۷۰

سوال مصاحبه جاوااسکریپت با توضیح اختصاصی

www.ditty.ir

دیتی

سلام دوستان! 👋

سلام دوستان! ۷۰ سوال جدید و بروز که توی مصاحبه‌های جاوااسکریپت پرسیده میشه رو می‌تونین توی این PDF ببینین. این PDF همون مجموعه پست‌هایی هست که قبلاً توی وبسایت دیتی منتشر شده بود. امیدوارم استفاده کنین، توی کارتون پیشرفت کنین و توی بهترین جاها فعالیت کنین.

نکته: برای یادگیری کامل و جزئی بعضی از این موارد نیازمند یک مقاله اختصاصی هست. برای مواردی که اینجا بررسی می‌کنیم (هر چند خلاصه هستن)، تا حد امکان سعی می‌کنم کامل و واضح توضیح بدم که اگه قصد یادگیری هم داشته باشین، نیاز شما رو برطرف کنه.

خب، بریم که سوالا رو بررسی کنیم 🔥

1. تفاوت بین undefined و null چیه؟

قبل از اینکه درباره تفاوت‌های این دو صحبت کنیم، بهتره که بدونیم این دو جزو ۸ نوع داده‌ای هستن که توی جاوا اسکریپت وجود دارن:

```
['null','undefined', 'string', 'number', 'boolean',
'object', 'symbol', 'bigint'];
```

همچنین این دو به اصطلاح falsy value هستن. یعنی مقدارهایی که وقتی اونها رو به Boolean تبدیل می‌کنیم، خروجی `false` هست:

```
console.log(!!null); // false
console.log(!!undefined); // false

console.log(Boolean(null)); // false
console.log(Boolean(undefined)); // false
```

و اما تفاوت‌ها. از undefined شروع کنیم:

ویژگی‌های منحصر به فرد undefined:

- undefined یک مقدار پیشفرض برای متغیرهایی هست که مقدار ندارن.
  یعنی موقع ساختن این متغیر بهش مقدار داده نشده
- undefined یک خروجی پیشفرض هست برای تابعی که return نداره
- وقتی یک پراپرتی (یا key) از یک آبجکت رو صدا بزنیم که وجود نداره، خروجی undefined هست

```
let undefinedVar;
function noReturn() {

}

const alphabet = {
    a: "ay",
    b: "bee",
    c: "si"
};

console.log(undefinedVar); // undefined
console.log(noReturn()); // undefined
console.log(alphabet.d); // undefined

console.log(typeof undefinedVar); // undefined
```

ویژگی‌های منحصر به فرد null:

در واقع null خودش یک مقدار هست. یک مقدار "**پوچ**" یا "**خالی**" که می‌تونیم اون رو به متغیرها نسبت بدیم:

```
let x = null;

console.log(x); // null
console.log(typeof x); // object!
```

به این مقایسه‌ها دقت کنین:

```
null == undefined; // true
null === undefined // false
```

مقایسه اول true شد چون مقایسه این دو با `==` همیشه `true` هست.
مقایسه دوم برای این `false` شد که توی سه مساوی (===) تبدیل نوع انجام نمی‌گیره. نوع یک `null` برابر با آبجکت و نوع یک `undefined` همون `undefined` هست.

بیشتر بدانید

[ویدئو] تفاوت بین null و undefined چیه؟

با ویژگی‌های null و undefined و تفاوت بین اونها توی این قسمت آشنا می‌شیم

2. عملگر && چکار میکنه؟

عملگر && که AND منطقی گفته میشه بیشتر ماها با اون آشنایی داریم، اما شاید توضیح و کاربرد دقیقش رو ندونیم. دو یا چند عبارت رو درنظر بگیرید که ممکنه هر نوعی داشته باشن. مثلا رشته، عدد یا بولین. وقتی یک && بین دو یا چند عبارت قرار بگیره، بررسی می‌کنه که آیا همه‌ی این عبارت‌ها غیر false هستن یا نه و اگه هیچ کدوم از این مقادیر false نبودن، **مقدار آخرین عبارت** رو برمی‌گردونه. اما اگه یک کدوم از این عبارت‌ها falsy بودن، **مقدار اولین عبارت falsy** رو برمی‌گردونه. به بیان ساده‌تر:

> *AND منطقی مقدار اولین عبارت falsy رو برمی‌گردونه و اگه عبارتی falsy نبود، مقدار آخرین عبارت رو برمی‌گردونه.*

مثال‌های زیر رو در نظر بگیرید:

```
true && true && true; // true
true && true && false; // false
null && false && undefined; // null

"Hans" && "Alex" && "Ali"; // "Ali"
true && true && "Ali"; // "Ali"
```

خروجی خط ۱ و ۲ که واضح هستن. خروجی مثال سوم null شده. چون اولین عبارت falsy هست بین این عبارات. توی مثال چهارم و پنجم هیچ کدوم از عبارت‌ها falsy نیستن. پس مقدار آخرین عبارت که "Ali" هست برگردونده میشه.

بیشتر بدانید

[ویدئو] عملگر && چکار میکنه؟

با ویژگی‌های عملگر AND منطقی که به صورت && هست آشنا می‌شیم

3. عملگر || چکار میکنه؟

عملگر || یا OR منطقی اگه بین دو یا چند عبارت با نوع‌های گوناگون قرار بگیره، دنبال اولین عبارتی می‌گرده که truthy هست:

```
console.log(null || "Oh you again!" || undefined); // Oh
you again!
console.log(false || false || "Yaaay!"); // Yaaay!
```

بیشتر بدانید

[ویدئو] عملگر || چکار میکنه؟

با عملگر Logical OR و ویژگی‌های منحصر به فرد اون آشنا می‌شیم

4. سریع‌ترین راه تبدیل یک رشته به عدد چیه؟

راه‌های زیادی برای تبدیل یک رشته به یک عدد وجود داره. اما سریع‌ترین راه، استفاده از Unary plus (+) هست:

```
console.log(+ "29"); // 29 (typeof number);
console.log(+ "-29"); // -29 (typeof number);
```

بیشتر بدانید

[ویدئو] سریع‌ترین راه تبدیل یک رشته به عدد چیه؟

با عملگر Unary Plus که استفاده از اون سریع‌ترین راه تبدیل رشته به عدد توی جاوااسکریپت هست آشنا می‌شیم

