**جاوا اسکریپت**

کامیونیتی بزرگی دارد و محبوب است.

جاوا اسکریپت یک زبان سطح بالا هست. (سطح بالا یعنی اینکه ما با زبان ماشین که ۰ و ۱ باشد سرو کار نداریم و از طریق یک زبان سطح بالا مثل جاوا اسکریپت که به زبان انسان نزدیک است و یادگیری آن آسان تر است).

جاوا اسکریپت یک زبان مفسری است یعنی کد ها خط به خط اجرا میشوند ولی در زبان های کامپایلری همه کد ها یک جا کامپایل میشه و بعد به صورت یکجا در سیستم عامل اجرا میشود.

جاوا اسکریپت یک زبان شی گرا است. جزییات در طول دوره توضیح داده میشود.

جاوا اسکریپت در سال ۱۹۹۵ توسط Brendan Eich ایجاد شد. که کارمند شرکت netscape بود و هدف ایجادش هم ایجاد سایت های پویا تر بود.

**اکما اسکریپت ECMAScript**

یک اسری استانداردهایی است که از سوی انجمنی اروپایی برای ایجاد زبان های برنامه نویسی تعریف میشود که مخفف :

European Computer Manufacturer’s Association

جاوا اسکریپت هم از استاندارد های اکما اسکریپت برای تمامی نسخه های خود استفاده میکند. و معروف ترین ورژن جاوا اسکرپیت اکاما اسکریپت ۶ است که توش تغییرات زیادی مشاهده شده. و در سال ۲۰۱۵ منتشر شد.

**لیست نسخه های جاوا اسکریپت**

|  |  |  |
| --- | --- | --- |
| **Description** | **Official Name** | **Version** |
| First edition | |  |  | | --- | --- | |  | ECMAScript 1 (1997) | | ES1 |
| Editorial changes | |  |  | | --- | --- | |  | ECMAScript 2 (1998) | | ES2 |
| Added regular expressions Added try/catch Added switch Added do-while | ECMAScript 3 (1999) | ES3 |
| Never released | ECMAScript 4 | ES4 |
| Added "strict mode" Added JSON support Added String.trim() Added Array.isArray() Added Array iteration methods Allows trailing commas for object literals | |  |  | | --- | --- | |  | ECMAScript 5 (2009) | | ES5 |
| Added let and const Added default parameter values Added Array.find() Added Array.findIndex() | ECMAScript 2015 | ES6 |
| Added exponential operator (\*\*) Added Array.includes() | ECMAScript 2016 | ES7 |
| Added string padding Added Object.entries() Added Object.values() Added async functions Added shared memory Allows trailing commas for function parameters | ECMAScript 2017 | |  | | --- | |  |   ES8 |
| Added rest / spread properties Added asynchronous iteration Added Promise.finally() Additions to RegExp | ECMAScript 2018 | ES9 |
| String.trimStart() String.trimEnd() Array.flat() Object.fromEntries Optional catch binding | ECMAScript 2019 | ES10 |
| The Nullish Coalescing Operator (??) | ECMAScript 2020 | |  | | --- | |  |   ES11 |
| String.replace.all() promise.any() | ECMAScript 2021 | ES12 |
| Top-level await New class elements Static block inside classes | ECMAScript 2022 | ES13 |
| toSorted method toReversed findLast, and findLastIndex methods on Array.prototype and TypedArray.prototypet | ECMAScript 2023 | ES14 |

به es6 به بعد اصطلاحا میگن modern js

برای کسب اطلاع از جزییات ورژن ها به لیتک زیر مراجعه شود:

[**https://webreference.com/javascript/basics/versions**](https://webreference.com/javascript/basics/versions)

**اجرای کد های جاوا اسکریپت:**

اجرای کد های جاوا اسکریپتی در مرورگر صورت میگیرد و در تمام مرورگر ها موتور جاوا اسکریپتی وجود دارد.

معروف ترین موتور جاوا اسکریپت v8 کروم است که در نود جی اس هم از همین استفاده شده است.

برای اجرا میتوان از تگ script در سند html استفاده کرده و کد های جاوا اسکریپت را داخل این تگ بنویسیم ولی این روش از لحاظ خوانایی و تمیز بودن کد مشکل دارد و بهتر است انجام نشود.

روش دوم و بهتر این است که کد های جاوا اسکریپت را داخل یک فایل جداگانه قرار دهیم. و تنها آدرس آن را با تگ script در سند html معرفی کنیم. بهتره تگ script در انتهای سند html قرار بگیره تا اول محتویات html لود شود و بعد بره سراغ جاوا اسکریپت.

نکته vsCode : اگر میخوای تغییراتت بدون ctrl+s سیو بشه برو تو setting و auto save را روی after delay قرار بده.

