …numbers) {

Console.log(a) //1

Console.log(b) //2

Console.log(numbers)

// [3,4,5,6]

}

Sum(1,2,3,4,5,6)

عملگر Spread ؟

استفاده از spread در آرایه ها:

این عملگر ها در آرایه ها و آبجکت ها کاربرد داره که در واقع یک سری عملیات میشه روی آرایه ها و آبجکت ها انجام داد .

مثلا میدونیم که آرایه ها و آبجکت ها Refrence type هستن برای درک بیشتر کد زیر رو مطالعه کنین.

Let array1 = [‘nodejs’ , ‘electronjs’]

Let array2 = array1

Console.log(array1 === array2)

// true

Array2[0] = ‘js’

Console.log(array1) // [‘js’ , ‘electronjs’]

Console.log(array2) // [‘js’ , ‘electronjs’]

در این برنامه میتونیم مشاهده کنیم که آرایه دوم مساوی با آرایه اول قرار گرفته به همین علت مقایسه هر دو ارایه ما true هستش یعنی این دو با هم برابرند و براری به این معناست که به یک قسمت از آدرس حافظه اشاره می کنند . پس اگر از آرایه دوم عنصر اول رو تغییر بدیم قسمتی از حافظه که هر دو بهش اشاره می کنند هم تغیری میکند در نتیجه آرایه اول هم تغیر میکند چون آدرس حافظه عنصر اولی که بهش اشاده میکند با آدرس عنصر اول آرایه دومی یکسان هستش (ویژگی Refrence type بودن )

کپی گرفتن از یک آرایه

ببینیم چطور میشه آرایه ها نسبت به هم value type کرد یعنی هر کدوم حافظه ای حافظه ای مخصوص به خودش تخصیص بده یا به اصطلاح یک کپی از آرایه تهیه کرد . یکی از روش های موجود استفاده از Spread Operator هستش.

Let array1 = [‘nodejs’ , ‘electronjs’]

Let array2 = […array1]

Console.log(array1 === array2) // false

Array2[0] = ‘js’

Console.log(array1) // [‘nodejs’ , ‘electronjs’]

Console.log(array2) // [‘js’ , ‘electronjs’]

همون طور که دیدیم spread operator هم مثل rest operator از علامت (... 3 نقطه ) استفاده میکند .

ولی تفاوت مهم شون این هستش که بعد از علامت rest یک متغیری میاد که قراره توش یک آرایه از آرگومان های ورودی ریخته بشه ولی بعد از علامت spread متغیری میاد که قراره محتویاتش خونده بشه و تو همون bracket یا curleybracket که spread ما قراره داره محتویاتش رو بریزه

( به شرطی که مجاز باشه ) برای مثال توی همین کد بالا ما میبینیم که array1 با قرار گرفتن علامت spread پشتش محتویاتش خونده میشه و توی همون bracket که قرار داره محتویاتش رو میریزه و بعدش این آرایه مساوی array2 قرار میده پس این 2 آرایه نسبت به هم value type هستن چون یک کپی از تمام مقادیر array1 درون array2 قرار داره کد زیر نیز استاندارد های دیگر استفاده از spread هستش.

Let array1 = [‘nodejs’, ‘electronjs’]

Console.log(array1) // [‘nodejs’ , ‘electronjs’]

Let array2 = [‘js’ , …array1 , ‘expressjs’]

Console.log(array2) // [‘js’ , ‘nodejs’ , ‘electronjs’ , ‘expressjs’ ]

Let array3 = […array1 , …array2]

Console.log(array3) // [‘nodejs’ , ‘electronjs’ , ‘js’ , ‘nodejs’ , ‘electronjs’ , ‘expressjs’ ]

ولی مشکلی که در این برنامه داریم این هستش که بعضی عناصر duplicate (تکراری ) هستن ، پس باید 2 تا آرایه رو merge کنیم.

کد پایین برنامه merge دو آرایه هستش ، در واقع دو ارایه رو در یک آرایه میریزیم و با new set عناصر تکراری رو حذف میکنیم و سپس spread operator پشت new set اون رو به آرایه تبدیل میکنیم و داخل array3 میریزیم .

