جلسه اول – معرفی جاوا اسکریپت

آنچه خواهیم آموخت

در این جلسه به این می پردازیم که جاوا اسکریپت چیست؟ و به چه دردی می خورد؟ و در نهایت جاوا اسکریپت به چه دردی می خورد؟

درباره ی جاوا اسکریپت

جاوا اسکریپت در سال 1995 اختراع شد، برای Netscape 2 توسعه داده شده بود. در گذشته و اوایل هدف جاوا اسکریپت پویا (Dynamic) کردن صفحات وب بود، اما مشکلات زیادی داشت و زیاد مورد استقبال قرار نگرفته بود. تا سال 2009 جاوا اسکریپت پستی و بلندی های زیادی داشت و خیلی از توسعه دهنده ها و وبسایت ها به عنوان جایگزین داشتند به فلش روی می آوردند و از آن استفاده می کردند اما در سال 2009 جاوا اسکریپت تحولات زیادی داشت و Node Js هم به میدان آمد.

کل جاوا اسکریپت به قبل و بعد سال 2009 تقسیم میشه، در سال 2009 جاوا اسکریپت به یک زبان Cross Platform تبدیل شد که یعنی مستقل از نوع سیستم عامل روی سیستم عامل های مختلف قابل اجرا است.

Node Js چیست؟

همانگونه که گفتیم جاوا اسکریپت در ابتدا با هدف پویا سازی صفحات وب آمده بود اما با آمدن Node Js جاوا اسکریپت دیگر قادر شد سمت سرور شبیه به زبان های سمت سرور اجرا شود. پس جاوا اسکریپت حالا برای هردو Frontend و Backend می تواند استفاده شود.

اصلا راه نداره کسی بخواهد توسعه وب کار کند و جاوا اسکریپت را یاد نگیرد!

جلسه دوم – تفاوت EcmaScript و JavaScript

مرجع کامل مستندات و یادگیری جاوا اسکریپت

وب سایت دولوپر موزیلا یک مرجع جامع و کامل هست برای یادگیری جاوا اسکریپت که پیشنهاد می شود حتما از آن استفاده شود برای یادگیری بهتر جاوا اسکریپت. <https://developer.mozilla.org>

همانگونه که جلسه قبل گفته شد برای اولین بار Netscape جاوا اسکریپت را در سال 1995 ارائه کرد، در سال 1997 Netscape و Ecma با همدیگر همکاری کردند تا یک استاندارد جهانی برای جاوا اسکریپت به وجود بیاورند تا از پراکنده شدن جاوا اسکریپت جلوگیری کنند و هرکسی آن را به میل خود توسعه ندهد.

این استاندارد EcmaScript نام گرفت یا دقیق تر EcmaScript-262 و از آن پس هرچند وقت یکبار این استاندار آپدیت می شود و موارد جدید به آن اضافه می شود و پس از مدتی مرورگر ها آن موارد را اعمال می کنند اما ممکن است اینکار صورت نگیرد و ویژگی ایی اضافه نگردد.

بعد از یادگیری جاوا اسکریپت باید EcmaScript 6 یه به اختصار ES6 رو یاد بگیریم. در ES6 ویژگی های مهمی ارائه شد و تحولات بزرگی به وجود آمد که یادگیری ES6 را ضروری کرده است.

پس در دروه پیش رو ES5 رو یاد می گیریم و قدم بعدی ES6 است.

جلسه سوم – اجرای کد های جاوا اسکریپت

پیش نیاز یادگیری جاوا اسکریپت

برای یادگیری جاوا اسکریپت باید با HTML و CSS حداقل در سطح ابتدایی آشنا باشیم. باید یک ویرایشگر کد داشته باشیم که در این دوره از VS Code استفاده می شود.

اجرای اولین پروژه جاوا اسکریپتی

ابتدا یک دایرکتوری با نام دلخواه میسازیم در اینجا ما آن را js می نامیم، سپس روی آن کلیک راست کرده و روی open in vs code کلیک می کنیم تا در محیط ویژوال استدیو کد باز شود.

در دایکتوری ذکر شده یک فایل با نام index.html میسازیم و در آن ساختار اولیه یک سند html را می نویسیم.

سپس جاوا اسکریپت را به پروژه خود اضافه می کنیم که دو روش برای این کار وجود دارد.

روش اول:

نوشتن کد های جاوا اسکریپت در میان تگ <script> … </script>

تگ فوق را می توان هم در داخل تگ <head> قرار داد و هم در داخل تگ <body> و در تگ <body> می تواند در هرجایی قرار گیرد و اجرا می شود اما پیشنهاد می شود که حتما در انتهایی ترین قسمت تگ <body> قرار گیرد یعنی درست قبل از </body> تا کد های جاوا اسکریپت پس از بارگذاری کامل تگ های html اجرا شوند تا اگر در بخشی از جاوا اسکریپت با DOM کار کرده ایم به مشکل برنخوریم.

دستور console.log() هرچیزی که به آن پاس داده شود را در پنجره console مرورگر نمایش می دهد.

برای باز کردن console در مرورگر در صفحه وب کلیک راست می کنیم و گزینه Inspect را می زنیم و سپس تب console را انتخاب می کنیم.

روش دوم:

کدهای جاوا اسکریپت را می توان در یک سند جداگانه نوشت تا از شلوغ شدن فایل html جلوگیری شود.

برای اینکار یک فایل با پسوند .js مسیازیم و کد های جاوا اسکریپت خود را در آن می نویسیم، برای فراخوانی آن در سند html یک همانند روش اول تگ <script></script> را قرار می دهیم با این تفاوت که اینبار به جای نوشتن کد ها در میان آن یک attribute با عنوان src برای آن تعریف می کنیم و آن را برابر با آدرس فایل جاوا اسکریپتی که ساخته ایم قرار می دهیم.

در هنگام نوشتن آدرس در دنیای کامپیوتر ./ به معنای دایرکتوری است که در آن قرار داریم و ../ به یک دایرکتوری پیش تر اشاره می کند که می توان آن را متوالی به کار برد مثلا، ../../ به دو دایرکتوری پیش تر اشاره می کند.

اگر در میان تگ <script src=””></script> کد جاوا اسکریپت بنویسیم دیگر اجرا نخواهد شد. اما می توان هر تعداد که مورد نیاز است تگ <script> در یک سند تعریف کرد و هر کدام می توانند به سند دیگر اشاره کنند و یا اینکه حاوی کد های جاوا اسکریپت باشند.

در جاوا اسکریپت ترتیب نوشتن کد ها مهم است و به ترتیب نوشته شده کدها اجرا می شوند. همچنین در تب console در مرورگر می توان مستقیم کد جاوا اسکریپت نوشت.

جلسه چهارم – متغیر ها و انواع داده

آنچه خواهیم آموخت

یکی از اصلی ترین و پایه ای ترین مباحث هر زبان برنامه نویسی مبحث متغیر ها است چون بدون متغییر تقریباً برنامه نویسی دیگر بی معنی است. در این جلسه به این می پردازیم که در جاوا اسکریپت چگونه متغیر تعریف می کنیم و چه نوع داده های را می توان در متغیر های ساخته شده قرار داد.

متغیر

متغیر قابلیت این را دارد که مقداری را با نوع مشخصی در خود ذخیره کند تا در بخش های مختلف برنامه از آن استفاده کنیم. در زیر نحوه ایجاد کردن متغیر را می نویسیم:

Var variableName = variableValue;

نوشتن نام متغییر اصول خاصی دارد مثلا در ابتدای نام نباید عدد قرار دهیم ویا علامت و کاراکتر های خاص همانند #، %، ! و ... را نمی تواند در نام متغییر نوشت و حتی نمی شود – را در آن بکار ببریم.