همچنین واسه اینکه هربار توی مرورگر ریلود نکنی از اسکتنشن live server استفاده کن.

**دستور console.log():**

برای دیدن نتیجه کد هامون از این دستور استفاده میکنیم. برای مشاهده ی این نتیجه باید به قسمت inspect element مراجعه کنیم.

**تفاوت bug و error:**

وقتی برنامه کار میکند ولی به هر دلیلی نتیجه ی مورد انتظار را نمیدهد میگیم برنامه باگ دارد. اما ارور زمانی که رخ میدهد برنامه دیگر اجرا نمیشود و کلا متوقف میشود.

**متغییر ها در جاوا اسکریپت:**حافظه هایی هستند که ما یک سری دیتاهای مورد نیازمون رو نگه داری میکنیم. در واقع متغیر ها مثل ظرفی می مانند که در داخلش دیتا هامون رو ذخیر میکنیم و در واقع نگه دارنده ی دیتاهای ما هستند. که یک اسم خاصی براش در نظر میگیریم.

به متغیر ها اصطلاحا data container گفته میشود.

**تعریف متغیر ها در برنامه:**

با استفاده از var – let – const متغیر را تعریف میکنیم مثلا:

Let age = 12

Var برای es5 بوده و ازش استفاده نمیکنیم ما.

Const فقط برای تعریف متغیرهایی که مقدار ثابتی دارند استفاده میشود. و این نکته رو در نظر بگیر که وقتی با const متغیری را تعریف میکنی حتما همان موقع باید مقدار دهی اولیه هم بکنی اما برای let و var این طور نیست مثلا کد زیر غلطه و باید درجا مقدار دهی اولیه هم بشه:

Const name; غلطه  
const name = ‘parisa’ درسته

**قواعد نام گذاری متغیرها:**

چهار روش نام گذاری داریم به صورت زیر :  
  
 camel case : حرف اول کلمه اول کوچیک و حرف اول کلمه دوم بزرگ نوشته میشود.

Snake case: بین دو کلمه از \_ استفاده میشود : first\_name

Kebab case: بین دو کلمه از – استفاده میشود : first-name

Pascal case: حرف اول همه کلمه ها بزرگ است: FirstName

در جاوا اسکریپت از camel case استفاده میشود.

برای نام گذاری میتوان از $ \_ استفاده کرد اما عدد اول اسم استفاده نمیشود. از کلمات کلیدی هم استفاده نمیشود

**Data type ها در جاوا اسکریپت:**

Number**:** تایپ عدد که شامل همه اعداد صحیح و اعشاری میشوند.

String: رشته ها بین ‘’ یا “” قرار می دهیم.

Boolean: فقط دو مقدار true , false دارد.

Array: داخل آرایه همه ی تایپ ها میتونن قرار بگیرند. تایپ آرایه آبجکت است

Object:

Null: این خودش یه مقداره و ما میتونیم مقدار null رو به یک متغیر assign کنیم. تایپ null آبجکت است

Undefined: وقتی به متغیری مقداری assign نمی کنیم مثلا let name در این صورت میشه undefined و تایپش undefined است

با متد typeof میشه نوع یا تایپ هر متغیر رو ببینیم : console.log(typeof name)

Operator ها یا عملگر ها :

1. Arithmetic Operator:

مثل جمع و تفریق و ضرب و تقسیم.

const x = 5;

const y = 6 ;

console.log(x + y)

واسه به توان رسوندن:

const x = 5;

const y = 6 ;

console.log(x \*\* y)

increment:

X = x + 1 یا x = + 1 یا x++

۲. assignment operators :  
انتساب دادن یک مقدار به یک متغییر مثلا let x = 5 که به این علامت مساوی میگن assignment operator  
یک روش assign کردن دیگر که بهش میگن addition assignment به این صورته که ما به جای اینکه بنویسیم   
x = x+4 مینویسیم : x += 4 برای منها و ضرب و تقسیم هم به همین صورت انجام میشه  
واسه باقی مونده هم به همین صورت انجام میشه x %=3 و برای توان هم به این صورت نوشته میشه x \*\*= 3

۳. comparison operators:

واسه مقایسه دو مقدار استفاده میشود و پاسخ در قالب Boolean داده می شود.یعنی نتیجه true یا false داده میشود.

واسه بررسی برابری دو مقدار از دوتا مساوی استفاده میکنید : == مثلا  
 دو تا مساوی فقط **برابری مقدار** رو بررسی میکنه

Let x = 5

Let y = 7

Console.log( x == y ) 🡺 false

واسه بررسی مساوی نبودن مقدار بالا از != استفاده میشه که نتیجه بالا میشه true   
  
واسه اینکه علاوه بر مقدار – تایپ هم در نظر گرفته بشه از سه تا مساوی استفاده میکنیم === و توصیه میشه همیشه از سه تا مساوی استفاده بشه. واسه مخالف هم میشه !==

از عملگر های > و < و >= و <= هم واسه مقایسه استفاده میشه

۴ . logical operators:  
مثل && و || که در واقع میاد چندین عملگر انتسابی رو با هم مقایسه میکنه .