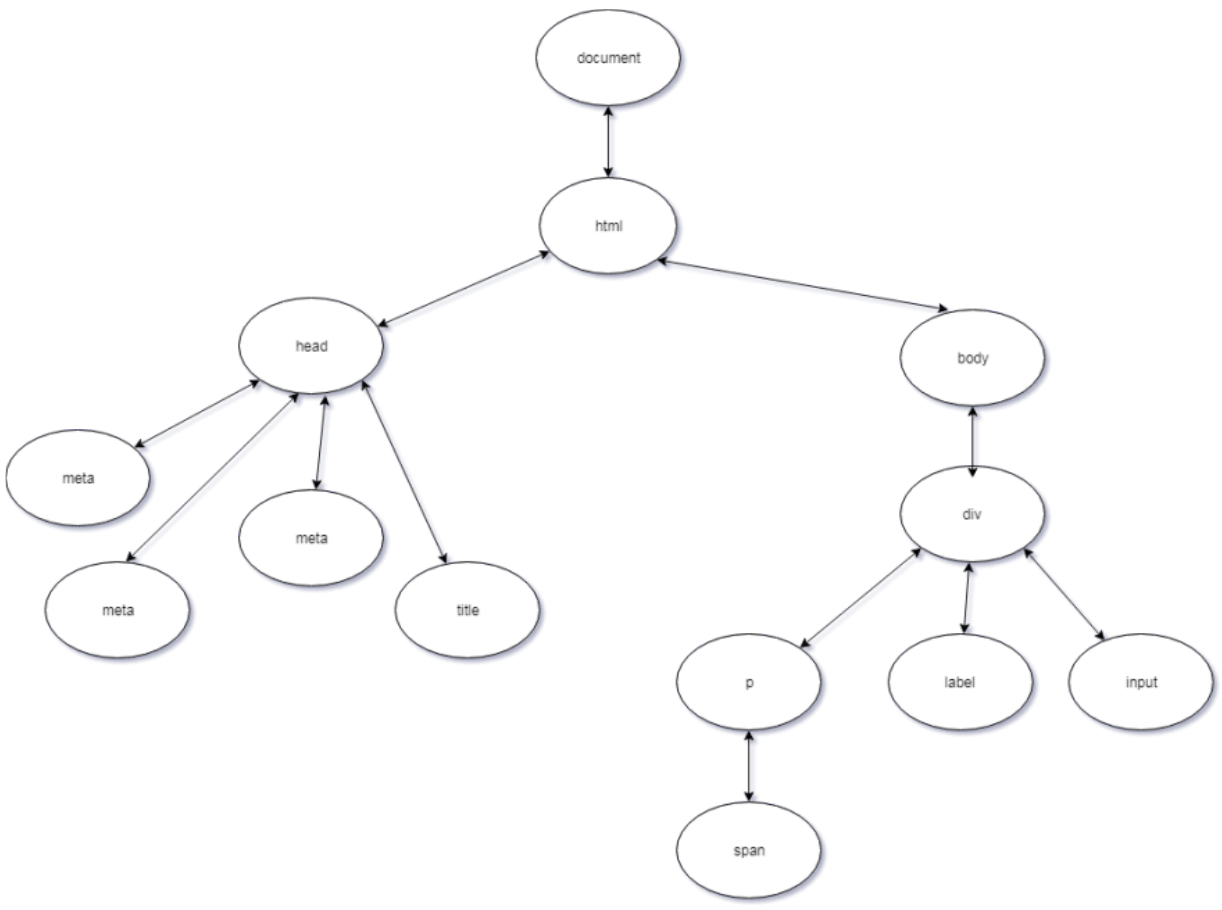
5. DOM چیه؟

> *DOM مخفف Document Object Model و یک آبجکت از تمام المنت‌های موجود در یک صفحه HTML هست.*

سند HTML زیر رو در نظر بگیرید:

```html
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Document Object Model</title>
</head>

<body>
   <div>
      <p>
         <span></span>
      </p>
      <label></label>
      <input>
   </div>
</body>

</html>
```

وقتی این صفحه توسط مرورگر تفسیر میشه، یک آبجکت جاوا اسکریپتی شامل همه‌ی المنت‌های موجود به همراه همه‌ی ویژگی‌های اونها (پراپرتی‌ها، رویدادها) درست میشه. یه چیزی مثل این:



به این مدل آبجکتی از سند میگن DOM.

حالا با استفاده از این آبجکت که DOM نام داره، می‌تونیم المنت‌های HTML رو ویرایش، حذف و اضافه کنیم، به اونها رویداد اضافه کنیم و ... .

بیشتر بدانید

[ویدئو] DOM چیه؟

با DOM که یک آبجکت سراسری توی جاوااسکریپت هست آشنا می‌شیم

6. Event Propagation چیه؟

کلمه Propagation یعنی گسترش و ترویج. پس Event Propagation یعنی یک رویداد (Event) چه جوری توی یک صفحه HTML گسترش پیدا میکنه. وقتی به یک المنت، یک رویداد مثلا onclick ضمیمه می‌کنیم، این رویداد فقط روی اون المنت اتفاق نمی‌افته. بلکه یک سری المنت‌های دیگه هم ممکنه تحت تاثیر این رویداد قرار بگیرن. اینجا یعنی این رویداد گسترش پیدا کرده.

ما دو نوع گسترش رویداد داریم: Capturing و Bubbling. هر دو مورد رو توی دو سوال زیر بررسی می‌کنیم.

بیشتر بدانید

[ویدئو] Event Propagation چیه؟

با مفهوم Event Propagation و انواع اون توی جاوااسکریپت آشنا می‌شیم

7. Event Bubbling چیه؟

یک راه برای گسترش یک رویداد توی یک DOM هست. کد HTML زیر رو در نظر بگیرید:

```
<div id="div">
    <p id="p"></p>
</div>
```

حالا می‌خوام به تگ‌های div و p یک مدیریت‌کننده رویداد onclick ضمیمه کنم:

```
document.querySelector('div');
  .addEventListener('click', function() {
    alert("I'm a div");
  });

document.querySelector('p');
  .addEventListener('click', function() {
    alert("I'm a p!");
  });
```

برای اجرا کردن این کد اینجا رو ببینید. وقتی این کد رو اجرا و روی تگ p کلیک کنیم، دو تا alert می‌گیریم. اولی پیغام "I'm a p" و سپس پیغام "I'm a div". این یعنی رویداد ابتدا از تگ داخلی که p باشه گسترش پیدا کرده به سمت بیرون. به

بیان ساده:

> *گسترش پیدا کردن رویداد توی این روش از المنت داخلی به سمت المنت بیرونی هست.*

این حالت از گسترش رویداد، حالت پیشفرض هست. چرا اسمش Bubbling هست؟ Bubble یعنی حباب. حباب‌ها هم معمولا از داخل بزرگ میشن به سمت بیرون.

بیشتر بدانید

[ویدئو] Event Bubbling چیه؟

با مفهوم Event Bubbling توی جاوااسکریپت آشنا می‌شیم

8. Event Capturing چیه؟

خب این روش برعکس روش بالا هست. یعنی رویداد ابتدا از تگ بیرونی شروع میشه و به تگ‌های داخلی میرسه. برای اینکار باید آرگومان سوم متد addEventListener رو برابر true قرار بدیم:

```
1  document.querySelector('div');
2    .addEventListener('click', function() {
3      alert("I'm a div");
4    }, true);
5
6  document.querySelector('p');
7    .addEventListener('click', function() {
8      alert("I'm a p!");
9    }, true);
```

خب اگه این کد رو اجرا کنیم، مسیر اجرا شدن رویداد برعکس میشه. یعنی اولین alert میگه "I'm a div" و سپس میگه "I'm a p". به بیان ساده‌تر:

> *گسترش پیدا کردن رویداد توی این روش از المنت بیرونی به سمت المنت داخلی هست.*

برای اجرا کردن این کد اینجا رو ببینید.

بیشتر بدانید

[ویدئو] Event Capturing چیه؟

با مفهوم Event Capturing توی جاوااسکریپت آشنا می‌شیم

9. فرق متدهای e.preventDefault() و e.stopPropagation() چیه؟

متد `event.preventDefault()` وقتی روی یک المنت اعمال میشه، مانع عملکرد ذاتی اون المنت میشه. مثلا وقتی رو یک form اعمال بشه، مانع ثبت شدن فرم میشه. وقتی روی یک تگ a اعمال بشه، باعث میشه اون لینک کار نکنه. برای اینکه متوجه بشید این متد چه‌جوری کار می‌کنه این مثال رو ببینید.

همونطور که توی سوال شماره 6 گفتم، وقتی یک رویداد روی یک المنت اعمال میشه، ممکنه المنت‌های دیگه هم تحت تاثیر این رویداد قرار بگیرن. متد `event.stopPropagation()` برای زمانی استفاده میشه که می‌خوایم یک رویداد به سطح‌های بعدی گسترش پیدا **نکنه**. برای اینکه بدونین این متد چه‌جوری کار می‌کنه این مثال رو ببینید.