برای نمایش مقدار یک متغییر نام آن را به console.log() پاس می دهیم، نام را بدون “ و ‘ می نویسیم.

نکته مهم: در جاوا اسکریپت می توان ; را در انتهای هر بلاک کد قرار داد و یا نه، جاوا اسکریپت حساس نیست.

انواع داده ها

1. عدد، number
2. اعداد اعشاری، float یا double
3. رشته، string
4. Boolean (true or false)
5. Null
6. Undefined
7. NaN

برای نوشتن مقدار متغیر از نوع string باید رشته را درون ‘ ‘ یا “ “ نوشت.

به صورت پیش فرض مقدار یک متغیر undefined است.

NaN یک عدد است که عدد نیست در محاسبات ریاضی ممکن است برگشت داده شود.

اگر کلمه کلیدی typeof رو قبل از نام متغیر قرار دهیم نوع آن را بر می گرداند. مثال:

typeof num; مقدار متغیر را بر می گرداند

دو نوع داده پر کاربرد دیگر داریم، array (list) و object که به آن ها خواهیم پرداخت.

تعریف آرایه

برای تعریف آرایه identifier متغیر را برابر با [ ] قرار می دهیم و اعضای آرایه را درون آن می نویسیم و اعضا را با , از همدیگر جدا می کنیم.

برای دسترسی به آیتم نام آرایه را مینویسیم و مقابل آن [ ] را قرار می دهیم و درونش شماره ایندکس آرایه را می نویسیم که از 0 شروع می شود همانند اکثر زبان های برنامه نویسی. مثلا:

var arr = [1,2,3,4,5,6,7,8]

console.log(arr[0]) این کد اولین عضو آرایه را نمایش می دهد

برای تعریف شئ (آبجکت) در جاوا اسکریپت به صورت زیر عمل می کنیم:

var obj = {

key: value,

…

}

در ES5 متغیر را با کلمه کلیدی var تعریف میکنیم اما در ES6 از کلمه کلیدی دیگری استفاده می کنیم که مشکلات var را ندارد. این کلمه کلیدی جدید let می باشد که توصیه میشود از آن استفاده شود. مثلا اگر از var استفاده کنیم می توانیم دوبار متغیری را تعریف کنیم در طول برنامه که یک ایراد است.

در جاوا اسکریپت نام متغیر ها Case Sensitive هستند یعنی به بزرگی و کوچکی حروف حساس است.

نوشتن var قبل از تعریف متغیر ضروری نیست اما ممکن است پیامد هایی داشته باشد پس نوشتن آن توصیه میشود، همچنین این کلمه کلیدی تنها در تعریف اولیه نوشته می شود و در استفاده از متغیر آن را نمی نویسیم.

جلسه پنجم – عملگر ها و نحوه استفاده از آن ها – بخش اول

پیشنهاد میشود از وبسایت موزیلا برای یادگیری بیشتر در این مبحث استفاده شود.

در مدرسه عملگر های ابتدایی مثل + - × ÷ را آموخته ایم.

اگر بخواهیم مقدار متغیری با عددی جمع گردد بصورت زیر عمل می کنیم:

a = a + 1

اگر مقدار فوق را مجددا درون a نریزم مقدار a همان قبلی باقی خواهد ماند و عملیات فوق بی تاثیر است.

نتیجه تقسیم هر عددی بر 0 مقدار Infinity است که به معنای بی نهایت است.

اگر قرار است حاصل جمع یا هر یک از 4 عملگر اصلی مجدد داخل خود متغیر ریخته شود یک راه برای کوتاه کردن کد وجود دارد:

به جای نوشتن a = a + 3 می نویسیم a += 3، هر دو همانند هم عمل می کنند اما دومی کوتاه تر است.

همچنین اگر فقط یک واحد اضافه می شود حتی می توان کد را کوتاه تر کرد به جای a += 1 می نویسیم a++ البته می توان ++ را قبل از متغیر نیز قرار داد که تفاوت آن در زیر بیان می گردد.

اگر قبل نام قرار گیرد ابتدا مقدار متغیر یک واحد اضافه می شود و سپس متغیر نمایش داده میشود اما اگر پس از آن قرار گیرد ابتدا متغیر نمایش داده می شود سپس یک واحد اضافه می گردد.

داستان فوق برای -- نیز وجود دارد مثلا: a-- یا --a

اگر رشته و عدد را باهم جمع کنیم عدد را نیز رشته در نظر می گیرد و هر دو به هم می چسبند و رشته جدید را تشکیل می دهند. حتی اگر یک طرف عدد و طرف دیگر Boolean باشد باز نیز همانند رشته به هم می چسبند. حتی آرایه را نیز همانند رشته در نظر می گیرد و به رشته دیگر می چسباند اما این فقط برای عملگر + صدق می کند و بقیه عملگر ها فقط اعمال ریاضی را انجام می دهند و اگر با مثلا عددی را از رشته ایی کسر کنیم مقدار NaN برگشت داده خواهد شد.

جلسه ششم – عملگر ها و نحوه استفاده از آن ها – بخش دوم

عملگر های مقایسه ای

همیشه نتیجه عملگر های مقایسه true یا false می باشد.

a == b تساوی دو متغیر a و b را بررسی می کند اگر برابر باشند true برگشت داده خواهد شد در غیر اینصورت false برگشت داده می شود.

نتیجه ‘5’ == 5 true می باشد اما نتیجه ‘5’ === 5 false است چراکه === علاوه بر برابر بودن دو مقدار برابر بودن نوع داده ی آن ها را نیز بررسی می کند.

نتیجه 1 == true صحیح یا true می باشد.

4 عملگر مقایسه ایی پر کاربر دیگر نیز داریم:

a > b اگر a از b بزرگتر باشد مقدار true برگشت داده خواهد شد.

a < b اگر a از b کوچکتر باشد مقدار true برگشت داده خواهد شد.

a >= b اگر a از b بزرگتر باشد ویا با آن برابر باشد مقدار true برگشت داده خواهد شد.

a <= b اگر a از b کوچکتر باشد ویا با آن برابر باشد مقدار true برگشت داده خواهد شد.

اگر دوتا رشته با هم برابر باشند true برگشت داده می شود اما اگر حتی در یک کاراکتر اختلاف داشته باشند مقدار false برگشت داده می شود.

عملگر ! نتیجه را معکوس می کند. مثلا !false برابر است با true. این عملگر می تواند قبل از عملگر های مقایسه ای قرار گیرد مثلا != یعنی اگر برابر نباشند.

‘true’ = true نتیجه false است.

‘1’ == true نتیجه true است اما اگر بجز 1 مقدار رشته چیز دیگری باشد false بر می گردد.

null == undefined نتیجه true هست.

null == NaN نتیجه false است.

0 == null نتیجه false است.

0 == NaN نتیجه false است.

اولویت عملگر ها

اولیت در عملگر ها همانند آنچه است که در ریاضیات آموخته ایم، اولویت به ضرب است سپس جمع و همچنین اولویت با داخل پرانتز است.

Jsbin.com یک وبسایت خیلی خوب است برای نوشتن جاوا اسکریپت آنلاین و مشاهده خروجی.

متد .toString() متغیری که دریافت می کند را به رشته تبدیل یا caste می کند.