۵. string Operators:

عملگر هایی که بر روی رشته ها کار میکنند. عملگر + که میاد دو تا رشته رو بهم میچسبونه :

const firstName = 'Parisa';

const lastName = 'Fallahi';

const result = firstName + '' + lastName;

console.log(result)

۶ . unary operators:   
عملگر های یکانی :  
واسه اینکه “5” رو به عدد تبدیل کنیم باید یک + قبلش بذاریم :

به این میگن unary plus :

const firstName = 'Parisa';

const lastName = 'Fallahi';

const result = firstName + '' + lastName;

console.log(result)

۷. nullish coalescing operator:  
  
که با ?? نمایش داده میشود. و بین این عملگر دو تا عبارت قرار میگرید و کاربردش اینه که میاد بین این دو عبارت اونی رو برمیگردونه که null یا undefined نباشه

let value1;

const value2 = 3;

console.log(value1 ?? value2)

نتیجه کد بالا میشه ۳ یعنی اولین مقداری رو که null یا undefined نبوده رو نشون داده.

۸. ternery operator یا عملگر های سه گانه

const x = 5;

const y = 6;

const result = x>y ? 'yes' : 'no'

مقادیر truthy و falsy:  
  
مقدار ۰ falsy است

رشته ها همگی truthy هستند

رشته خالی ولی falsy است.

Null مقدارش falsy است

Undefined هم مقدارش falsy است

آرایه چه خالی و چه پر truthy است

آبجکت هم مثل اریه در هر صورت truthy است

NaN چیست ؟

مثلا جواب زیر میشه NaN:

console.log('test'/2)

متد های toString() و toFixed() :

واسه تبدیل به رشته از toString استفاده میشه.

متد toFixed میاد یه عدد اعشاری رو میگیره و استرینگ برمیگردونه و به تعداد اعدادی که داخل پرانتر میذاریم تا اون رقم اعشار میاد نشون میده.

متد Number:  
واسه تبدیل به عدد کردن :  
|مثل عملگر + کار میکند :

const str = "50";

const num = +str

const num2 = Number(str)

console.log(num2)

متد parsInt:

دقیقا مثل متد Number است اما با این تفاوت که اگر بهش عدد اعشاری بدیم عدد بدون اعشار بهمون میده برای اینکه عدد اعشاری حفظ بشه از متد parseFloat() استفاده میشود.

بررسی backslash در string

برای نمایش تکست Hi, I’m Parisa ما به روش زیر عمل میکنیم:

const str = "Hi, I'm Parisa";

اما یک روش ذیگر هم وجود دارد و آن اینکه میتوانیم به این صورت با استفاده از backslash بنویسیم:

const str = "Hi, I\'m Parisa";

حالا اگر بخوایم یک متنی را توی چندین خط بنویسم هم باز میتونیم با backslash و n این کار را انجام بدیم:

const str = "Hi, I'm Parisa, \n I came from Iran";

برای پیدا کردن طول تکست از length استفاده میکنیم که میاد کاراکتر ها و فضاهای خالی رو در اون تکست میشمارد و به ما مدهد.

const str = "Hi, I'm Parisa, \n I came from Iran";

console.log(str.length)

Index در string

هر کاراکتر رشته یک ایندکسی دارد که از ۰ شروع میشود. مثال :

Const text = “Test”;

Console.log(text[0]) => T

به جای عبارت بالا هم میتونیم از متد charAt(0) هم استفاده کنیم. که میاد در واقع کاراکتر واقع در ایندکس ۰ را برمیگرداند.

برعکسش هم هست که اگر ما بخواهیم که ایندکس یه کاراکتری را بهمون برگردونه از متد indexOf استفاده میکنیم:

Const text = “Test”;

Console.log(indexOf(“e”)) => 1

متد includes() :

واسه بررسی اینکه آیا یک دسته کاراکتر مورد نظر ما در متن وجود دارد یا نه.

const str = "parisa is a good developer";

console.log(str.includes("is"))

در صورتیکه مطابقت داشته باشد true و در غیر این صورت false می شود.