بیشتر بدانید

[ویدئو] فرق متدهای preventDefault و stopPropagation چیه؟

با این دو متد آشنا می‌شیم و ویژگی‌های منحصر به فرد اونها رو بررسی می‌کنیم

10. چطوری میشه متوجه شد event.preventDefault() روی یک المنت اعمال شده؟

برای هر المنت یک پراپرتی وجود داره به اسم defaultPrevented که یک مقدار بولین هست. اگر مقدار اون true بود، یعنی روی این المنت، متد preventDefault اعمال شده:

```
<a id="anchor" href="#">The Slight Edge</a>

<script>
    var a = document.getElementById('anchor');
    a.addEventListener('click', function(event) {
        event.preventDefault();

        alert(event.defaultPrevented); // alerts true
    });
</script>
```

بیشتر بدانید

[ویدئو] چطوری ببینیم preventDefault روی یک المنت اعمال شده؟

با روشی آشنا می‌شیم که با اون می‌تونیم بررسی کنیم که آیا روی یک المنت متد preventDefault اعمال شده یا نه

. . .

Resources:

- https://www.w3schools.com/jsref/event_defaultprevented.asp
- https://stackoverflow.com/questions/4616694/what-is-event-bubbling-and-capturing
- https://dev.to/macmacky/70-javascript-interview-questions-5gfi
- https://stackoverflow.com/questions/5076944/what-is-the-difference-between-null-and-unde...
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Ope...
- https://stackoverflow.com/questions/4616694/what-is-event-bubbling-and-capturing

11. چرا کد زیر به خطا منجر میشه؟

```
const book = {};

console.log(book.page.number); // TypeError: book.page is undefined
```

ابتدا باید بدونیم که book.page خودش undefined هست. وقتی می‌خوایم یک پراپرتی که تو یک آبجکت وجود **نداره** رو صدا بزنیم، مقدار پیشفرض undefined اما بدون خطا رو می‌گیریم:

```
console.log(book.page); // undefined
console.log(book.gateau); // undefined
console.log(book.wheel); // undefined
console.log(book.sunflower); // undefined
```

خب نبود یک پراپرتی توی یک آبجکت، باعث نمیشه که برنامه مختل بشه. چون آبجکت‌ها ممکنه هنگام اجرا شدن برنامه بارها دستکاری بشن. مثلا پراپرتی‌ها اضافه و ویرایش بشن. و اگه بخاطر نبودن یک پراپرتی توی یک آبجکت روند اجرای برنامه مختل میشد، جاوا اسکریپت زبان قابل انعطافی به حساب نمی‌اومد!

و اما جواب این سوال که چرا کد اولی ارور میده (باعث میشه اجرای برنامه متوقف بشه)؟🤔
چون ما داریم از یک چیز undefined که page.book هست، یک پراپرتی دیگه رو فراخونی می‌کنیم. یعنی مثلا یک چیزی مثل این:

```
<undefined value>.property;
```

مشخصه که یک مقدار undefined نمیتونه شامل یک پراپرتی یا هر چیز دیگه‌ای باشه.

بیشتر بدانید

[ویدئو] چرا این کد به خطا منجر میشه؟

یک خطای رایج رو توی جاوااسکریپت بررسی می‌کنیم

12. event.target چیه؟

پراپرتی target از آبجکت event، شامل المنتی هست که باعث اجرای یک رویداد خاص شده. کد زیر رو ببینید:

```html
<div onclick="getElement(event)">
    <p>
        <button></button>
    </p>
</div>


<script>
function getElement(event) {
    alert(event.target.tagName);
}
</script>
```

همونطور که می‌بینید یک رویداد onclick به یک div نسبت داده شده. مقداری که event.target می‌گیره، اگه روی تگ div کلیک کنیم شامل تگ div، اگه روی p کلیک کنیم شامل تگ p و اگه روی button کلیک کنیم شامل تگ button خواهد بود.

این مثال عملی رو ببینید.

بیشتر بدانید

[ویدئو] event.target چیه؟

پراپرتی target از event شامل المنتی هست که باعث اجرای یک رویداد خاص شده

13. event.currentTarget چیه؟

پراپرتی currentTarget از آبجکت event شامل المنتی هست که رویداد به اون نسبت داده شده. همون کد بالا رو در نظر بگیرید. من تابع getElement رو بازنویسی می‌کنم:

```
function getElement(event) {
    alert(event.currentTarget);
}
```

خب اگه روی تگ‌های div و p و button کلیک کنیم، پراپرتی currentTarget شامل تگ div خواهد بود. چون ما رویداد رو به div نسبت دادیم.

بیشتر بدانید

[ویدئو] event.currentTarget چیه؟

پراپرتی currentTarget رو بررسی می‌کنیم و تفاوت اون با پراپرتی target رو بررسی می‌کنیم

14. فرق == و === چیه؟

این یکی از ساده‌ترین سوال‌ها هست اما یک سری نکاتی جالبی وجود داره. برای

مقایسه آیتم‌ها وقتی عملگر دو مساوی استفاده بشه، یک کار اضافه‌تر نسبت به سه مساوی انجام می‌گیره.

وقتی از دو مساوی استفاده می‌کنیم، عمل مقایسه مقدارها بعد از **تبدیل نوع موقت** صورت می‌گیره. برای مثال وقتی یک عدد و یک رشته رو با «دو مساوی» مقایسه می‌کنیم، ابتدا بطور موقت رشته تبدیل به عدد میشه و سپس مقایسه واقعی صورت می‌گیره:

```
let x = 29;
let y = "29";

x == y;

// 1. Convert y to a number
// 2. Compare 29 with 29
```

پس زمانی که از دو مساوی (==) استفاده می‌کنیم باید در نظر داشته باشیم که موارد زیر با هم مساوی هستن:

```
5 == '5'          // true
0 == ''           // true
0 == '0'          // true
0 == false        // true
```

اما توی مقایسه با سه مساوی هیچ تبدیل نوعی انجام نمی‌گیره و دو طرف مقایسه زمانی برابر هستن که نوع‌ها هم برابر باشن.

بهتره که سه مساوی رو ترجیح بدیم به دو مساوی. چون اولا با خروجی غیر منتظره کمتری رو به رو میشیم. مقایسه‌های مثال بالا اگه با سه مساوی انجام بشن همگی false هستن. و همچنین چون تبدیل نوعی انجام نمی‌گیره، سه مساوی سریع‌تر از دو مساوی عمل می‌کنه.

بیشتر بدانید

[ویدئو] عملگرها - قسمت 5 | عملگرهای مقایسه‌ای

خیلی وقت‌ها لازمه که دو مقدار رو با هم مقایسه کنیم. توی این قسمت یاد می‌گیریم که چطوری این کار رو توی جاوااسکریپت انجام بدیم

15. وقتی دو آبجکت مشابه رو با هم مقایسه می‌کنیم خروجی چیه؟

کد زیر رو در نظر بگیرید:

```
let alex = { hairs: true }
let john = { hairs: true }

console.log(alex == john); // false
```

خب اگه کد بالا رو اجرا کنیم خروجی false خواهد بود.

> *توی مقایسه‌ی دو آبجکت مشابه جواب همیشه false هست.*

چرا🤔 ؟ توی جاوا اسکریپت زمان مقایسه دو آبجکت با هم، رفرنس یا آدرس هر دو آبجکت توی حافظه مقایسه میشه، نه مقدار اونها. برای همین دو آبجکت ممکنه ظاهر یکسانی داشته باشن اما آدرس اونها توی حافظه با هم متفاوت هست.