متد .toFixed() به عنوان ورودی عدد می پذیرد و به تعداد ورودی اعداد اعشاری را بر می گرداند البته رقم اعشار را بسته به شرایط گرد نیز می کند.

جلسه هفتم – آشنایی با توابع

ممکن است چندین خط کد بنویسیم و این کد ها در جاهای دیگری نیز کاربرد داشته باشند بجای کپی پیست کردن و تکرار کردن کد ها آن ها را درون تابع می ریزم و هرجا که لازم باشد تابع را فراخوانی می کنیم.

همانند آنچه که در مدرسه آموخته ایم توابع ورودی می پذیرند سپس پردازش انجام میدهند و نهایتاً خروجی را بر می گردانند. البته بسته به شرایط ممکن است ورودی یا خروجی نداشته باشد.

نحوه تعریف کردن تابع:

function funName() {

}

نوشتن نام توابع نیز از قواعد نوشتن نام متغیر پیروی می کند.

عنوان ورودی ها را درون () می نویسیم تا در هنگام فراخوانی وقتی به تابع پاس داده می شوند آن ها را به ترتیب عنوان های نوشته شده در () هنگام تعریف تابع دریافت کنیم.

تمام کد های تابع درون {} قرار می گیرند، کد هایی که پردازش را انجام می دهند.

برای فراخوانی تابع نام آن را می نویسیم و جلوی آن () را قرار می دهیم و اگر پارامتر یا ورودی بپذیرد آن ها را درون () می نویسیم و هر پارامتر را با , جدا می کنیم.

در ES6 قابلیت تعریف متد در پیش فرض وجود دارد اما در ES5 این امکان وجود ندارد و باید مثل زیر این مورد را پیاده سازی کنیم.

Function echo(a) {

var b = a || 1

}

در کد فوق عملگر || اگر مقدار a وجود نداشته باشد متغیر b برابر با 1 میشود اما اگر a وجود داشته باشد در آن ریخته میشود.

Parameter به مقادیری میگویند که در () هنگام تعریف تابع نوشته می شوند اما Argument یا آرگومان به مقادیری گفته میشود که هنگام فراخوانی تابعی درون () قرار می گیرند.

خروجی تابع با کلمه نوشتن کلمه کلیدی return در جلوی آن برگشت داده می شود.

Function echo(a) {

return a

}

مقدار a برگشت داده می شود در تابع فوق.

جلسه هشتم – محدوده متغیر ها در جاوا اسکریپت

دو نوع محدوده داریم، Local Scop یا محدوده محلی و Global Scope یا محدوده سراسری

در داخل بلاک تابع یعنی درون { } متغییر های سراسری یعنی متغییر هایی که خارج از تابع تعریف شده اند در دسترس هستند. اما متغیر هایی که درون بلاک تابع نوشته می شوند در خارج آن در دسترس نیستند.

اگر متغیر سراسری ایی را درون یک تابع بازنویسی کنیم دیگر متغیر محلی در دسترس است و متغیر سراسری نادیده گرفته می شود. اگر مقدار متغیر سراسری ایی را درون تابع آپدیت کنیم در اصل متغیر نیز تغییر می کند.

نکته: اگر در بالای یک سند جاوا اسکریپت عبارت ‘use strict’ را بنویسیم قوانین سخت گیرانه تر میشوند مثلا اگر با استفاده از var یک متغیر با identifier برابر بار دیگر تعریف کنیم این بار خطا گرفته می شود.

جلسه نهم – دستورات شرطی – بخش اول

برای تعریف شرط درون کد خود به صورت زیر عمل می کنیم:

If (condition) {

}

در کد فوق اگر شرط داخل پرانتز true باشد کد درون بلاک شرط یا { } اجرا می گردد در غیر اینصورت نادیده گرفته میشود.

If (condition) {

} else {

}

در کد فوق اگر شرط صدق کند کد درون بلاک شرط اجرا میشود اما اگر صدق نکند شرط داخل بلاک else اجرا میشود از این ساختار برای شروطی استفاده میشود که دو حالت دارند.

متد .length طول یک رشته را بر می گرداند و یا تعداد آیتم های یک آرایه را برمی گرداند.

درون پرانتز می توان هرچیزی را قرار داد مثلا دو عبارت را مقایسه کنیم یا تابعی قرار دهیم که خروجی اش true یا false باشد.

عملگر ! شرط را معکوس می کند، && به معنای و هست و می توان دو تا شرط را در if قرار داد که در صورتی که هردو صحیح بودند کد اجرا شود. || به معنی یا است که اگر یکی از دو شرط صدق کند بلاک کد اجرا می گردد.

اگر شرط ما چندین حالت داشته باشد از else if() استفاده می کنیم:

If (condition) {

} else if (another condition) {

} else {

}

به هر تعداد که لازم باشد می توان از else if استفاده کرد. نوشتن else ضروری نیست. شرط ها به ترتیب بررسی میشوند اگر شرطی درست باشد دیگر مابقی شرط ها بررسی نمی شوند و نادیده گرفته می شوند.

جلسه دهم – دستورات شرطی – بخش دوم

اگر حالات شرط زیاد باشند می توان از switch case استفاده کرد:

switch (value) {

case value :

// code to run

case value :

// code to run

}

در کد فوق مقدار داخل پرانتز مقابل switch با مقدار نوشته شده در مقابل case مقایسه میشود و اگر برابر باشند کد ها اجرا می شوند. کد فوق ناقص است و برای همین هرجا که شرط صدق کند کد case های بعدی به کل اجرا می شوند. اما اگر در پایان کد های هر case کلمه کلیدی break; را قراردهیم دیگر فقط کد های مربوط به همان case اجرا می شوند.

اگر بخواهیم همانند else در if یک حالت داشته باشیم که در صورت صدق نکردن هیچکدام از شرط ها کدی اجرا شود در switch case به صورت زیر آن را تعریف می کینم:

switch (value) {

case value :

// code to run

break;

case value :

// code to run

break;

default:

// code to run

}

در جاهایی که به جای بررسی کردن برابر بودن مقادیر با مقدار switch case نیاز داریم نتیجه مقایسه لحاظ شود مقدار داخل پرانتز switch را true قرار میدهیم و در مقدار case ها مقایسه را انجام می دهیم پس اگر مقایسه true باشد کد داخل case اجرا می شود.

switch (true) {

case (num == 10) :

// code to run

break;

case (num > 11) :

// code to run

break;

…

}

جلسه یازدهم – آموزش کار با حلقه ها

حلقه ها از تکرار در کد نویسی به شدت جلوگیری می کنند.

انواع حلقه داریم که دو نوع آن for و while هستند.

حلقه for:

به صورت زیر تعریف می شود:

for (var i = 0; i < 5; i++) {

// code

}

در کد فوق حلقه for سه عبارت می پذیرد و با ; هر عبارت پاین داده می شود.

اولین عبارت متغیری است که قرار است پیمایش روی آن صورت گیرد، می توان متغیر را همان جا نیز تعریف کرد.

دومین عبارت شرط است مثلا مادامی که i در کد فوق کوپکتر از 5 باشد حلقه ادامه می یابد.

سومین عبارت دستوری است که پس از هر بار اجرای کد اجرا میشود مثلا در کد فوق پس از هر بار اجرا مقدار i یک واحد اضافه می شود. می تواند مثلا i-- باشد تا هربار متغیر یک واحد کم شود یا می تواند موارد مشابهی باشد مثلا i \* 2 که هربار i دوبرابر شود و شرط روی آن بررسی شود.