متد toLowerCase() :   
برای کوچک کردن تمام کاراکتر های یک رشته استفاده میشود. و برعکس آن متد toUpperCase()

متد split () :

رشته را براساس کاراکتری که به عنوان ورودی بهش میدیم میاد تفکیک میکنه و نتیجه را در قالب آرایه برمیگرداند.

const str = "parisa is a good developer";

console.log(str.split(" "))

متد concate() :

دو رشته را به همدیگر متصل میکند.

const str1 = "parisa is a good developer";

const str2 = "and a teacher";

const finalText = str1.concat(" ",str2, "  and painter" )

console.log(finalText)

متد trim():

فضاهای خالی اول و اخر یک رشته را حذف میکند.

const str1 = "  parisa is a good developer";

const finalText = str1.trim();

console.log(finalText)

متد slice() :

به کمک این متد میتونیم یه slice یا یک تکه از رشته را جدا کنیم فقط کافیه ایندکس ابتدا و انتهای اون قسمت مورد نظر را به متد بدهیم. ایندکس انتها رو از ایندکس واقعی همیشه یدونه ببیشتر بده. مثلا تو مثال زیر ایندکس انتهای تکه ی مورد نظر میشه ۵ ولی ما ۶ میدیم که اخرین کاراکتر رو جا نندازه. چون آخرین ایندکس رو در نظر نمیگیره و یکی مونده به اخری رو در نظر میگیره.

const str1 = "parisa is a good developer";

const finalText = str1.slice(0,6);

console.log(finalText)

اگر فقط یک ورودی به متد slice() بدهیم از اون ایندکس تا اخر رشته رو برمیگردونه .

اگر عدد منفی بدیم بهش از اخر شروع به شمردن میکند مثلا slice(-4) میاد از developer و از r به عنوان اولین ایندکس یعنی ۱ در نظر میگیرد. دقت کنید که وقتی از اخر میاد ایندکس از ۱ شروع میشه.

متد substring() :

دقیقا عین متد slice() است با این تفاوت که دیگه اعداد منفی قبول نمیکند.

متد substr():

یک متد deprecate شده است. که دو تا عدد میگیره اولین عدد ایندکس است و دومی میگه از اون ایندکس چنتا بری جلو.

substr(3,5) یعنی از ایندکس ۳ شروع کن و پنج تا برو جلو و نتیجه رو نشون بده.

تمام متد های مورد استفاده بر روی رشته در جدول زیر آورده شده است.

**متد های مورد استفاده در رشته ها :**

|  |  |
| --- | --- |
| متد | توضیحات |
| toString() | واسه تبدیل به رشته از toString استفاده میشه. |
| toFixed() | یه عدد اعشاری رو میگیره و استرینگ برمیگردونه و به تعداد اعدادی که داخل پرانتر میذاریم تا اون رقم اعشار میاد نشون میده. |
| Number() | تبدیل به عدد میکنه |
| parsInt() | دقیقا مثل متد Number است اما با این تفاوت که اگر بهش عدد اعشاری بدیم عدد بدون اعشار بهمون میده |
| parseFloat() | دقیقا عین متد parsInt اما اگر عدد اعشاری بگیره اعشارش رو نگه میداره |
| charAt(2) | کاراکتر واقع در ایندکس ۲ را برمیگرداند |
| indexOf(“s”) | ایندکس مربوط به کاراکتر s را برمیگرداند. |
| Includes(“AM”) | برای بررسی اینکه یک دسته کاراکتر مورد نظر در رشته وجود دارد یا نه که نتیجه به صورت Boolean میباشد |
| toLowerCase() | برای کوچک کردن تمام کاراکتر های یک رشته استفاده میشود. |
| toUpperCase() | برای بزرگ کردن تمام کاراکتر های یک رشته استفاده میشود. |
| split(“ ”) | رشته را براساس کاراکتری که به عنوان ورودی بهش میدیم میاد تفکیک میکنه و نتیجه را در قالب آرایه برمیگرداند. |
| concate() | دو رشته را به همدیگر متصل میکند. |
| trim() | فضاهای خالی اول و اخر یک رشته را حذف میکند. |
| slice(4,7) | یه slice یا یک تکه از رشته را جدا میکند فقط کافیه ایندکس ابتدا و انتهای اون قسمت مورد نظر را به متد بدهیم. ایندکس انتها رو از ایندکس واقعی همیشه یدونه ببیشتر بده. چون آخرین ایندکس رو در نظر نمیگیره و یکی مونده به اخری رو در نظر میگیره. و اعداد منفی هم قبول میکند و در صورتیکه عدد منفی بهش بدی از اخر رشته شروع میکند. |
| substring() | دقیقا عین متد slice() است با این تفاوت که دیگه اعداد منفی قبول نمیکند. |
| substr(3,5) | یک متد deprecate شده است. که دو تا عدد میگیره اولین عدد ایندکس است و دومی میگه از اون ایندکس چنتا بری جلو.  substr(3,5) یعنی از ایندکس ۳ شروع کن و پنج تا برو جلو و نتیجه رو نشون بده |

استفاده از template literals :

این قابلیت از es6 به بعد اضافه شده است. برای اینکه دو رشته را به هم بچسبونیم با استفاده از + یا concat این کار را میکردیم اما یک روش خیلی راحت تر استفاده از روش template literals می باشد:

const name = 'test';

const age = 30;

console.log(`my name is ${name} and I am ${age} years old`)

**آرایه ها :**   
  
دو روش برای تعریف آرایه در جاوا اسکریپت وجود دارد:

const firstArray = [1,2,3];

const secondArray = new Array(1,2,3)

داخل یک آرایه هر چیزی میشه گذاشت. حتی یک ارایه دیگه .اگه یک ارایه دیگه توی ارایه بذاریم میشه nested array یعنی ارایه های تو در تو.