چکار کنیم هنگام مقایسه دو آبجکت جواب true بگیریم؟ باید هر دو آبجکت به یک آدرس از حافظه اشاره کنن که با کد زیر امکان پذیر هست:

```
let fruit1 = { name: 'potato' };
let fruit2 = fruit1;

console.log(fruit2 === fruit1); // true
```

توی کد بالا و خط دوم وقتی که یک آبجکت رو برابر یک آبجکت دیگه قرار دادیم، در واقع آدرس حافظه متغیر fruit1 به متغیر fruit2 داده شد.

بیشتر بدانید

[ویدئو] وقتی دو آبجکت مشابه رو با هم مقایسه می‌کنیم خروجی چیه؟

توی این ویدئو یاد می‌گیریم که چرا دو آبجکت که ظاهر یکسانی دارن با هم برابر نیستن

16. عملگر !! چکار می‌کنه؟

عملگر Double NOT یا !! وقتی پشت یک مقداری قرار بگیره، اون رو به بولین تبدیل می‌کنه:

```js
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(!! []);        // true
console.log(!! 1);         // true
```

بیشتر بدانید

[ویدئو] عملگر !! چکار می‌کنه؟

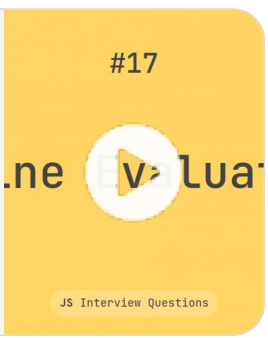
توی این ویدئو با عملگر Double NOT که به صورت !! هست آشنا می‌شیم

17. چه جوری توی یک خط یک عبارت رو محاسبه کنیم؟

چند عمل ضرب و تقسیم و ... رو می‌تونیم با کاما جدا کنیم تا توی یک خط ارزیابی بشن و برای اینکه اون رو به یک متغیر نسبت بدیم همه رو توی یک پرانتز قرار میدیم. مثل چیزی که تو خط آخر نوشتیم:

```js
function addFive(num) {
  return num + 5;
}

let x = 5;

x = (x++ , x = addFive(x), x *= 2, x -= 5, x += 10);
```

توی خط آخر عبارت‌ها از سمت چپ به راست محاسبه میشن و نهایتا خروجی که عدد ۲۷ هست به متغیر x نسبت داده میشه.



بیشتر بدانید

[ویدئو] چه جوری توی یک خط یک عبارت رو محاسبه کنیم؟

توی این ویدئو یاد می‌گیریم که چطوری توی یک خط یک عبارت جاوااسکریپتی رو محاسبه کنیم

18. Scope یا حوزه چیه؟

> *اسکوپ (Scope) یا حوزه، به جاهایی گفته میشه که متغیرها قابل دسترسی و استفاده هستن.*

سه نوع حوزه توی جاوا اسکریپت وجود داره:

- حوزه‌ی سراسری (Global Scope)
- حوزه‌ی محلی (Local Scope)
- حوزه‌ی بلاک (Block Scope)

حوزه‌ی سراسری

متغیرهایی که توی حوزه‌ی سراسری تعریف میشن توی همه جای برنامه قابل دسترسی هستن. توی مثال زیر متغیر name محصور به هیچ جایی نیست و بنابراین یک متغیر سراسری هست و همه جا حتی توی توابع هم قابل دسترسیه:

```
// Global
var name = "Michael";

function person() {
    console.log(name);
}

person(); // Michael
```

حوزه‌ی محلی

متغیرهایی که توی یک تابع تعریف میشن دارای حوزه‌ی محلی هستن و بیرون از تابع قابل دسترسی نیستن:

```
function person() {
    var name = "Michael";
}

console.log(name); // ReferenceError: name is not defined
```

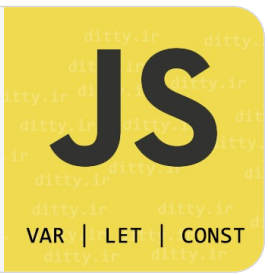
حوزه‌ی بلاک

متغیرهایی که با let و const توی فضای بین دو براکت { ... } که به اصطلاح بلاک گفته میشه، تعریف میشن، فقط توی همون بلاک قابل دسترسی هستن:

```
if (true) {
    let name = "Michael";
    console.log(name); // Michael
}

console.log(name); // ReferenceError: name is not defined
```

در واقع حوزه‌ای که یک بلاک درست می‌کنه یک حوزه‌ی محلی هست و متغیرها فقط توی همون قسمت قابل دسترسی هستن.

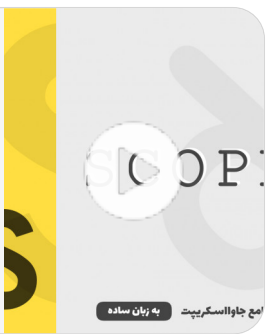
پیشنهاد می‌کنم مقاله‌های زیر رو بخونین:



بیشتر بدانید

تفاوت var و let و const در جاوا اسکریپت ES6

۳ راه مختلف تعریف متغیر توی جاوااسکریپت یعنی var و let و const رو توی این پست با هم بطور مفصل بررسی می‌کنیم



بیشتر بدانید

[ویدئو] اسکوپ یا Scope | آموزش جاوااسکریپت به زبان ساده

اسکوپ یا حوزه که یکی از مهمترین مفاهیم جاوااسکریپت هست و یادگیری اون برای هر برنامه‌نویسی یک قدم بزرگ توی یادگیری این زبان به حساب میاد

19. Hoisting چیه؟

معنای تحت‌الفظی کلمه Hoisting "بالا بردن" هست. توی جاوااسکریپت قبل از اینکه کدهای ما به شکل واقعی اجرا بشن، همه‌ی توابع و متغیرها به اول حوزه‌ی خودشون جابجا میشن. به همین دلیل هست که کد زیر بدون خطا کار می‌کنه:

```
smile();

function smile() {
    return ":)";
}
```

اول تابع رو صدا زدیم و پایین‌تر خود تابع رو تعریف کردیم.

همونطور که گفتم متغیرها به اول حوزه‌ی خودشون جابجا میشن. یعنی اگه یک متغیر توی حوزه‌ی سراسری تعریف شده باشه، به بالاترین قسمت کد جابجا میشه و اگه یک متغیر توی حوزه‌ی محلی تعریف شده باشه، به اول حوزه‌ی خودش جابجا میشه.

نکته: قبل از اجرای کد و وقتی که Hoisting انجام میشه، متغیرها، **بدون مقدار** به اول حوزه‌ی خودشون میرن. کد زیر رو درنظر بگیرید:

```
console.log(name);

var name = "Serena";
```

کد بالا در واقع به شکل زیر در نظر گرفته میشه:

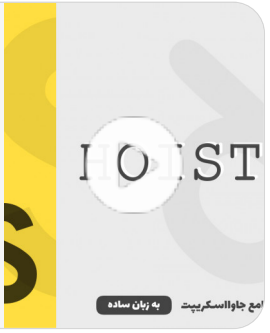
```
var name;

console.log(name);

name = "Serena";
```

خب احتمالا باید بدونیم که خروجی console.log برابر با undefined هست. به این دلیل که متغیر name تعریف شده ولی مقدار دهی نشده.