نکته متغییر درون () حلقه اگر با استفاده از var تعریف گردد در خارج از حلقه همچنان در دسترس است اما اگر با استفاده از let تعریف شود فقط در داخل for در دسترس است و درست هم همین است.

حلقه while:

برای تعریف این نوع حلقه به صورت زیر عمل می کنیم:

While (condition) {

}

در کد فوق شرط می تواند هرچیزی باشد مثلا داده ایی را از دیتابیس بگیرد و مادامی که درست باشد حلقه ادامه پیدا کند و موارد مشابه اما می توان از آن همانند حلقه for نیز استفاده کرد.

var i = 100

while (i > 10) {

// code

i++

}

با نوشتن break درون حلقه می توان آن را متوقف کرد مثلا یک استثنا را در یک if تعریف بررسی کنیم و break درون if باشد و اگر if اجرا گردد حلقه متوقف میشود چون درون if دستور break قرار گرفته.

نوع دیگری از حلقه while وجود دارد، حلقه do-while:

do {

} while ();

تفاوت حلقه فوق با حالت قبل این است که کد حداقل یکبار اجرا میشود مثلا هنگامی که پسوردی ساخته میشود ابتدا فرایند ایجاد پسورد در بلاک do یکبار اجرا می شود سپس بررسی میشود که تکراری نباشد و اگر پسورد تکراری باشد حلقه یکبار دیگر اجرا میشود و پسورد جدیدی ساخته می شود.

کامنت گذاری

اگر در جلوی کد یا متنی که نوشتیم // قرار دهیم کامنت میشود و نادیده گرفته می شود. همچنین برای کامنت کردن 2 یا چند خط آن خطوط را در بیان /\* \*/ قرار می دهیم.

جلسه دوازدهم – آرایه ها و حلقه ها

تعریف کردن لیست: var list=[1,2,3,4]

برای افزودن یا آپدیت آیتمی به آرایه فوق به صورت روبرو عمل می کنیم: list[4] = 5

اگر ایندکس از قبل مقدار داشته باشد مقدار آن عوض خواهد شد اما اگر مقدار نداشته باشد ایجاد می گردد و مقدار درونش ریخته می شود.

اگر آرایه تا ایندکس 4 ام مقدار دهی شده باشد و یهویی به ایندکس 7 ام بپریم و آن را مقدار دهی کنیم، وقتی که مقدار آرایه را نمایش دهیم توسط console.log در میان ایندکس 4 و 7 می نویسد 3 × empty یعنی سه خانه ایی که در این بین قرار دارند خالی هستند. و اگر مثلا ایندکس 5 ام را نمایش دهیم مقدار undefined را بر می گرداند.

متد .length همانطور که قبلا نیز ذکر شد تعداد آیتم های آرایه را برمی گرداند اما اگر برای string استفاده شود طول رشته را بر می گرداند.

متد .forEach() حلقه ای را اجرا می کند که به اعضای آرایه را پیمایش می کند.

arr.forEach( function(item) {

}

در کد فوق item آیتمی از آرایه است که می توان کلید را نیز به تابع بی نام به عنوان پارامتر دوم پاس داد.

list.forEach(function(item,key) {

console.log(item,key)

})

متد .reduce برای پیمایش یک آرایه و محاسبه مجموع آیتم های آن به کار می رود:

Numbers.reduce(function(total, value) {

console.log(total, value)

})

در کد فوق Numbers آرایه ای است که تمامی آیتم های آن عدد هستند، total مجموع آیتم ها را در خود میریزد و با هر بار اجرا شدن حلقه مقدار آیتم به total اضافه میشود که مقدار آیتم خودکار به عنوان دومین پارامتر پاس داده میشود به تابع بینام، توجه داشته باشید که total و value در کد فوق می توانند هر چیز دیگری نام گذاری شوند ولی همیشه پارامتر اول مجموع آیتم ها است و دومی مقدار آیتم است.

می توان مقدار پیش فرض نیز برای total در نظر گرفت که به عنوان پارامتر دوم به خود متد .reduce پاس میدهیم اما نوشتن آن ضروری نیست و دلبخواه است.

جلسه سیزدهم – توابع کار با آرایه ها

در این جلسه متد های کار با آرایه ها معرفی می شوند.

.push() این متد یک مقدار را به عنوان ورودی می پذیرد و آن را به عنوان آخرین عضو آرایه اضافه می کند. مقداری که بر می گرداند ایندکس عضو جدید است.

.pop() آخرین عضو یک آرایه را حذف می کند و خروجی آن مقدار حذف شده می باشد.

.shift() اولین عضو یک آرایه را حذف می کند و خروجی آن مقدار حذف شده می باشد.

.unshift() این متد یک مقدار را به عنوان ورودی می پذیرد و آن را اولین عضو آرایه می کند و خروجی آن تعداد عضو های آرایه پس از اضافه شدن عضو جدید می باشد.

.indexOf() به عنوان ورودی مقداری را می پذیرد و در آرایه می گردد و هر عضوی که مقدارش برابر مقدار ورودی متد باشد ایندکس آن را بر می گرداند. اگر هیچ یک از مقادیر با آن برابر نباشند 1- را برمی گرداند.

.splice(param1, param2) این متد مقدار ورودی را ایندکس شروع در نظر می گیرد و به تعداد ورودی دوم از آن ایندکس شروع، عضو ها را حذف می کند. درضمن مستقیم روی آرایه اعمال می شود. و خروجی آن عضو های حذف شده است. اگر تعدادی که برای حذف پاس داده شده بیشتر از تعداد آرایه های باشد که از نقطه شروع وجود دارد فقط آنچه که وجود دارد حذف می شود.

.slice(param1, param2) ورودی اول ایندکس شروع است و ورودی دوم ایندکس پایان است و از نقطه شروع تا پایان کل عضو ها را بر می گرداند. و در ضمن اصل آرایه تغییر نمی کند و این متد فقط عضو های انتخاب شده را بر می گرداند.

.join() اعضای آرایه را به حالت string در می آورد و , ها نیز در بین آن ها باقی می ماند البته می تواند کاراتر یا عبارتی را به این متد پاس داد تا به جای , آن مقدار قرار گیرد. ضمناً با اصل آرایه کاری ندارد.

.concat() به عنوان وردوی identifier یک آرایه را می پذیرد و آن را در ادامه آرایه ایی که متد روی آن صدا زده شده است قرار می دهد. برای ذخیره آرایه جدید باید آن را در یک آرایه دیگر بریزیم چراکه روی اصل آرایه ها اثری نمی گذارد.

.sort() آرایه را مرتب سازی می کند و روی لیست اصلی مستقیم اعمال می شود. می تواند پارامتر های متنوع بگیرد برای اهداف مختلف که می توان در MDN آن ها را بیشتر بررسی کرد.

.reverse() آرایه های لیست را معکوس می کند.

list.filter(function(item) {

return item % 2 == 0

)}

متد فوق که یک تابع بینام می پذیرد، تک تک اعضای آرایه را به تابع پاس می دهد و در بلاک متن تابع می توان شرطی قرار داد تا فقط اعضایی که شرط پاس کی کنند برگردانده شوند مثلا کد فوق فقط اعضای زوج آرایه را برمی گرداند. همچنین روی لیست اصلی تاثیری نمی گذارد و مقادیر برگشت داده باید در یک آرایه ریخته شوند برای ذخیره شدن.

list.map(function(item) {

return item \* 2

}

همانند .filter است نحوه ی کار کردنش با این تفاوت که هر عضوی را که میگیرد تغییراتی مطابق آنچه ما چیدیم روی آن اعمال میکند و مقدار جدید را جایگزین عضو قدیمی می کند. و لیست اصلی تغییر نمی کند. کد فوق همه ی اعضا را دو برابر می کند.