در ارایه هم مثل رشته ها ما ایندکس داریم که از ۰ شروع میشود. پس ترتیب در ارایه مهمه درست عین رشته ها   
واسه بررسی طول آرایه از length استفاده میکنیم.

const firstArray = [1,2,3,'parisa','fallahi'];

console.log(firstArray.length)

برای دسترسی به اعضای ارایه هم میتوانیم به صورت زیر عمل کنیم و با ایندکس اون عضو بهش دسترسی داشته باشیم:

const firstArray = [1,2,3,'parisa','fallahi'];

console.log(firstArray[2])

برای اضافه کردن آیتم جدید به ارایه میتونیم از push و unshift استفاده کنیم . که push میاد به انتهای ارایه اضافه میکنه و در نتیجه ایندکس قبلیا تغییری نمیکنه:

const firstArray = [1,2,3,'parisa','fallahi'];

firstArray.push(52);

console.log(firstArray);

با unshift هم به ارایه اضافه میشه ولی ترتیب بقیه رو بهم میزنه:

const firstArray = [1,2,3,'parisa','fallahi'];

firstArray.unshift(52);

console.log(firstArray);

pop() از اخر ارایه یکی کم میکنه

const firstArray = [1,2,3,'parisa','fallahi'];

firstArray.pop()

console.log(firstArray);

shift() از اول ارایه میاد کم میکنه

const firstArray = [1,2,3,'parisa','fallahi'];

firstArray.shift()

console.log(firstArray);

متد toString() در ارایه:   
برای تبدیل ارایه به رشته استفاده میشود. و نتیجه باید حتما در یک متغیر جدید ریخته شود و اون متغییر مربوط به ارایه را تغییری نمیدهد.

const firstArray = [1,2,3,'parisa','fallahi'];

const res = firstArray.toString()

console.log(res); ---------> 1,2,3,parisa,fallahi

دستور delete در ارایه :  
  
اگر بخواهیم یک آیتم را از ارایه حذف کنیم از این دستور استفاده میکنیم: حذف کردن با این روش ترتیب ایندکس ها را بهم نمیریزد چون به جای آن آیتم حذف شده empty قرار میدهد.

const firstArray = [1,2,3,'parisa','fallahi'];

delete firstArray[3]

console.log(firstArray); -----> parisa is deleted

اگر بخواهیم یک آیتم در ارایه را تغییر دهیم با استفاده از ایندکس ها این کار را انجام میدهیم:

const firstArray = [1,2,3,'parisa','fallahi'];

firstArray[1] = 'test'

console.log(firstArray);

متد join() در ارایه :  
  
این متد اگر بدون پارامتر استفاده شود دقیقا عین متد toString() است اما در غیر این صورت اگر بهش مقداری بدیم میاد ارایه رو تبیدل به رشته میکنه و با استفاده از پارامتری که به عنوان ورودی بهش دادیم ایتم های ارایه را به هم وصل میکند :

const firstArray = [1,2,3,'abc','fty'];

const res = firstArray.join(" and ")

console.log(res); -----------> 1 and 2 and 3 and abc and fty

متد concat() در ارایه:  
  
برای ترکیب کردن دو آرایه استفاده میشود.

const firstArray = [1,2,3];

const secondArray = ['a','b','c']

const result = firstArray.concat(secondArray);

console.log(result); ----->  [ 1, 2, 3, "a", "b", "c" ]

متد flat() در ارایه :  
  
برای ارایه های تودر تو استفاده میشود. در صورتیکه داخل یک ارایه از ارایه ای دیگر به صورت تودر تو استفاده شده باشد میتوانید با استفاده از متد flat() حالت تودرتویی را بردارید:

const firstArray = [1,2,3,['a','c'],['b','m',4]];

const result = firstArray.flat()

console.log(result); ----->  [ 1, 2, 3, "a", "c", "b", "m", 4 ]

متد indexOf() در آرایه:  
  
اگر بخواهیم ایندکس یک آیتم از ارایه را پیدا کنیم از این متد استفاده میکنیم اگر بخواهیم ایندکس عضوی را بخواهیم که وجود نداشته باشد نتیجه میشه -1 مثلا میتونی تو شرط ها از این روش استفاده کنی که اگر مثلا ایندکس شد -۱ که یعنی اون ایتم وجود نداشت بیا یه کاری بکن ....