Let array3 = […new set([…array1 , …array2])]

Console.log(array3) // [‘nodejs’ , ‘electronjs’ , ‘js’ , ‘expressjs’ ]

یکی از کاربرد های دیگر spread operator این هستش که در بعضی توابع جاوااسکریپتی قابل استفاده هستش ، بعضی از این توابع مثل Math.max() فقط پارامتری به شکل spread میگیرن . میتونین document های مختص درباره این توابع رو پیدا کنین و بخونین .

Let array = [1,2,3,4,5,6]

Let max = Math.max(array)

Consol.log(max) // NaN

max = Math.max(…array)

console.log(max) // 6

NaN به چه معناست ؟

NaN یک مشخصه شئ سراسری است ، مقدار اولیه NaN به صورت Not-a-number است.

پیچیدگی در بررسی NaN ؟!

متاسفانه در هنگام بررسی NaN دو مشکل وجود دارد :

1. Typeof NaN به صورت Number است.
2. NaN با همه مقادیر دیگه در جاوااسکریپت نابرابر است.

به مثال زیر توجه کنید:

Let myNaN = NaN ; // or let myNaN = Number.NaN

Console.log(typeof myNaN ) // numbers

Console.log(myNaN == NaN ) // false

Console.log(myNaN === NaN ) // false

اکنون سئوال این جاست که چگونه میتوانیم بررسی کنیم یک مقدار NaN داریم که عملیات ریاضی را مختل کرده است .

بررسی NaN از طریق برابری با خود:

چنان که بیشتر اشاره کردیم. تنها و تنها NaN است که با خودش برابر نیست ، در js بهترین روش برای بررسی NaN بودن یک مقدار بررسی برابری آن با خود ( self-equality ) است.

از آن جا که NaN با خودش برابر نیست NaN != NaN همواره مقدار صحیح تولید میکند . البته این روش تست مقدار NaN در کد همواره خوانا نیست ، بنابراین استفاده از یک کامنت یا ایجاد یک تابع پوششی ایده خوبی به نظر می رسد.

Let variable = NaN // or let variable = Number.NaN

//ckeck for NaN by checking for self-equality

Console.log(variable != variable) // true

Console.log(variable != variable) // true

// NaN is the only value this is not equal itself

Let is NaN = (mybeNaN) => mybeNaN != mybeNaN ;

Console.log(isNaN(variable)) // true

نکاتی در مورد Number.isNaN() :

جاوااسکریپت مدرن یک پیاده سازی برای بررسی NaN بودن مقدار به نام Number.isNaN() دارد که مطابق انتظار عمل میکند ، به مثال زیر توجه کنید:

Let mybeNaN = NaN

Console.log(Number.isNaN(mybeNaN)) // true

mybeNaN = 3

console.log(Number.isNaN(mybeNaN)) // false

توجه داشته باشید که Number.isNaN() از تابع isNaN() متفاوت است . isNaN() یک پیاده سازی قدیمی تر است که هدف اصلی آن بررسی این است که آیا نک مقدار می تواند به صورت عددی در آید یا ند .

در صورتی که مقدار در حال حاضر NaN باشد. یا اگر قرار باشد پس از کاهش یافتن به عدد ، به صورت NaN در آید، isNaN() مقدار صحیح بازگشت میدهد . به بیان دیگر اگر مقداری دریافت کند که میتواند به یک عدد کاهش یابد ، isNaN() مقدار true بازگشت خواهد داد.

Number.isNaN() تنها در صورتی مقدار true بازگشت می دهد که مقدار در حال حاضر NaN باشد.

isNaN(‘hello sina’) // true

Number.isNaN(‘hello world’) // true

آرایه ها Array :

آرایه ها نوعی متغیر هستند که میتوانند چندین مقدار را در خود ذخیره کنند.

Var numbers = [1,2,3,4,5,6,7,8,9,10]

به هر مقدار در آرایه element می گویند ، که هر element دارای جایگاهی در درون آرایه است که به آن جایگاه index گفته می شود.

در مثال بالا هر کدام از مقادیر 1 تا 10 یک element هستند.

مقادیر داخل یک آرایه ی جاوااسکریپت می توانند از هر نوعی باشند، Object , Boolean , String و حتی خود آرایه ها هم می توانند عضوی آرایه دیگری باشند.

چطور ببینیم یه مقدار از نوع آرایه هست ؟

توی جاوااسکریپت اگر بخواییم بررسی کنیم که یه مقدار از نوع آرایه هست یا خیر می تونیم از typeof استفاده کنیم .