نکته: مثال بالا برای متغیرهایی بود که با var تعریف شدن. متغیرهایی که با let و const تعریف میشن هم شامل Hoisting میشن. اما عملکرد اونها با var متفاوت هست. وقتی متغیرها با let و const ساخته میشن، قبل از استفاده از اونها باید تعریف شده باشن. در غیر این صورت کد با خطا مواجه میشه:

```
let me = "know";
console.log(me); // know

console.log(puppy); // ReferenceError: Cannot access
'puppy' before initialization
let puppy = 30;
```

برای آشنایی کامل با Hoisting پست زیر رو ببینید:



بیشتر بدانید

[ویدئو] مفهوم Hoisting | آموزش جاوااسکریپت به زبان ساده

توی این قسمت می‌خوایم یکی از مفاهیم اساسی جاوااسکریپت یعنی Hoisting رو بررسی کنیم

20. use strict چکار می‌کنه؟

کلمه strict به معنی *سخت‌گیرانه* هست. وقتی توی جاوا اسکریپت از این دستور استفاده می‌کنیم برنامه‌ی وارد حالت سخت‌گیرانه میشه. به قول معروف Strict Mode. استفاده از این دستور باعث میشه تا کدهایی امن‌تر و با باگ‌های کمتری بنویسیم. یکی از سخت‌گیری‌هایی که اعمال میشه موقع استفاده از متغیرهایی هست که تعریف نشدن:

```
"use strict";
x = 3.14; // ReferenceError: assignment to undeclared
variable x
```

خب کد بالا با استفاده از use strict باعث بروز خطا میشه و میگه به متغیر x داریم مقدار می‌دیم درحالی که تعریف نشده. اما این کد بدون استفاده از use strict بدون مشکل اجرا میشه. خب برای اصلاح این کد باید اون رو به شکل زیر بنویسیم:

```
"use strict";
var x = 3.14;
```

عبارت use strict رو همیشه اول اسکریپت یا اول توابع می‌نویسیم:

```
x = 3.14;       // No error
myFunction();

function myFunction() {
    "use strict";
    y = 3.14;   // ReferenceError: assignment to
undeclared variable y
}
```

برای آشنایی بیشتر با این ویژگی پست زیر رو ببینید:

بیشتر بدانید

[ویدئو] Strict Mode | آموزش جاوااسکریپت به زبان ساده

Strict Mode حالتی هست که به ما اجازه میده دقیق‌تر و سالم‌تر برنامه‌نویسی کنیم

. . .

Resources:

- https://stackoverflow.com/questions/19317943/why-referencing-non-existent-property-of-an-...
- https://dev.to/macmacky/70-javascript-interview-questions-5gfi
- https://stackoverflow.com/questions/359494/which-equals-operator-vs-should-be-used-in-jav...
- https://medium.com/javascript-in-plain-english/https-medium-com-javascript-in-plain-english-...
- https://www.w3schools.com/js/js_strict.asp

21. Closure چیه؟

Closure یا کلوژر، یک تابعی هست که توی یک تابع دیگه تعریف میشه که می‌تونه علاوه بر متغیرهای حوزه‌ی خودش، به متغیرهای حوزه‌ی تابع بیرونی هم دسترسی داشته باشه.

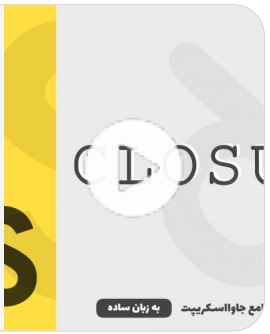
همونطور که می‌دونیم توابع دارای حوزه‌ی دید محلی هستن. متغیرهایی که توی توابع تعریف میشن، فقط مخصوص همون حوزه هستن و بیرون از اونجا قابل دسترسی نیستن. اون متغیرها زمانی ساخته میشن که تابع صدا زده میشه و زمانی که کار تابع تموم شد از بین میرن و دیگه قابل دسترسی نیستن. برعکس متغیرهای سراسری که توی همه جای برنامه و تا زمانی که برنامه فعال هست قابل دسترسی هستن.

خب از مشکلات متغیرهای محلی اینه که عمر کوتاهی دارن. و از مشکلات متغیرهای سراسری اینه که توی کل برنامه قابل دسترسی و ویرایش هستن. کلوژر این مشکل رو حل میکنه! می‌تونیم متغیرهایی با سطح دسترسی محدود، اما با طول عمر به اندازه متغیرهای سراسری داشته باشیم!

نحوه‌ی نوشتن کلوژرها خیلی ساده هست. به سادگی نوشتن یک تابع توی یک تابع دیگه. اما نکته‌ی مهم درک کاربرد اونهاست:

```
function init() {
    var counter = 0;

    function jump_counter() {
        counter++;

        return counter;
    }

    return jump_counter;
}

var jump = init();

jump(); // counter: 1
jump(); // counter: 2
jump(); // counter: 3
```

توضیح کامل کلوژرها و همچنین توضیح این مثال رو می‌تونین توی این مقاله‌ی اختصاصی رو بخونید.



بیشتر بدانید

[ویدئو] کلوژرها Closure | آموزش جاوااسکریپت به زبان ساده

کلوژرها یکی از پرکاربردترین تکنیک‌ها توی جاوااسکریپت هستن و نکته‌های ظریفی دارن که اونها رو توی این قسمت بررسی می‌کنیم

22. مقدارهای falsy چی هستن؟

به مقادیری گفته میشه که وقتی اونها رو به Boolean تبدیل می‌کنیم، به false تبدیل میشن. مقادیر زیر falsy هستن:

```
const falsyValues = ['', 0, null, undefined, NaN, false];
```

بیشتر بدانید

[ویدئو] عبارت‌های Falsy و Truthy | آموزش جاوااسکریپت به زبان ساده

توی این قسمت کوتاه با مقادیر falsy و truthy آشنا می‌شیم

23. چطور بررسی کنیم یک عبارت falsy هست؟

از دو روش می‌تونیم بررسی کنیم که آیا یک عبارت falsy هست یا خیر. یکی استفاده از تابع Boolean و یکی استفاده از عملگر Double Not یا !!

```
var sch = null;

// #1:
console.log(Boolean(sch)); // false

// #2:
console.log(!! sch); // false
```

استفاده از روش دوم مخصوصا توی شرط‌ها راحت‌تر هست.

بیشتر بدانید

[ویدئو] چطور بررسی کنیم یک عبارت falsy هست؟

بررسی falsy بودن یک مقدار توی جاوااسکریپت رو توی این ویدئو یاد می‌گیریم

24. this چیه و چه مقادیری داره؟

> *مقدار کلمه کلیدی this، به قسمتی از کد اشاره می‌کنه که توی همین لحظه داره اجرا میشه. مقدار this یک آبجکت هست.*

مثلا اگه this روی توی یک تابع بنویسیم و این تابع رو توی حوزه‌ی سراسری فراخوانی کنیم، مقدار this آبجکت سراسری هست که توی مرورگر متغیر window هست:

```
var name = "Sarah";

function person() {
    console.log(this.name);
}

person(); // Sarah
```

مهم نیست تابع رو کجا و چه جوری نوشتیم. چیزی که روی مقدار this تاثیرگذار هست مکان و نحوه‌ی فراخونی تابع هست. برای this یک مقاله اختصاصی نوشتم که اون رو می‌تونید از اینجا بخونین.

بیشتر بدانید

[ویدئو] Closure چیه؟

توی این ویدئو با یکی از پرکاربردترین قابلیت‌های جاوااسکریپت آشنا می‌شیم

25. Prototype چیه؟

وقتی آبجکت‌ها توی جاوااسکریپت به وجود میان، یک سری پراپرتی‌ها و متدها رو از یک آبجکت به اسم prototype به ارث می‌برن. آبجکت prototype مثل یک طرح و الگو هست برای بقیه آبجکت‌ها.