const firstArray = [1,2,3];

console.log(firstArray.indexOf(2)) ----> 1

متد includes() در آرایه:  
برای بررسی وجود یک آیتم در یک آرایه استفاده میشود که نتیجه Boolean است

const firstArray = [1,2,3];

console.log(firstArray.includes(3)) ----> true

متد slice() در آرایه   
دقیقا عین slice که در string ها استفاده میشد

const firstArray = [1,2,3,'a','b','c'];

const slicedArray = firstArray.slice(0,3);

console.log(slicedArray); --- > [1,2,3]

متد splice() در ارایه :  
  
وقتی بخواهیم یه بخشی از آرایه رو حذف کنیم پارامتر اول میشه ایندکس شروع و دومی تعدادی که میخوایم حذف کنیم رو میدیم.

اگر تعداد رو بهش ندیم میاد از همون ایندکس به بعد همه رو حذف میکند.

const firstArray = [1,2,3,'a','b','c'];

const spliceddArray = firstArray.splice(1,1);

console.log(firstArray) ---> [1,3,'a','b','c']

const firstArray = [1,2,3,'a','b','c'];

const spliceddArray = firstArray.splice(1,3);

console.log(firstArray) ----> [1,'b','c'];

به splice میتونیم مقدار منفی هم بدیم :

const firstArray = [1,2,3,'a','b','c'];

const spliceddArray = firstArray.splice(-4,2);

console.log(firstArray) ----> [ 1, 2, "b", "c" ]

متد splice() یک کاربرد دیگری هم دارد و آن اینکه میتونیم در ایندکس مور نظرمون یک آیتم اضافه کنیم به این صورت میشه:

const firstArray = [1,2,3,'a','b','c'];

const spliceddArray = firstArray.splice(3,0,'parisa')

console.log(firstArray) ----> [ 1, 2, 3, "parisa", "a", "b", "c" ]

یعنی برو تو ایندکس ۳ ایتم parisa رو اضافه کن و چون قرار نیست چیزی رو حذف کنیم پس تعداد delete رو هم ۰ میدهیم. که البته میتونیم delete رو هم تعداد بدیم. که در اون صورت میاد حذف هم برامون انجام میده

متد sort() در آرایه:  
برای مرتب کردن ایتم های آرایه استفاده میشود. که اگر آیتم ها کاراکتر باشند (مثل اسامی) براساس ترتیب حروف الفبای ابتدای کاراکتر مرتب میشود و اگر اعداد باشد هم مرتب سازی براساس کوچیکترین عدد خواهد بود اگر عدد دو رقمی باشد مرتب سازی براساس رقم اخر عدد انجام میشود:

const firstArray = [1,2,3,11,21,5];

const sortedArray = firstArray.sort()

console.log(sortedArray) ---> [ 1, 11, 2, 21, 3, 5 ]

متد reverse()

ترتیب آیتم های آرایه را برعکس میکند

const firstArray = [1,2,3,11,21,5];

const reversedArray = firstArray.reverse()

console.log(reversedArray) ---> [ 5, 21, 11, 3, 2, 1 ]