Const numbers = [ ]

Alert(typeof number ) // object

چون آرایه ها توی جاوااسکریپت نوعی آبجکت هستن و typeof یک آرایه برابر با object هست.

برای بررسی این که آیا این مقدار از نوع آرایه هست ، از Array.isArray() استفاده میکنیم. مثل کد زیر :

Const numbers = [ ]

Alert.(Array.isArray(numbers)) // true

چند روش برای پیمایش روی آرایه ها توی جاوااسکریپت وجود دارد؟

7 روش برای پیمایش روی آرایه ها در جاوااسکریپت وجود دارد:

1. For / while
2. forEach
3. map
4. reduce
5. filter
6. every
7. some
8. حلقه های کلاسیک : استفاده از for and while

این روش از قدیمی ترین روش هاست و اجازه پیمایش در تمام خانه آرایه رو به ما میده.

Let I = 0 // index

Const array = [1,2,3]

While(I < Array.length ) {

Console.log(array[i])

I++

}

حلقه (for) کلاسیک ترین راه ممکن

Const array = [1,2,3]

For(let I = 0 ; I < array.lenth ; i++ ) {

Console.log(array[i])

}

1. استفاده از روش forEach() (یه دوری بزنیم)

این متد در Es6 معرفی شد و مثل روش های بالا از الگوریتم حلقه برای پیمایش استفاده میکنند.

Const array = [1,2,3]

Array.forEach(function(current-value , index , array ) {

Console.log(‘At index ${index} in Array ${array} ${current-value} } // undefined

مشکلی که ممکنه در کار این متد بهش بر خورد کنیم اینه که بعد از استفاده دیگر چیزی رو return نمیکنه و به همین دلیل دیگر نمیشه در ادامه دو باره از آرایه استفاده کرد.

برای حل این مشکل یه سر به روش بعدی بزنید:

1. استفاده از نقشه map()

روش قبلی خوب و منطقی بود ولی در یک سری استفاده خاص ممکنه جواب نده.

این متد هم در Es6 متولد شده و کارش اینه که یک تابع رو روی تک تک اعضای یک آرایه اعمال کند.

در مثال زیر ما تابع func که یک تابع Arrow Anonymous هست رو با استفاده از map به آرایه array نسبت دادیم و میتونیم در یک متغیر جدید نتایج مورد نظرمون رو دریافت کنیم:

Var array = [1,4,9,16]

Const func = x => x \* 2;

Const map = array(func)

Console.log(map)

// [2,8,18,32]

1. استفاده از reduce ( کمکش کن)

این متد یک تابع رو به یک نگهدارنده (accumulator ) و تک تک خانه های آرایه از چپ به راست اعمال میکنه تا به یک مقدار برسه.

Const array = [1,2,3]

Const sum = (x , y) => x + y

Const array-sum = array.reduce(sum , 0 )

Console.log(‘The some of Array : ${array} is ${array-sum}`)

// The some of Array : 1,2,3 is 6

1. استفاده از filter (فیلترش کن)

خیلی ساده ! یه شرط بده بهش ، هر کدوم از خونه ها با شرط جور اومد میاد بیرون.

Const array = [1,2,3,4]

Const event = x => x % 2 === 0;

// صفر به معنای عدد زوج است و یک به معنای عدد فرد

Const event-array = array.filter(event)

Console.log(`Even numbers in array ${array} : ${even-array}` )

// Even numbers in array 1,2,3,4 : 2,4

1. استفاده از روش every (ببین همشون تو یه شرط مشترکن)

مواظب باشید این متد روی آرایه خالی استفاده نکنید که تو هر شرایطی true برمیگردونه.

Function isBelowThreshold(currentValue) {

Return currentValue < 40;

}

Var array1 = [1,30,39,29,10,13]

Console.log(array1.every(isBelowThreshold)) // true

1. روش آخر استفاده از some میباشد ( حداقل یکیشون که این شرط رو دارن ؟ نه؟! )

Const array = [ 1,2,3,9,5,6,4 ]

Const over-seven = x => x > 7

If(array.some(over-seven)) {

Console.log(‘yaft-shod’) // 9

} else {

Console.log(‘yaft-nashod’)

}

Ajax چیست ؟

اگر به طرز کار صفحات وب توجه کنید متوجه می شوید که برای نمایش هر مطلب یا صفحه ی جدید ، صفحات وب مجددا بارگذاری (refresh ) میشن تا محتوای جدید را نمایش دهند.