برای مثال همه رشته‌ها چند متد پیش‌فرض مثل ()replace، ()split و یک پراپرتی به اسم length دارن. وجود این پراپرتی‌ها و متدها به این دلیل هست که هنگام ساخته شدن یک رشته، متدها و پراپرتی‌ها از آبجکت String.prototype به ارث برده میشن. بنابراین اگه ما آبجکت String.prototype رو ویرایش کنیم و یا به اون متد و پراپرتی اضافه کنیم، این متدها و پراپرتی‌ها توی همه‌ی رشته‌های دیگه وجود خواهد داشت.

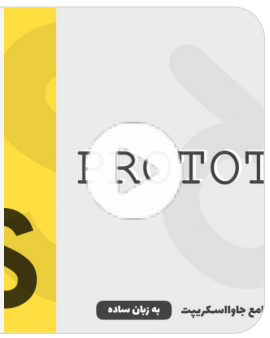
توی کد زیر من یک متد به prototype آبجکت String اضافه کردم به اسم limit. این متد توی همه‌ی رشته‌ها وجود خواهد داشت:

```
String.prototype.limit = function (length) {
  return this.length > length ? this.substring(0, length)
+ "..." : this;
}
```

و به صورت زیر ازش استفاده می‌کنیم:

```
var str = "JavaScript is often described as a prototype-
based language";

console.log(str.limit(20)); // JavaScript is often ...
```

برای آشنایی بیشتر پست زیر رو ببینید:



بیشتر بدانید

[ویدئو] پروتوتایپ | آموزش جاوااسکریپت به زبان ساده

اساس جاوااسکریپت رو آبجکت‌ها و پروتوتایپ‌ها تشکیل میدن و درک پروتوتایپ‌ها کمک بزرگی به درک نحوه کارایی جاوااسکریپت می‌کنه

26. IIFE چیه؟

یک مخفف برای عبارت **Immediately Invoked Function Expression** هست. به تابعی گفته میشه که به محض اینکه تعریف شد، فراخوانی بشه. نوشتن اون خیلی ساده هست. کافیه تعریف تابع رو بین دو پرانتز باز و بسته قرار بدیم.

وقتی تعریف یک تابع رو بین دو پرانتز قرار می‌دیم، چیزی که بین دو پرانتز قرار می‌گیره مثل یک عبارت (Expression) با اون رفتار میشه. عبارت یا Expression یعنی می‌تونیم از اون انتظار یک مقدار داشته باشیم و مثلا اون رو بریزیم توی یک متغیر. به پرانتزهای اطراف تابع زیر دقت کنین:

```
(function () {

})();
```

با اینکار تابعی که بین دو پرانتز هست بلافاصله بعد از تعریف اجرا میشه که یکی از مزیت‌های IIFE هست. اگه نخوایم از این مزیت استفاده کنیم باید اون رو بصورت زیر می‌نوشتیم:

```
function funcName() {

}

funcName();
```

خب از معایب کد بالا اینه که یک کار اضافی داره صورت می‌گیره. همچنین از funcName یک متغیر سراسری درست شد. از مزیت‌های IIFE اینه که مقدارهای سراسری دچار تغییر نمی‌شن:

```
function name() {
    console.log('Bob');
}

(function name() {
    console.log('Lucas');
})(); // Lucas

name(); // Bob
```

برای آشنایی کامل با IIFE ها پست زیر رو ببینین:

بیشتر بدانید

[ویدئو] توابع IIFE | آموزش جاوااسکریپت به زبان ساده

با کمک این توابع می‌تونیم برنامه‌ای با امنیت بیشتر داشته باشیم

27. متد apply چکار می‌کنه؟

با متد apply می‌تونیم یه کاری کنیم که this توی یک تابع به آبجکت دلخواه ما اشاره کنه.

همونطور که می‌دونیم آبجکت‌ها می‌تونن متد داشته باشن. مثل متد `name` توی آبجکت زیر:

```
const person = {
  firstname: 'John',
  lastname: 'Doe',

  name() {
    alert(`${this.firstname} ${this.lastname}`);
  }
}

person.name(); // John Doe
```

خب در حالت عادی متد `name` متعلق به آبجکت `person` هست و `this` به این آبجکت اشاره می‌کنه. با استفاده از متد `apply` می‌تونیم یه کاری کنیم که `this` توی متد `name` به یک آبجکت دلخواه دیگه اشاره کنه. یعنی متد `name` رو بتونیم جاهای دیگه استفاده کنیم:

```
const person = {
  firstname: 'John',
  lastname: 'Doe',

  name() {
    alert(`${this.firstname} ${this.lastname}`);
  }
}

const person1 = {
  firstname: 'Jeff',
  lastname: 'Olson'
}

const person2 = {
  firstname: 'Maria',
  lastname: 'Debug'
}

person.name.apply(person1);
person.name.apply(person2);
```

اگه متد `name` نیاز به آرگومان داشته باشه، می‌تونیم آرگومان‌ها رو **بصورت آرایه** پاس بدیم. ابتدا متد `name` رو بازنویسی کنیم که دو تا آرگومان بگیره:

```
const person = {
  name(age, country) {
    alert(`${this.firstname} ${this.lastname}, ${age}, $
{country}`);
  }
}
```

و حالا بصورت زیر می‌تونیم ازش استفاده کنیم:

```
const person1 = {
  firstname: 'John',
  lastname: 'Doe'
}

person.name.apply(person1, [3, 'Japan']);
```

آرگومان دوم متد `apply` یک آرایه هست. هر کدوم از آیتم‌های این آرایه به عنوان آرگومان‌های متد name در نظر گرفته میشه.

بیشتر بدانید

[ویدئو] متد apply چکار می‌کنه؟

با متد apply می‌تونیم یه کاری کنیم که this توی یک تابع به آبجکت دلخواه ما اشاره کنه

28. متد call چکار می‌کنه؟

با استفاده از متد call می‌تونیم از یک تابع جوری استفاده کنیم که مقدار `this` توی اون، به آبجکت دلخواه ما اشاره کنه:

```
function add(first, second, third) {
  this.result = first + second + third;
}

const item1 = { result: 0 }
const item2 = { result: 0 }
const item3 = { result: 0 }

add.call(item1, 3, 2, 1);
add.call(item2, 9, 3, 2);
add.call(item3, 6, 1, 3);

alert(item1.result); // 6
alert(item2.result); // 14
alert(item3.result); // 10
```

اگه تابع ما نیاز به آرگومان داشته باشه، اون رو **بصورت جدا جدا** پاس می‌دیم.

همونطور که دیدیم `call` و `apply` شبیه به هم هستن. اما یک تفاوت جزئی دارن که توی سوال بعدی با اون آشنا میشیم.

بیشتر بدانید

[ویدئو] متد call چکار می‌کنه؟

با استفاده از متد call می‌تونیم از یک تابع دیگه، جوری استفاده کنیم که مقدار this توی این متد به آبجکت دلخواه ما اشاره کنه

29. تفاوت call و apply چیه؟

خروجی متدهای call و apply یکی هستن و دقیقاً یک کار رو انجام میدن. تفاوت اونها توی نحوه‌ی استفاده از اونهاست. توی متد `apply` ما آرگومان‌ها رو با **یک آرایه** پاس می‌دادیم. اما توی `call` باید **بصورت جدا جدا** پاس بدیم. مثال سوال قبل رو در نظر بگیرید:

```
add.call(item1, 3, 2, 1);
add.apply(item1, [3, 2, 1]);
```

هر دو عبارت خروجی یکسانی دارن.