جلسه چهاردهم – شروع کار با Object ها

در جاوا اسکریپت شی ها به صورت زیر تعریف می گردند:

let obj1 = {

key: value,

key: value,

…

}

درون شی ها می توان به صورت key, value اعضا را تعریف کرد و هر عضو می تواند متغییر باشد، شی باشد، آرایه باشد ویا حتی می تواند یک تابع باشد.

let obj2 = {

name: ‘aran abdi’,

age: 24,

languages: [‘kurdish’, ‘persian’, ‘english’],

face: {

hair: ‘dark’,

eyes: ‘dark brown’

},

run: function(speed) {

return speed \* 3

}

}

کد فوق یک نمونه از آبجکت با شی ای است که انواع value ها را دارد.

امکان ویرایش مقادیر یک آبجکت وجود دارد، برای دسترسی به یک key از شی از . استفاده می کنیم مثلا در مثال فوق برای دسترسی به کلید name به صورت روبرو عمل می کنیم، obj2.name این کد مقدار name را بر می گرداند و یا می توان مقدار جدیدی را به آن منتصب کرد. به ویژگی های داخل شی ها property می گویند.

گرفتن مقدار یک پراپرتی را get کردن و مقدار دهی یک پراپرتی را set کردن می گویند.

نکته مهم: - در نوشتن key نمی تواند استفاده شود مگر اینکه کل عنوان key درون “ “ نوشته شود که در اینصورت دیگر نمی توان آن را با استفاده از . فراخوانی کرد و بهش دسترسی داشت بلکه باید بصورت زیر آن را فراخوانی کرد: objecName[“aran-abdi”] حال دیگر به پراپرتی مورد نظر دسترسی داریم البته پراپرتی های معمولی را نیز می توان اینگونه صدا زد.

در جاوا اسکریپت همانگونه که در obj2 نیز مشاهده می شود می توان درون یک پراپرتی یا متغییر تابعی را قرار دهیم که در اینصورت دیگر بعد از کلمه function نام تابع لازم نیست و نام تابع همان نام متغیر یا پراپرتی می باشد.

var run = function() {

}

تابع های درون شی بصورت روبرو فراخوانی می گردند: objectName.functionName() در شی گرایی به تابع متد نیز گفته می شود.

برای دسترسی به یک پراپرتی از آبجکت داخل خود آبجکت از کلمه کلیدی this استفاده می کنیم.

this.model

برای تعریف کردن شی روش دیگری نیز وجود دارد:

var obj3 = new Object()

در روش فوق از کلاس Object استفاده می کنیم و یک شی جدید میسازیم که می توان بصورت زیر به آن پراپرتی افزود:

Obj3.name = “-----”

روش دیگری نیز برای ساخت شی وجود دارد که در آینده بیشتر به آن می پردازیم:

var obj4 = Object.create()

جلسه پانزدهم – تابع سازنده Object

تابع سازنده شی یا آبجکت تابعی است که به آن constructor function می گویند، تابعی است که می توان از روی آن شی هایی را تولید کرد.

Function Car() {

this.brand = ‘Ford’

this.model = ‘Mustang’

this.run = function() {

return ------------

}

}

درون تابع سازنده پراپرتی ها و متد ها را با کلید this تعریف می کنیم، برای ساخت نمونه از روی تابع سازنده فوق بدین صورت عمل می کنیم:

var car1 = new Car()

می توان برای تابع سازنده پارامتر نیز قرارداد تا بر اساس ورودی ایی که می گیرد پراپرتی ها را مقدار دهی کند.

car1 instanseof Car

کد فوق بررسی می کند که آیا car1 نمونه ای از Car هست یا خیر.

جلسه شانزدهم – آشنایی با Prototype

در اصل باید از مفاهیمی همچون کلاس و ... در شی گرایی استفاده کینم که در ES6 این استاندارد ها به جاوا اسکریپت افزوده شد اما همچنان در پس زمینه از constructor function استفاده می شود.

متد .constructor متد سازنده ی آبجکت رو بر می گرداند.

آبجکتی که به صورت معمولی تعریف شده نه به کمک تابع سازنده، سازنده اش Object برگردانده می شود که خود بخشی از هسته جاوا اسکریپت است و کد Native هست.

پروتوتایپ

هر شی یک پروتوتایپ دارد که موارد مربوط به شی درون آن قرار گرفته است.

متد .hasOwnProperty چک می کند که پراپرتی وجود دارد که نام پراپرتی باید به عنوان یک رشته به این متد پاس داده شود تا بررسی کند.

آبجکتی که با تابع سازنده ساخته شده است proto دارد که خود باز جاوی \_\_proto\_\_ می باشد.

شی گرایی در جاوا اسکریپت برای داستان ارث بری از مفهمو پروتوتایپ استفاده می کند.

تعریف یک متد برای تابع سازنده، یا به عبارتی افزودن یک متد به تابع سازنده:

objectName.prototype.methodName = function() {

}

اگر از حالت فوق برای افزودن متدی به Object اصلی استفاده شود چی؟

Object.prototype.sum = function() {

}

آن وقت چون Object پدر همه ی شی های درون جاوا اسکریپت است تمام آبجکت های دیگر این متد را نیز خواهند داشت. اما توجه داشته باشید که موارد به ارث برده شده را می توان بازنویسی کرد.

یک روش دیگر برای تعریف آرایه ها وجود دارد که اعضای آن بصورت key و value خواهند بود مثه آبجکت ها.

var list = new Array(1,2,3)

constructor آرایه فوق Array است که همانند Object در هسته جاوا اسکریپت قرار دارد و Native است.

Proto خود Array، Object است.

var name = new String(‘ ‘)

برای تعریف کردن رشته بکار می رود که از String ساخته شده است.

متد .\_\_proto\_\_ پروتوتایپ را بر می گرداند.

var num = new Number() // number

var status = new Boolean() // true or false

همانند موارد قبل کاربرد دو کد فوق نیز بدیهی است و برای انواع نوع داده ها می شود اینگونه آن ها را تعریف کرد.

جلسه هفدهم – ارث بری در مفهوم Prototype

حالتی برای ساختن شی ایی که proto نداشته باشد:

Object.create(null)

حتی متد .constructor را اگر روی شی ساخته شده ی فوق صدا بزنیم مقدار undefined برگشت داده میشود.

برای ارث بری در جاوا اسکریپت به صورت زیر عمل می کنیم:

var car2 = object.create(car1)

در کد فوق car1 از car2 ارث بری می کند. در ابجکت فرزند می توان پراپرتی ها و متد های پدر را بازنویسی کرد. توجه داشته باشید که این موارد در Frontend زیاد کاربردی نیستند اما در Backend و NodeJs کاربرد زیادی دارند.

جلسه هجدهم – کار با call، apply، bind

این سه مورد همگی هم در Backend و هم در Frontend کاربرد دارند.

داستان از این جا شروع می شود که اگر تابعی را خارج از بلاک یک شی بنویسیم و درون تابع از کلمه کلیدی this استفاده کنیم این this به خود تابع اشاره می کند و نمی توان از آن برای دسترسی به موارد درون شی (آبجکت) استفاده کرد. برای حل این مشکل از این سه گزینه به شرح زیر بهره می بریم:

شایان ذکر است که apply، bind و call هرسه یک کار را انجام می دهند با مقداری تفاوت.