تکنولوژی Ajax ایجاد شده تا بدون نیاز به refresh کل صفحه ، بخشی از اطلاعات صفحه بارگذاری شود ، در واقع این تکنولوژی به کاربر اجازه می دهد بدون بارگذاری مجدد صفحه ، داده هایی را به صفحه ارسال کند تا در آن جا پردازش شوند، سپس اطلاعات را دریافت و در صفحه وب به نمایش در آورد.

نحوه کار Ajax چگونه است؟

این کار طی جند مرحله و ایجاد یک event ( مثلا با کلیک روی یک دکمه یا لینک ) آغاز میشود .

مرورگر به کمک جاوااسکریپت یک شئ از XMLhttprequest میسازد و به کمک اینترنت آن را به سرور ارسال میکند.

در سرور درخواست دریافت و پردازش میشود سپس نتیجه آن به مرورگر بازگردانده میشود مرورگر پاسخ را به کمک جاوااسکریپت در یافت و نمایش میدهد.

مزایا و معایب Ajax :

مهم ترین مزایای Ajax

1. خواندن اطلاعات از سرور بعد از لود شدن صفحه
2. بروز رسانی یک صفحه بدون نیاز به بارگذاری مجدد آن
3. ارسال داده به سرور در پس زمینه ی سایت

مهم ترین معایب Ajax

1. استفاده از درخواست Xmlhttp که ممکن است توسط برخی از مرورگر ها پشتیبانی نشود
2. اجرا نشد دستورات Ajax در صورتی که جاوااسکریپت مرورگر توسط کاربر غیر فعال شده باشد
3. دشواری بیشتر در اشکال زدایی و تست برنامه های Ajax

برای مدیریت کردن عملیات ناهمگام چه راه هایی رو میشناسید ؟

توی خود جاوااسکریپت با 3 روش میتونیم این کار رو انجام بدیم :

1. Callback function
2. Promise
3. Async / await

البته کتابخونه هایی اختصاصی مثل co , q , bluebird , async.js هم وجود دارن.

مقایسه Synchronous and Asynchronous ؟

در جاوااسکریپت در حالت پیش فرض کد ها به صورت asynchronous اجرا میشوند.

اگر بخواییم تعریف عامیانه ای از synchrounous داشته باشیم :

حالت اول : در این حالت در هر لحظه از زمان یک خط کد ما اجرا خواهد شد اگر محیط اپلیکیشن خود را یک رستوران در نظر بگیریم و خود اپلیکیشین را گارسون و کلاینت را مشتری به این صورت خواهد بود که اول گارسون به میز شماره یک می رود سفارش را گرفته و او منتظر می نشیند تا سفارش میز شماره یک آماده شود و بعد از تحویل سفارش سراغ میز شماره دومی می رود خب چنین راهکاری قطعا منطقی نخواهد بود.

کد های ما هم در حالت پیش فرض این شکلی کار میکند ( خط به خط تا انتها )

بررسی Asynchronous بهتر بود اگر گارسون سراغ چند میز که آماده سفارش دادن هستند برود و سفارش را دریافت کند و به کیترینگ تحویل بدهد: هر زمان که سفارش میزی آماده شد آن را تحویل می دهد ، به این صورت وقفه ای در کارهای اجرایی رستوران پیش نخواهد آمد.

حالت دوم یعنی Asynch و در عمل به این صورت است که کارهایی که انجام دادن آنها نیازمند صرف زمان هستند باید به صورت Asynch نوشته شوند و در این حالت اپلیکیشین ما کارش متوقف نمیشود تا یک سفارش به مرحله پایانی برسد و بعد سراغ سفارش بعدی برود.

Asynch چه زمانی استفاده میشود ؟

وقتی قرار است یک اتفاقی رخ دهد که نیازمند صرف زمان است

وقتی قرار است یک فایل دانلود و یا آپلود شود وقتی از توابع asynch استفاده می شود اپ ها کار خودشون رو انجام میدهد تا مرحله آپلود یا دانلود به اتمام برسد.