بیشتر بدانید

[ویدئو] تفاوت متدهای call و apply چیه؟

این دو متد کاربرد مشابهی دارن اما یک تفاوت جزیی دارن که اون رو توی این ویدئو بررسی می‌کنیم

30. متد bind چکار می‌کنه؟

متد `bind` برای ساختن تابعی استفاده میشه که مقدار `this` معین و مشخصی داره. همونطور که می‌دونیم در حالت عادی توابع ما تا زمانی که اجرا نشن نمی‌تونیم مقدار `this` اونها رو تشخیص بدیم. اما شرایطی هست که ما می‌خوایم صراحتاً مشخص کنیم که `this` چه مقداری داشته باشه. اینجا چنین متدی به کار ما میاد.

متد `bind` به ما یک تابع رو برمی‌گردونه که وقتی اون رو صدا می‌زنیم، مقدار `this` توی اون تابع به آبجکت دلخواه ما اشاره می‌کنه. پس به این بستگی نداره که این تابع چطوری و کجا داره صدا زده میشه.

مثال زیر رو درنظر بگیرید:

```
1  function hit() {
2    this.count++;
3  }
```

اینجا تا زمانی که تابع `hit` اجرا نشه، مقدار `this` توی خط ۲ رو نمی‌تونیم تشخیص بدیم. اما از متد `bind` به شکل زیر استفاده می‌کنیم تا this دلخواه خودمون رو مشخص کنیم:

```
1  const jumps = { count: 0 }
2
3  const hitJumps = hit.bind(jumps);
```

ما اینجا گفتیم که آبجکت `jumps` رو در نظر بگیر برای `this` که توی `hit` داره استفاده میشه. خروجی `bind` که الان یک تابع هست، ریخته میشه توی متغیر `hitJumps` و با هر بار فراخونی اون، متد `hit` یک واحد به `count` توی آبجکت `jumps` اضافه می‌کنه:

```js
function hit() {
  this.count++;
}

const jumps = { count: 0 }
const hitJumps = hit.bind(jumps);

hitJumps();
hitJumps();
hitJumps();

alert(jumps.count); // 3
```

برای آشنایی کامل با این متد می‌تونین پست زیر رو ببینین:

بیشتر بدانید

متد Bind در جاوااسکریپت به زبان ساده

با متد Bind می‌تونیم تابعی بسازیم که this توی اون به آبجکت دلخواه ما اشاره کنه

بیشتر بدانید

[ویدئو] متد bind چکار می‌کنه؟

متد bind یکی از مهمترین متدهای جاوااسکریپت هست. اگه توسعه‌دهنده ری‌اکت باشید حتماً با این متد آشنایی دارید

. . .

Resources:

- https://stackoverflow.com/questions/6309947/javascript-closure-advantages
- https://dev.to/macmacky/70-javascript-interview-questions-5gfi
- https://stackoverflow.com/questions/12703214/javascript-difference-between-a-statement-a...
- https://www.w3schools.com/js/js_function_apply.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Functio...
- https://www.w3schools.com/js/js_object_prototypes.asp
- https://www.geeksforgeeks.org/prototype-in-javascript
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes
- https://stackoverflow.com/questions/7463658/how-to-trim-a-string-to-n-chars-in-javascript

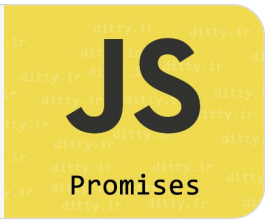
31. پرامیس‌ها چی هستن؟

پرامیس‌ها برای مدیریت کردن عملیات ناهمگام استفاده میشن. قبلا برای هندل کردن عملیات ناهمگام، از کال‌بک فانکشن استفاده می‌کردیم که اینکار معایبی داشت. اما با استفاده از پرامیس‌ها خیلی راحت می‌تونیم تعداد زیادی از عملیات ناهمگام رو هندل کنیم.

مزایای پرامیس‌ها

- خوانایی کد بالاتر میره. چون کدهای ساده‌تری خواهیم داشت.
- مدیریت بهینه‌تر نتیجه موفقیت‌آمیز یک عملیات ناهمگام
- مدیریت بهینه‌تر خطای یک عملیات ناهمگام

پیشنهاد می‌کنم این مقاله اختصاصی درباره پرامیس‌ها در جاوااسکریپت رو بخونید.



بیشتر بدانید

آموزش Promise در جاوا اسکریپت به زبان ساده

Promise ها رو بررسی می‌کنیم که چی هستن و به چه دردی می‌خورن

32. توابع مرتبه‌بالا چی هستن؟

توابع مرتبه بالا یا Higher order functions به توابعی گفته میشه که می‌تونن یک تابع رو به عنوان ورودی بگیرن و یا یک تابع رو به عنوان به خروجی بفرستن.

```
1  function higherOrderFunction(param, callback) {
2    callback(param);
3
4    return function() {
5      // blah
6    }
7  }
```

آرگومان دوم تابع بالا یعنی callback یک تابع هست. همچنین توی خط ۴ تابعی رو دارم به خروجی می‌فرستیم.

برعکس توابع مرتبه بالا، توابع مرتبه اول یا First order functions هستن که نه تابعی رو به عنوان ورودی می‌گیرن و نه تابعی رو به خروجی می‌فرستن.

33. چرا به توابع جاوا اسکریپت می‌گن موجودیت درجه یک؟

اول باید بدونیم معنی موجودیت درجه یک یا First class entity چی هست.

توی یک زبان برنامه‌نویسی، موجودیت‌ها مثل متغیرها و توابع، یک سری عملیات مثل پاس داده‌شدن به‌عنوان یک آرگومان، به خروجی فرستاده شدن توی یک تابع، ویرایش شدن و به یک متغیر نسبت داده شدن رو رو پشتیبانی می‌کنن. یک موجودیت درجه یک به موجودیتی گفته میشه که از همه‌ی عملیات پشتیبانی کنه! به موجودیت درجه یک، آبجکت درجه یک هم گفته میشه.

توابع توی جاوااسکریپت موجودیت‌های درجه یک هستن. یعنی می‌تونن به عنوان یک آرگومان به یک تابع دیگه پاس داده بشن، به عنوان خروجی یک تابع در نظر گرفته بشن و به یک متغیر نسبت داده بشن. توی کد زیر همه‌ی این عملیات رو می‌بینیم:

```
var func = function passAsArg(value, callback) {
    return callback(value);
}

func('value', alert);
```

34. متد map چکار می‌کنه؟

متد map، یک تابع روی همه‌ی اعضای یک آرایه اعمال می‌کنه. خروجی این متد یک آرایه هست که شامل خروجی تابع اعمال شده روی اعضای آرایه مورد نظر ماست:

```
let numbers = [ 1, 2, 3, 4 ];

let output = numbers.map(function(num) {
    return num * 2;
});

console.log(output); // [ 2, 4, 6, 8 ]
```

پارامتر اول متد map اون تابعی هست که می‌خوایم روی تک‌تک اعضای آرایه مورد نظر ما اعمال بشه.

شاید توی مصاحبه‌ها از شما پرسیده بشه که این متد رو بصورت دستی پیاده‌سازی کنین. خب اولین چیز اینه که مسئله رو درک کنیم، یعنی این متد داره چکار می‌کنه. الان که با نحوه‌ی کارکرد این متد آشنا شدیم می‌تونیم اون رو پیاده‌سازی کنیم:

```
function map(arr, mapCallback) {
    if (! Array.isArray(arr) || ! arr.length || typeof
mapCallback !== 'function') {
        return [];
    } else {
        let result = [];

        for (let i = 0, len = arr.length; i < len; i++) {
            result.push(mapCallback(arr[i], i, arr));
        }

        return result;
    }
}
```

توی این تابع اول بررسی کردیم که آیا ورودی‌های ما معتبر هستن یا نه، و بعد تابع مورد نظرمون رو روی تک تک اعضا اعمال کردیم و خروجی رو ریختیم توی یک آرایه جدید و نهایتا اون رو return کردیم.