متد Apply

functionName.apply(objectName)

کد فوق کلمه کلیدی this را دیگر در تابع به شی پاس داده شده به apply منتسب می کند.

اگر خود تابع نیز پارامتر داشته باشد، آرگومان ها را به صورت زیر به آن پاس می دهیم:

functionName.apply(objectName, [arguments])

اگر بنویسیم:

functionName(arguments).apply(objectName)

به هدف فوق دست پیدا نمی کنیم چراکه قبل از رسیدن به مند apply تابع اجرا می شود و دیگر این گونه تعریف کردن فایده ایی ندارد. اما در عوض آرگومان ها را درون یک آرایه به عنوان پارامتر دوم به متد apply پاس می دهیم.

متد call

تفاوت متد call با apply در این است که متد call آرگومان ها را به صورت آرایه نمی پذیرد بلکه آرگومان اول نام آبجکت در نظر گرفته می شود و مابقی آرگومان ها که تک تک پاس داده میشوند به عنوان آرگومان های پاس داده شده به تابع لحاظ می شوند.

functionName.call(objectName, arg1, arg2 ..)

متد bind

دو متد قبلی یعنی call و apply هردو پس از نوشته شدن تابع را فراخوانی می کنند و آن را اجرا می کنند اما متد bind اینگونه نیست. این متد در React بسیار پر کاربر است.

var variable = functionName.bind(objectName, arg1, arg2 ..)

متد bind همانند متد call آرگومان ها را تک به تک دریافت می کند اما آن را در متغیری میریزیم و بعد متغیر را به صورت تابع صدا می زنیم: variable() و اینگونه متد اجرا می شود ویا می توان مستقیم و بدون نیاز به ریختن آن درون آرایه به صورت زیر آن را فراخوانی کرد تا اجرا شود.

functionName.bind(objectName, arg1, arg2 ..)()

اینگونه دیگر کد فوق با قرار گرفتن () در جلویش اجرا می گردد مستقیم.

حتی اگر متد یا تابعی درون آبجکت دیگری تعریف شده باشد به کمک این سه متد می توان از متد آن آبجکت برای آبجکت دیگری نیز استفاده کرد.

جلسه نوزدهم – دیباگ کردن و رفع مشکل

Debugging یعنی پیدا کردن مشکلات احتمالی و رفع کردن آن ها. در جاوا اسکریپت نیز مانند اکثر زبان های دیگر پروسه ی دیباگ کردن داریم.

console.log() یکی از ابزار های دیباگ کردن است که بسیار کاربردی است و در NodeJs مخصوصاً خیلی کمک می کند در گذشته از alert نیز استفاده می کردند که برای این کار گزینه نامناسبی هست.

می توان افزونه Debugger for chrome را نصب کرد در vs code که deprecate شده و منسوخ است.

اما یک افزونه دیگر به جای آن معرفی شده است.

از طریق مرورگر نیز می توان پروسه ی دیباگ را داشت، در inspect تب source را باز میکنیم و برای ما کد های جاوا اسکریپت رو میاره و میشه کد رو خط به خط اجرا کرد و مشکل رو پیدا کرد.

همچنین اگر کد debugger رو در میون کد هامون بنویسیم هر جا که به این دستور برسد مرورگر به حالت debug سویچ می کند.

جلسه بیستم – آشنایی با try catch

یک دستور خیلی خوب است که در صورت بروز خطا آن را برای ما مدیریت می کند و بسیار کاربردی است.

Try {

} catch (err) {

}

در کد فوق اگر به هر دلیلی کد هایی که در بلاک try اجرا می شوند با خطا مواجه شوند خطا به عنوان آرگومان به catch پاس داده می شود و می توان نام پارامتر را هرچزی که می خواهیم بگذاریم.

آرگومان مربوط به ارور دریافت شده دو مورد را بر می گرداند:

err.message که پیغام ارور را در بر دارد و err.name که عنوان ارور را در بر می گیرد. پس مقدار err میتواند هرچیز دیگری نیز باشد.

می توان به صورتی دستی نیز اروری را فراخوانی کنیم تا در catch گرفته شود، مثلا این کد را درون یک بلاک شرطی میگذاریم تا درصورت صدق نکردن شرطی که در بلاک try تعریف شده یک خطا برگشت داده شود.

throw ‘error text’

یا

throw {name: ‘ ‘, message: ‘ ‘}

به دنبال بسته شدن بلاک کد catch می توان finally { } را نیز صدا کرد:

Try {

} catch (err) {

} finally {

}

کد موجود در بلاک کد finally چه کد داخل try درست باشد و چه کد داخل catch اجرا شود و کد ارور داشته باشد اجرا میشود.

یادآوری: throw توسط catch دریافت می گردد خودکار.

جلسه بیست و یکم – کار با Timer و Interval

برای این که کد مورد نظر پس از مدت مورد نظر اجرا شود از کد زیر استفاده می کنیم:

setTimeout(function() {

// code

}, میلی ثانیه)

هرگاه بخواهیم کدی هر چند وقت یکبار اجرا شود از کد زیر استفاده می کنیم:

setInterval(function() {

// code

}, میلی ثانیه)

جلسه بیست و دوم – توابع کار با رشته

متد .length تعداد کاراکتر های رشته را می شمارد، حتی فاصله ها را نیز می شمارد.

متد .charAt() این متد به عنوان ورودی یک عدد می پذیرد و پوزیشن n ام که در اینجا n عدد ورودی باشد را بر می گرداند هر کاراکتری که باشد را بر می گرداند. ضمناً همانند آرایه ها شمارش از 0 شروع می شود.

اگر همانند آرایه برای رشته نیز از stringName[0] استفاده کنیم مانند کد فوق است و آن کاراکتر را بر می گرداند که ایندکس در میان [] قرار گرفته است.

.concat() به عنوان ورودی باید به آن رشته پاس بدهیم و آن رشته را در ادامه ی رشته ای که متد روی آن فراخوانی شده است قرار می گیرد. اما توجه داشته باشید که متغیر اصلی تغییری نمی کند.

.toUpperCase() این متد تمام حروف را بزرگ می کند اما متغیر اصلی تغییر نمی کند.

.toLowerCase() این متد کلیه حروف رو با حرف کوچک جایگزین می کند. اما متغیر اصلی تغییر نمی کند.

.split() به عنوان ورود کاراکتر یا عبارت خاصی را دریافت می کند و هر جا در رشته یه آن متن برسد آن را جدا می کند و عضوی از آرایه در نظر می گیرد. باید خروجی این متد درون آرایه ای ریخته شود.

.trim() این متد فاصله های اضافی بعد و قبل رشته رو حذف می کند. متغیر اصلی تغییر نمی کند.

.indexOf() عبارتی که به عنوان ورودی گرفته و در رشته آن را جستجو می کند، اگر متن یافت نشود 1- بر گردانده میشود اما اگر عبارت وجود داشته باشد ایندکس یا پوزیشن اولین حرف آن برگردانده می شود.

.subStr() پارامتر اول آن ایندکس یا پوزیشن را می گیرد و پارامتر دوم تعداد حرفی را تعیین می کند که باید جدا شوند، از پوزیشن دریافتی این تعداد را جدا می کند. متغیر اصلی تغییری نمی کند.