متد های push , pop , shift , unshift

Const numbers = [1,2,3,4,5]

متد push :

به آخر آرایه یک value جدید اضافه میکنه

Numbers.push(6) // [1,2,3,4,5,6]

با استفاده از spread operator :

Const numbers = [1,2,3,4,5]

Const newNumbers = [6,7,8]

Numbers.push(…newNumbers) // [1,2,3,4,5,6,7,8]

متد pop :

از آخر ارایه به مقدار یک value کم کن

Numbers.pop() // [1,2,3,4]

متد shift :

از اول آرایه به مقدار یک value جدید اضافه میکند.

Numbers.unshift(0) // [0,1,2,3,4,5]

متد unshift :

به اول آرایه یک value جدید اضافه میکند.

Numbers.shift(0) // [0,1,2,3,4,5]

متد splice :

متد splice در آرایه کاربرد داره و کارش این هست که یک مقداری رو از داخل آرایه حذف کنه و در صورت نیاز جای اون مقدار حذف شده مقدار جدید قرار بده ، و این که می تواند هیچ مقداری رو حذف نکند داخل آرایه و فقط در اندیس دلخواه مقدار اضافه کند به کد پایین دقت کنید

Const arr = [0,1,2,3,4]

Const valueSplice = arr.splice( first arguman , second arguman)

آرگومان اول => در این پارامتر مشخص میکنیم این متد از کدام اندیس شروع به حذف کردن کند .

آرگومان دوم => در این پارامتر مشخص می کنیم که چه تعداد از مقادیر از مقادیر درون آرایه حذف شود .

مثال :

حذف صفر از اول آرایه:

Const arr = [0,1,2,3,4]

Const valueSplice = arr.splice(0,1)

\*\*\* از اندیس صفر شروع کن و یدونه حذف کن

Console.log(arr) // [1,2,3,4]

**React**

**React چیست ؟**

**یک کتابخونه متن باز جاوااسکریپتی برای ساختن رابط کاربری**

**Component ؟**

**هر بخش از صفحه وب که قابلیت مستقل شدن رو داره میتونیم به صورت جداگونه درون یک Component توسعه اش بدیم.**

**اجزای سازنده ی اپلیکیشین های ری اکت هستند می تونیم هر قسمتی رو به صورت مستقل بنویسیم و هر جا که لازم بود ازش استفاده کنیم .**

**JSX چیست ؟**

**Javascript xml**

**Syntax هست که درون ری اکت معرفی شد تا ما بتونیم از Html به راحتی درون ری اکت استفاده کنیم.**

**Virtual DoM ؟**

**تشخیص میده که کدوم Component نیاز به رندر شدن داره اونو آپدیت می کنه و رندر میکنه و این باعث افزایش پرفورمنس و میزان کاهش مصرفی معماری میشه.**

**Pwa => progressive web application**

**چه چیز هایی به Es6 اضافه شدن ؟**

**Let – const – arrow function – export – import – class – set**

**تفاوت html , jsx ؟**

**در jsx ، element هامون حتما باید wrap بشن تو یه tag**

**و به جای for در html باید از Htmlfor در jsx استفاده کنیم و به جای class از className استفاده میکنیم.**

**React props system ؟**

**سیستمی برای ارسال data از کامپوننت والد به کامپوننت فرزند.**

**Functional component (state less comopenent) => good for simple component**

**Class component (state full component) => good for everything else**

**\*\* مدیریت وضعیت یا مدیریت state هم در این component وجود داره .**

**و البته در update های جدید این امکانات با Hooks و context در functional component نیز در دسترس هست.**

**State ؟**

**یه آپدیت جاوااسکریپتی که داده های داینامیک رو ذخیره میکنه .**

**State ها فقط درون همون کامپوننت وجود دارن و امکان ارسال آن به component های دیگر نیست .**

**Component ها با استفاده از این داده های dynamic میتونن داینامیک شن و هر موقع نیاز باشن render بشن یا تغییر کنن.**

**انواع Event ها :**

1. **onClick**
2. **onMouseOver**
3. **onSubmit**
4. **onkeydown**
5. **onkeyup**

**ajax : ارسال درخواست به API با استفاده از AJAX و http درون مرورگر ها انجام میشه .**

Redux

Redux چیست ؟

یه کتابخانه مدیریت state

State چیست ؟

داده هایی که درون component ها ذخیره میشن و component ها با استفاده از این داده ها خودشون رو اپدیت میکنن و بروزرسانی میکنن.