|  |  |
| --- | --- |
| متد | توضیحات |
| Push() | برای اضافه کردن آیتم جدید به انتهای آرایه که در نتیجه آن ایندکس آیتم های قبلی تغییری نمیکند. |
| Unshift() | برای اضافه کردن آیتم جدید به انتهای آرایه است اما ترتیب بقیه آیتم ها را به هم میزند. |
| Pop() | از انتهای آرایه یک آیتم کم می کند |
| Shift() | از ابتدای آرایه یک آیتم کم میکند. |
| toString() | برای تبدیل ارایه به رشته استفاده میشود. و نتیجه باید حتما در یک متغیر جدید ریخته شود و اون متغییر مربوط به ارایه را تغییری نمیدهد. |
| Delete testArray[2] | اگر بخواهیم یک آیتم را از ارایه حذف کنیم از این دستور استفاده میکنیم: حذف کردن با این روش ترتیب ایندکس ها را بهم نمیریزد چون به جای آن آیتم حذف شده empty قرار میدهد. |
| testArray.join(“ and ”) | این متد اگر بدون پارامتر استفاده شود دقیقا عین متد toString() است اما در غیر این صورت اگر بهش مقداری بدیم میاد ارایه رو تبیدل به رشته میکنه و با استفاده از پارامتری که به عنوان ورودی بهش دادیم ایتم های ارایه را به هم وصل میکند : |
| testArray1.concat(testArray2) | برای ترکیب کردن دو آرایه استفاده میشود. |
| Flat() | برای ارایه های تودر تو استفاده میشود. در صورتیکه داخل یک ارایه از ارایه ای دیگر به صورت تودر تو استفاده شده باشد میتوانید با استفاده از متد flat() حالت تودرتویی را بردارید: |
| indexOf() | اگر بخواهیم ایندکس یک آیتم از ارایه را پیدا کنیم از این متد استفاده میکنیم اگر بخواهیم ایندکس عضوی را بخواهیم که وجود نداشته باشد نتیجه میشه -1 مثلا میتونی تو شرط ها از این روش استفاده کنی که اگر مثلا ایندکس شد -۱ که یعنی اون ایتم وجود نداشت بیا یه کاری بکن .... |
| Includes() | برای بررسی وجود یک آیتم در یک آرایه استفاده میشود که نتیجه Boolean است |
| Slice() | دقیقا عین slice که در string ها استفاده میشد |
| Splice() | وقتی بخواهیم یه بخشی از آرایه رو حذف کنیم پارامتر اول میشه ایندکس شروع و دومی تعدادی که میخوایم حذف کنیم رو میدیم.  اگر تعداد رو بهش ندیم میاد از همون ایندکس به بعد همه رو حذف میکند.  به splice میتونیم مقدار منفی هم بدیم  متد splice() یک کاربرد دیگری هم دارد و آن اینکه میتونیم در ایندکس مور نظرمون یک آیتم اضافه کنیم |
| Sort() | برای مرتب کردن ایتم های آرایه استفاده میشود. که اگر آیتم ها کاراکتر باشند (مثل اسامی) براساس ترتیب حروف الفبای ابتدای کاراکتر مرتب میشود و اگر اعداد باشد هم مرتب سازی براساس کوچیکترین عدد خواهد بود اگر عدد دو رقمی باشد مرتب سازی براساس رقم اخر عدد انجام میشود |
| Reverse() | ترتیب آیتم های آرایه را برعکس میکند |

انواع حافظه

دو مدل مموری داریم stack memory و heap memory

میدانیم که اطلاعات ما مثل متغیر هایی که در جاوا اسکریپت داریم داخل یک سخت افزاری به اسم مموری ذخیره میشود. و ما دو مدل مموری داریم که در بالا اسامی آن ها آورده شده است

Stack memory: سرعت خیلی بالایی دارد و دیتا به سرعت در آن ذخیره و بازیابی میشود. اما محدودیت حافظه دارد و حجم اش کم است. و هر دیتایی رو نمیشه داخلش ذخیره کرد چون سریع پر میشه. و ترتیب ذخیره سازی پشت سرهم و با ترتیب خاصی انجام میشود

Heap memory: سرعتش نسبت به stack پایین تره اما حجم حاظه بیشتری دارد و دیتای بیشتر رو میشه توش ذخیره کرد. و دیتا ها با ترتیب خاصی در heap ذخیره نمیشوند چون هر دیتا یک آدرس یا reference دارد.

مفهوم primitive type و reference type:

به طور کلی تمام دیتا تایپ های جاوا اسکریپت در دو گروه تقسیم میشوند که primitive و reference هستند.

Primitive types: string – number –boolean – null – undefined - symbol

Reference types: array – object- function

وقتی در جاوا اسکریپت متغیری را تعریف میکنیم جاوا اسکریپت این دیتا را داخل stack memory ذخیره میکند.

وقتی که از نوع primitive باشد داخل stack ذخیره میشه چون حجم کمتری را اشغال میکنند.

مثلا اگر داشته باشی let x = 5 میاد اینو تو استک ذخیره میکنه حالا اگه بنویسی let y = x میاد y = 5 رو بعد از x=5 در استک ذخیره میکند حالا اگر بعد از let y = x بنویسی x=10 فقط میاد اون بخش از حافظه ی مربوط به x رو تغییر میده و y رو تغییر نمیده چون جاوا اسکریپت به صورت خطی کامپایل می شود.

Reference type:

ارایه و آبجکت حجم زیادی میگیرند و در stack ذخیره نمی شوند. اگر بخوای متغیر و دیتای توش رو توی استک ذخیره کنی که سریع پر میشه.

نحوه ی ذخیره reference type ها به این صورت هستش که مقدار متغیر مورد نظر در حافظه ی heap ذخیره میشود و به آن یک reference code ای هم اختصاص داده میشود.

در حافظه ی stack هم نام متغیر به همراه reference code ذخیره می شود. مثلا اگر کد ریفرنس ۱۲۳ باشه در stack به این صورت ذخیره می شود: x = 123

حالا اگر داشته باشیم y = x مقدار y = 123 به همان ریفرنس اشاره میکنه. چرا؟ چون مقدار x داخل stack هستش که اونم مقدارش یه ریفرنس کد هستش.