.replace() این متد ورودی اولش متنی است که باید در رشته جستجو شود. و ورودی دوم متنی است که باید جایگزین متن یافته شده شود. البته می توان در آن قاعده یا قانون تعریف کرد تا بر آن اساس عمل شود ولی قدری پیچیده است این کار و نیازمند تمرین و پیگیری بیشتری است.

جلسه بیست و سوم – کار با زمان

در موزیلا متدهای زیادی در این مورد معرفی شده اند که در اینجا تعدادی از آن ها آورده شده است.

نکته: پیشنهاد میشود زمان را در دیتابیس میلادی ذخیره کنیم.

var date1 = new Date()

دستور فوق تاریخ و زمان لحظه را درون متغیر می ریزد.

Timestamp از ابتدای سال 1970 تا به اکنون به صورت عددی است.

متد .getTime() زمان متغیری که در بر گیرنده تاریخ و زمان است را بر می گرداند.

اگر به new Date() مقدار timestamp ایی را پاس دهیم تاریخ و زمان رو نشون میده.

Date.now() وقت را بر با فرمت timestamp برمی گرداند.

Date.parse(‘2020/10/10’) این کد نیز timestamp را بر می گرداند.

.getHourse() ساعات را بر میگرداند.

.setHours() ساعت را set میکند و عدد می پذیرد به عنوان ورودی اگر ساعت بیشتر از 24 ساعت باشد در تاریخ به روز و ... افزوده میشود به تناسب.

جلسه بیست و چهارم – توابع محاسباتی Math

در موزیلا همه ی توابع محاسباتی آورده شده اند اما در این جلسه به برخی از آن ها می پردازیم.

Math.PI عدد p را بر می گرداند 3.14

Math.abs() عدد منفی را به عنوان ورودی می گیرد و آن را مثبت می کند.

Math.floor() عدد اشاری را گرد به پایین می کند حتی اگر 0.99 هم باشد.

Math.random() یک عدد رندوم را بر می گرداند.

Function getRandomInt(max) {

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

}

کد فوق یک عدد رندوم بر می گرداند که حداکثر می تواند تا max باشد، خود max نمی تواند باشد.

Math.round() اگر عدد اعشاری اعشارش از نصف بیشتر باشد گرد به بالا و اگر کمتر باشد گرد به پایین میکند. خود 0.5 را گرد به بالا می کند.

Math.pow(param1, param2) پارامتر اول را به توان پارامتر دوم می رساند.

جلسه بیست و پنجم – آشنایی با DOM

DOM دقیقاً همان تگ های html پروژه است.

نکته: تگ <head> نمی تواند فرزند تو در تو داشته باشد.

document 🡺 یک ابجکت است از نوع گلوبال

document.children 🡺 تگ های اچ تی ام ال رو بر می گرداند

document.children[index] 🡺 فرزند اینکدس ام را بر می گرداند

document.head و document.body 🡺 به تگ های هد و بادی اشاره می کند

document.body.style.backgroundColor = ‘red’ 🡺 استایل تگ را تغییر میدهد

در جاوا اسکریپت چون – نمیتونه در نام متغیر یا پراپرتی باشد پس به جای Kebab case برای ویژگی های css ای از camel case استفاده می کنیم.

document.body.children[0].textContent = ‘ ‘ 🡺 محتوای یک تگ را تغییر می دهد این پراپرتی

document.body.firstChild 🡺 اولین فرزند را بر می گرداند

document.body.firstElementChild 🡺 اولین تگ فرزند را بر می گرداند

document.body.firstElementChild.nextElementSibling 🡺 تگ کناری را انتخاب می کند

جلسه بیست و ششم – انتخاب کننده Element ها در DOM

document.getElementByTageName(‘tage-name’) 🡺 تمام تگ ها را میگیرد

document.getElementById(‘#id’) 🡺 تگی با آیدی خاص را میگیرد

document.getElementByClassName(‘.class) 🡺 تگ های با کلاس خاص را میگیرد

روش بعدی بر اساس selector های css است:

document.querySelector(‘.list-group’) 🡺 فقط اولین تگ با این مشخصات را میگیرد

document.querySelectorAll(‘.list-group’) 🡺 همه ی تگ های با این مشخصات را میگیرد

در حالت فوق که تگ ها دسته جمعی گرفته شدند یک استایل را نمی توان در ادامه برای همه تعیین کرد بلکه باید با forEach تک تک تگ ها پیمایش شوند و استایل برایشان تعیین گردد.

جلسه بیست و هفتم – روش حذف و ایجاد Element ها در DOM

برای ایجاد یک تگ از دستور زیر استفاده می کنیم البته در html نمایش داده نمی شود چراکه فعلا فقط در کد های جاوا اسکریپت ساخته شده است. و می توانیم آن را در یک متغیر بریزیم.

var ul = document.createElement(‘ul’)

متد .classList لیست کلاس ها را در یک آرایه بر می گرداند.

برای اضافه کردن کلاس به یک تگ از کد زیر استفاده می کنیم.

ul.classList.add(‘class-name’)

متد .toggle() اگر روی classList فراخوانی شود اگر کلاس باشد آن را حذف می کند و اگر نباشد آن را به لیست کلاس های تگ اضافه می کند.

.id آیدی تگ را تعیین می کند

برای افزودن attribute های html به تگ مورد نظر از متد .setAttribute() استفاده می کنیم که پارامتر اول نام ویژگی است و پارامتر دوم مقدار آن.

یادآوری میشود که ul هنوز در html نمایش داده نمی شود.

ul.appendChild()

کد فوق به عنوان ورودی تگ دریافت می کند و این تگ درون تگ فعلی ریخته می شود و فرزند آن میشود.

متد .after() تگ ورودی را بعد از تگ مورد هدف که after() روی آن فراخوانی شده است قرار میدهد.

متد .befor() تگ ورودی را قبل از تگ مورد هدف که befor() روی آن فراخوانی شده است قرار میدهد.

اگر after و before باهم استفاده شوند برای تگی روی تگ دیگری فقط یکبار تگ به داخل html افزوده میشود و دوباره تکرار نمی شود و فقط آخرین موردی که تعریف شده کار می کند.

حذف کردن DOM

document.querySelector(..).remove() 🡺 خود تگ حذف می شود

document.parentElement.remove(..) 🡺 تگ پدر حذف می شود

جلسه بیست و هشتم – آشنایی با Dialogs

مواردی که در این جلسه تدریس میشوند صرفاً برای آشنایی هستند و در پروژه های واقعی چون ظاهر مناسبی ندارند از آن ها استفاده نمی شود و در عوض از پکیج هایی همچون Sweet Alert استفاده میشود.

برای نمایش یک پیام به کاربر از کد زیر استفاده می کنیم:

alert(‘message’)

همانطور که در بخش دیباگ نیز گفته شد در گذشته از این alert برای log گرفتن استفاده میشد اما حالا از گزینه ی بهتری console.log() استفاده می کنیم.

پکیج هایی که پیش تر نیز به آن ها اشاره کردیم این موارد رو تغییر نمی دهند و این ها پیش فرض مرورگر هستند و مگر در مواردی موقتی استفاده شوند تا در آینده جایگزین شوند و پکیچ ها دیالوگ هایی را در html برای ما ایجاد می کنند که ظاهر بسیار مناسب تری دارند.

var status = confirm(‘message’)

کد فوق پیغام را به همراه دکمه های ok و cancel نمایش می دهد اگر کاربر ok را بزند مقدار متغیر برابر با true و در غیر اینصورت بلعکس. می توان از متغیر در شرط استفاده کرد و برنامه را مسیر دهی کرد اینگونه.

var answer = prompt(‘message’)

کد فوق یک input را در اختیار کاربر قرار میدهد تا مقداری را وارد کند و این مقدار در متغیر ما ریخته میشود.