همچنین redux کمک می کنه پروژه های پیچیده ری اکت رو آسون تر بسازیم.

\*\* state ها در redux درون store ساخته میشن . و هر پروژه ری اکت یه store نمی تونه داشته باشه .

Redux چطور کار میکنه ؟

به وسیله type ها در اکشن میفهمیم که کارشون چیه .

Action های ما در قالب آبجکت قرار میگیره و این آبجکت ها پاس داده میشن به Reducer ها که همون نقش Listener رو باز میکنه.

{

Type : ‘INCREMENT’

}

{

Type : ‘DECREMENT’

}

CSR AND SSR

SSR => Server-side rendering

CSR => Client-side rendering

SSR => یعنی پردازش ها و ساخته شدن element ها در سمت سرور اتفاق میافتد

CSR => و در csr این ها بر روی مرورگر کاربر اجرا میشن.

1. Ssr در react.js با ssr که در گذشته بود متفاوت است :

Wordpress , php(Laravel , …) , asp.net , … !== React ssr

1. رندر سمت سرور react.js به کمک node.js انجام میشود و مستقل از سرویس دهنده پروژه (سمت بک اند) است.
2. رندر سمت سرور (ssr ) را هم به صورت customize شده و هم با ابزار هایی مانند Next.js می توان انجام داد که هر کدام نقاط قوت و ضعف خود را دارند.
3. رندر سمت کلاینت (csr) را عموما به کمک ابزار create-react-app یا به اختصار csr انجام میدهند.

Csr :

در CSR وقتی کاربر وارد صفحه میشه لود اولیه اتفاق میافته که خزنده مرورگر فقط قادر به دیدن این مرحله می باشد که فایل HTML از سرور فرانت اند گرفته می شه ولی خزنده قادر به خوندن اون نیستن.

\*\* یکسان بودن فایل index.html برای تمام صفحات و دریافت اطلاعات صفر بعد از پایدار شدن صفحه (document ready ) ضد اصول سئو است.

\*\* لود اولیه سایت در این روش بخاطر چند مرحله ای بودن نمایش صفحه همراه با داده حقیقی کندتر خواهد بود.

\*\* نداشتن قدرت های کنترلی که در ساختار هایی که بک اند اساس توسعه است ( مانند کنترل درخواست ها ، دسترسی به فایل ها و ... )

Ssr :

در ssr سرور فرانت اندمون مستقیم با سرور بک اند در ارتباطه ، نه مرورگر کاربر و سرور بک اندمون که در csr اتفاق میافتاد .

در معماری csr پردازش ها و دریافت اطلاعات در سمت کلاینت و به کمک مرورگر کاربر انجام میشود ، ابزار create-react-app مشهور ترین ابزار تولید وب اپلیکیشین های csr ری اکت است.

در معماری ssr بخش اصلی از پردازش ها در سمت سرور و با هدف رفع نقض های معماری csr انجام میشود.

تفاوتی که بین csr , ssr هست اینه که در ssr کرولر ما (خزنده ) دیگر یه index ثابت برای تمام root ها نمیگیره بلکه component که react رندر کرده و data واقعی توش هست رو میگیره.

Unit Test

در unit test ما کد های پروژه مون رو به بخش های کوچیک تقسیم میکنیم و براشون تست مینویسیم .

TDD => Test Driven Development

BDD => Behavioral Driven Development

BDD میگه رفتار این کد چیه خیلی براش مهمه که مستند سازیاش انجام بشه .

برای تست از ابزاری به اسم mocha استفاده میکنیم :

Npm install –save-dev mocha

برای این بتونیم از این پکیج استفاده کنیم حتما باید فایل package.json رو در APP مون داشته باشیم که به این معناست صورت نصب می کنیم .

Npm init

و در قسمت script ، تست رو به این صورت تغیر میدیم به mocha

“script” : {

“test “ : “mocha”

}

و برای تست کردن در cmd مینویسیم npm test که برای اجرا شدن این حتما باید یه directory به اسم test باز کنیم و فایل های تست مون رو در اون قرار بدیم .

Test -> first-test.js ( file name )

ساختار کلی بدنه تست به این صورته

Describe(‘app <test name>’ , function() {

It(‘my first test < description test >’ , function () {

.

.

.

})

})