Garbage collector:

قبلا گفتیم که ذخیره متغیر های reference type به این صورت هستش که دیتا داخل heap memory ذخیره می شود و به آن یک reference code اختصاص می یابد و نام متغیر به همراه reference code در داخل stack memory ذخیره می شود.

حالا اگر به ازای هر دیتایی که داخل heap هستش ریفرنسی در stack نباشد یعنی یک سری دیتای اضافه در حافظه ی heap داریم که هیچ متغیری به آن ریفرنس داده نشده است.

Garbage collector کارش حذف کردن دیتاهای اضافه در heap memory است و توسط موتور جاوا اسکریپت که در داخل تمام مرورگر ها وجود دارد به صورت دیفالت همیشه اجرا میشود.

مثلا فرض کن که let x = [1,2,3] رو داری ولی بعد از مدتی دیگه این متغیر دیگه لازم نداری و کارت باهاش تموم شده در این صورت میتونی x = null بذاری که به garbage collector بفهمونی که دیگه هیچ ریفرنسی از این متغیر در stack وجود ندارد و در نتیجه دیتای آن را از heap memory حذف کند. در نتیجه برنامه با سرعت و performance بالاتری اجرا خواهد شد.

اگر دو متغیر داشته باشیم که هر دو مقدار آرایه یکسانی داشته باشند. این دو باهم برابر نیستند. :

const x = [1,2,3];

const y = [1,2,3];

console.log(x === y) ----> false

علت هم این است که : ریفرنس کد مربوط به x با ریفرنس کد مربوط به y در حافظه ی stack متفاوت است و به همین دلیل برابر نیستند. با اینکه دیتای یکسانی دارند اما در حافظه ی heap دو جایگاه متفاوت را اشغال میکنند. اما اگر بذاری y = x در این صورت این مقایسه true میشود چون در این صورت هر دو متغیر به یک ریفرنس اشاره میکنند. پس دو ارایه یا دو آبجت زمانی با هم برابر میشوند که به یک ریفرنس اشاره داشته باشند.

اما اگر بخواهیم دو متغیر primitive را مقایسه کنیم که مقدار یکسانی دارند چون مقدارشون در stack memory یکسان هستند بنابراین در مقایسه با هم برابر می شوند. چون مقدارشون و تایپشون باهم برابر است.

const x = 1;

const y = 1;

console.log(x === y) ----> true

mutation

به قابلیت تغییر ویژگی های یک مقدار گفته میشود

A mutation is the act of changing the properties of a value

تمام primitive value ها در جاوا اسکریپت immutable هستند. یعنی نمیتونی پراپرتی های یک value مثلا ایندکس ۲ یا هر ایندکسی از یک string رو تغییر بدی یا مقدار یک متغیر عددی رو تغییر بدی

اما reference type ها کثل ارایه و آبجکت mutable و قابل تغییر هستند به این معنا که میتوانیم مثلا مقادیر داخل یک ارایه یا آبجکت رو تغییر بدهیم.

تفاوت shallow copy و deep copy:

یه موقعی هست که ما نیاز داریم که مقدار یک متغییری رو یه جایی سیو داشته باشیم یا در واقع یک کپی از مقدار متغیر مورد نظرمون داشته باشیم.

این کار واسه متغیر های از نوع primitive که اسونه چون قبلا تو بحث stack و heap مموری در مورد این تغییرات گفته بودیم.

به عنوان مثال اگر متغیر x یا مقدار ۵ رو داشته باشیم و بخوایم یک کپی ازش بگیریم که وقتی بعدا مقدارش به ۱۰ تغییر کرد ما مقدار قبلیو داشته باشیم به صورت زیر عمل میکنیم:

let x = 5;

const copyX = x;

x = 10;

console.log(x,copyX);

که در این صورت مقدار x به ۱۰ تغییر میکند اما copyX همچنان ۵ خواهد ماند.

برای متغیر های reference type یکم فرق میکنه و میدونیم که به سادگی روش بالا نمیشه آنها را کپی کرد. و اگر مثل روش بالا بخواهیم کپی کنیم چون هر دو به یک ریفرنس اشاره میکنند با تغییر در دیتا هر دو x و copyX هم تغییر خواهد کرد و در واقع کپی مفهومی نخواهد داشت! برای کپی کردن متغیر های reference type مثلا آرایه که بخواهیم ریفرنس اش هم متفاوت بشه می بایست از روش slice() استفاده کرد. Slice در واقع ارایه اصلی رو mutate نمیکرد یعنی تغییر نمیداد و توسط این دستور میتونیم ارایه را کپی کنیم و در یک متغیر جدید و در نتیجه ریفرنس جدید در حافظه ذخیره کنیم.

const data = [1,2,3];

const copyData = data.slice();

data.push(4);

console.log("data=", data); ---> [ 1, 2, 3, 4 ]

console.log("copyData", copyData) --->  [ 1, 2, 3 ]