جلسه بیست و نهم – آشنایی با Event و Event Handler ها

هنگام بارگزاری وبسایت رویداد های متنوعی رخ میدهند، بحث رویداد ها را هم سمت سرور داریم هم سمت client، در وبسایت موزیلا مستندات مربوط به Event ها موجود است که زیاد هستند.

مواردی ( یا متد هایی) که on دارند Event Handler هستند.

TAG.onclick = function() {

}

کد فوق هنگام کلیک کردن کاربر روی تگ هدف اجرا می شود.

.onmouseenter همانند hover در css است وقتی موس روی تگ برود کد اجرا می گردد.

.ondbclick هنگام دبل کلیک روی تگ هدف کد اجرا می شود.

توجه داشته باشید که این موارد در مرورگر های مختلف، متفاوت پشتیبانی می شوند.

می توان مواردی را از این رویداد ها دریافت کرد مانند محل کلیک و ...:

TAG.onclick = function(event) {

event.target.style.backgroundColor = ‘red’

}

برای مثال کد فوق رویداد را به عنوان پارامتر به تابع بی نام پاس می دهد و رنگ زمینه تگ کلیک شده را قرمز می کند. event می تواند هر نام دیگری باشد.

جلسه سی ام – آشنایی با Event Listener ها

اگر چندین event handler داشته باشیم فقط آخری اجرا می شود و مابقی نادیده گرفته می شوند. اما این موضوع برای event listener ها صادق نیست و می توان چندین listener برای یک تگ تعریف کرد.

TAG.addEventListenter(‘on.., function() {

})

اگر تگی داخل تگی دیگر باشد و هردو event listenter داشته باشند، با کلیک روی فرزند رویداد پدر نیز فراخوانی می شود برای اینکه فقط رویداد فرزند اجرا شود کد زیر استفاده میشود:

TAG.addEventListener(‘onclick’, function(e) {

e.stoppropagation()

}

متد .preventDefault() را اگر روی event فراخوانی کنیم کاری که پیش فرض صورت می گرفت متوقف میشود مثلا در فرم دیگر فرم submit نمی شود.

میشود تابع بی نام را با نام یک تابع که در خارج از listener تعریف کرده ایم جایگزین کنیم. در صورت پاس دادن تابع به listener، نباید () جلوی نام تابع قرار گیرد تا هروقت مه خواست تابع اجرا بشه اگر () رو بنویسیم تابع فوری اجرا می شود.

.removeEventListener() این متد پارامتر اول رویداد را می گیرد ( کلیک و ...) سپس پارامتر دوم نام تابع را می گیرد، تابعی که در هنگام تعریف listener به آن پاس داده شده است پس برای اینکه بتوان listener را بعدا حذف کرد حتماً به جای استفاده از anonymous function باید نام تابع را به آن پاس بدهیم.

جلسه سی و یکم – آموزش کار با درخواست AJAX

با آمدن Ajax این امکان اضافه شد که بدون لود کردن صفحه درخواستی به سرور ارسال شود و پاسخی را دریافت کرد.

در سایت موزیلا xmlHttpRequest (جاوا اسکریپت خام) توضیح داده شده اما این روش پیشنهاد نمی شود و می توان از طریق کتابخوانه JQuery یا Axios که یک کتابخانه ی اختصاصی برای این مورد است درخواست ajax را مدیریت کرد.

var ajax = new xmlHttpRequest()

ajax.open(‘GET’, ‘api url goes here’)

ajax.send()

دو متد پر کاربر داریم برای ارسال درخواست، GET و POST که GET برای دریافت اطلاعات و POST برای ارسال دیتا از وبسایت استفاده می شود.

وبسایت JsonPlaceholder برای استفاده از یک api تستی وبسایت خوبی هست.

تا به این جای کار ساده بود با متد open یک درخواست ajax باز کردیم و با .send آن را ارسال کی کنیم اما مدیریت دریافت داده مقداری پیچیده است.

برای دریافت داده های برگشتی در میان .open و .send کد های زیر را می نویسیم:

Ajax.onreadystatechange = function() {

If (this.readystate == xmlHttpRequest.DONE) {

If (this.status == 200) {

console.log(this.responseText)

}

}

}

this.responseText داده های برگشتی را می گیرد در شرط اول چک میشود آیا درخواست انجام شده باشد و در شرط دوم چک می شود که آیا دریافت موفقیت آمیز بوده.

می توان به جای Event Handler ای که در بالا استفاده شد یعنی .onreadystatechange از event listener زیر استفاده کرد:

ajax.addEventListener(‘load’, function()..

جلسه سی و دوم – آموزش کار با درخواست AJAX – درخواست POST

این بار پارامتر اول متد .open را “POST” قرار می دهیم و ما بین .open و .send کد های زیر قرار می گیرند:

var data = [..]

داده هایی که ارسال میشوند را در آرایه فوق میریزیم.

ajax.setRequestHeader(‘Content-Type’, “application/json”)

کد فوق تعیین می کند که نوع داده های ارسالی json هست.

ajax.addEventListener(‘load’, function() {

if(this.readystate == xmlHttpRequest.DONE) {

console.log(‘done’, this.responseText)

} else..

})

دسترسی به مقادیر response

var data = JSON.parse(this.responseText)

کد فوق response ایی را که به شکل json دریافت شده است را به object تبدیل می کند و می توان و داده های آن به راحتی دسترسی داشت همانند زیر:

data.username

جلسه سی و سوم – فریم ورک ها کتابخونه ها در جاوا اسکریپت

JQuery در گذشته به شدت محبوب بود و کار ها به DOM را خیلی ساده تر می کرد و همه از آن استفاده می کردند اما به مرور زمان و بزرگتر شدن پروژه ها مدیریت کد ها همچنان سخت بود.

کتابخونه ها یکسری توابع از پیش تعریف شده را در اختیار ما قرار می دهند اما فریم ورک ها یک چهارچوبی هستند که کتابخونه های رو در جای مناسب قرار می دهند که کار ها رو برای ما خیلی ساده تر کنند. پس در فریم ورک ها مجموعه ایی از کتابخونه ها در دسترس ما قرار می گیرند و سازماندهی اختصاصی برای این کتابخونه ها ممکنه صورت گرفته باشه تا کارها ساده سازی شوند.

مثلا Axios یک کتابخونه هست برای کار با Axios

فریم ورک های VueJs، Angular و react.js رو برای جاوا اسکریپت داریم که 3 فریم ورک مشهورند.

البته react خودش رو کتابخونه خطاب می کنه اما مجموعه کتابخونه هایی که ارائه میده باعث شده در نهایت امکاناتی مثه فریم ورک رو ارائه بده.

وبسایت های SPA به کمک فریم ورک ها ساخته می شوند. اولین فریم ورک angular بود و یادگیری آن مقداری سخت است ولی امکانات کاملی دارد و برای سازمان های بزرگ مناسبتر است.

Vuejs بعد از angular که یک فریم ورک خیلی سبکه به میدان آمد، میشود از آن مانند JQuery به صورت ساده نیز استفاده کرد یعنی SPA نباشد. اما angular حتما SPA است.

JQuery زمانی که این سه فریم ورک نبودند بسیار محبوب بود اما به کمک این سه فریم ورک کار کردن با DOM بسیار راحت تر شده.