برای آشنایی کامل‌تر با این متد این پست رو ببینید:

بیشتر بدانید

متدهای کاربردی آرایه‌های جاوااسکریپت - قسمت اول

توی این قسمت متدهای find، map و filter رو بررسی می‌کنیم

35. متد filter چکار می‌کنه؟

این متد خب اسمش روش هست. آرایه مورد نظر ما رو فیلتر میکنه. جزئی‌تر بگیم، متد filter تابع دلخواه x که خروجی اون Boolean هست رو روی تک تک اعضای یک آرایه اعمال می‌کنه. خروجی این متد یک آرایه شامل اعضایی هست که خروجی تابع x برای اونها true هست:

```
let numbers = [ 111, 99, 51, 101, 102 ];

let output = numbers.filter(function(item) {
    return item > 100;
});

console.log(output); // [ 111, 101, 102 ]
```

بعد از درک مسئله، می‌تونین مسئله رو حل کنین. اگه از شما خواسته شد متد filter رو بصورت دستی بنویسید:

```
function filter(arr, filterCallback) {
    if (! Array.isArray(arr) || ! arr.length || typeof
filterCallback !== 'function')  {
        return [];
    } else {
        let result = [];

        for (let i = 0, len = arr.length; i < len; i++) {
            if (filterCallback(arr[i], i, arr)) {
                result.push(arr[i]);
            }
        }

        return result;
    }
}
```

برای آشنایی کامل‌تر با این متد این پست رو ببینید:

بیشتر بدانید

متدهای کاربردی آرایه‌های جاوااسکریپت - قسمت اول

توی این قسمت متدهای find، map و filter رو بررسی می‌کنیم

36. متد reduce چکار می‌کنه؟

بطور کلی متد reduce برای تبدیل آرایه به یک مقدار ساده‌تر استفاده میشه. در واقع این متد برای ساده کردن آرایه بکار میره. متد reduce روی اعضای یک آرایه (به ترتیب از چپ به راست) یک تابعی رو اجرا می‌کنه. هر باری که این تابع روی یک عضو اجرا میشه، خروجی اون، با مقدار خروجی تابع روی عضو قبلی، جمع میشه. منظور از جمع، انباشت اطلاعات قبلی هست. جاهایی از متد ریدوس استفاده میشه که اعضای قبل یا بعد یک عضو، وابسته به هم هستن، مثلاً جمع، میانگین اعضای یک آرایه عددی.

شاید توی مصاحبه‌ها از شما بخوان که یک راه حل بدین برای جمع کردن اعضای یک آرایه. قطعا از شما **نمی‌خوان** که با حلقه for به این صورت بنویسین:

```
let numbers = [1, 2, 3, 4];
let output = 0;

for (var i = 0; i < numbers.length; i++) {
    output += numbers[i];
}

console.log(output); // 10
```

مصاحبه‌کننده جاوااسکریپت قطعا از شما یک راه حل باحال‌تر و خلاقانه‌تر می‌خواد 😉

مسئله‌ی بالا رو میشه با متد `reduce` نوشت:

```
let numbers = [1, 2, 3, 4];

let output = numbers.reduce(function(total, currect) {
    return total + currect;
});

console.log(output); // 10
```

از پست زیر می‌تونید به طور مفصل با متد Reduce آشنا بشید:

بیشتر بدانید

متد Reduce در جاوااسکریپت به زبان ساده

یکی از متدهایی رو بررسی می‌کنیم که به ظاهر پیچیده هست ولی ...

37. arguments توی توابع چیه؟

arguments یک *آبجکت شبیه به آرایه* هست که توی همه توابع وجود داره و شامل آرگومان‌هایی هست که به تابع پاس داده شده:

```
function commit() {
    console.log(arguments); // [Arguments] { '0': 1, '1': 2, '2': 4 }
    console.log(arguments[0]); // 1
}

commit(1, 2, 4)
```

به arguments گفته میشه آبجکت شبیه آرایه. این دلیل که شبیه به یک آرایه هست که index اون از صفر شروع میشه و همچنین شامل پراپرتی length هست. اما متدهایی که یک آرایه داره مثل foreEach و map رو نداره.

Arrow Function ها چنین متغیری ندارن:

```
commit = () => {
    console.log(arugments);
}

commit(1, 2, 4); // ReferenceError: arugments is not defined
```

پس بجای اون می‌تونیم از *پارامتر rest* یا *سه نقطه* استفاده کنیم:

```
commit = (...args) => {
    console.log(args);
}

commit(1, 2, 4); // [ 1, 2, 4 ]
```

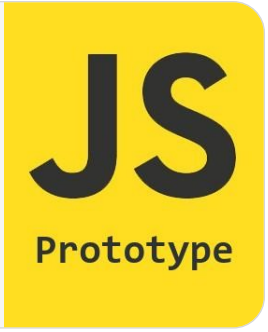
پارامتر rest یک آرایه واقعی هست و همه‌ی متدها و پراپرتی‌هایی که یک آرایه معمولی داره رو شامل میشه.

38. چه جوری یک آبجکت بدون prototype بسازیم؟

اگه از متد `Object.create` با ورودی `null` استفاده کنیم، آبجکت‌هایی که ساخته میشن پروتوتایپ ندارن:

```
const obj = Object.create(null);


alert(obj.toString()); // obj.toString is not a function
```

ورودی‌ای که متد `create` می‌گیره یک آبجکت هست و به عنوان پروتوتایپ آبجکت جدید درنظر گرفته میشه. اگه این ورودی `null` باشه، خروجی یک آبجکت بدون Prototype هست.



بیشتر بدانید

Prototype در جاوااسکریپت به زبان ساده - قسمت اول

پروتوتایپ یکی از مفاهیم پایه و مهم زبان جاوااسکریپت هست که با درک کردن اون می‌تونیم جاوااسکریپت رو بهتر بشناسیم

39. چرا متغیر b توی این کد یک متغیر سراسری شده؟

```
function myFunc() {
  let a = b = 5;
}

myFunc();
console.log(b) // 5
```

باید بدونیم که عملگر مساوی (=) از راست به چپ شروع به ارزیابی می‌کنه. پس خط دوم کد بالا در واقع یه چیزی مثل این میشه:

```
let a = (b = 5);
```

همونطور که می‌دونیم متغیرهایی که بدون `var` تعریف میشن، متغیر سراسری به حساب میان. پیشنهاد می‌کنم این مقاله درباره ساخت متغیرها توی جاواسکریپت رو بخونین.

برای حل این مسئله می‌تونیم از راهکار زیر استفاده کنیم:

```
function myFunc() {
  let a, b;
  a = b = 0;
}

myFunc();
```

ابتدا متغیرها رو تعریف کردیم و بعد به اونها مقدار دادیم.

40. ECMAScript چیه؟

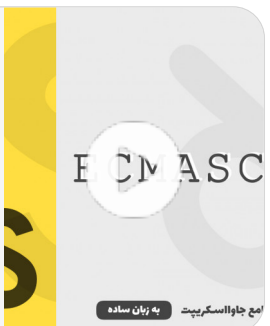
ECMAScript یا اکما اسکریپت یک استاندارد هست. و زبان‌هایی مثل جاواسکریپت و اکشن‌اسکریپت طبق این استاندارد طراحی شدن. پس جاواسکریپت و اکشن‌اسکریپت از اکما اسکریپت به عنوان هسته خودشون استفاده می‌کنن و زیر مجموعه‌ای از اکما اسکریپت هستن. مثل سه اتومبیل با ظاهر و مشخصات متفاوت اما موتور یکسان!

برای آشنایی کامل با اکماسکریپت می‌تونید پست زیر ببینید:

بیشتر بدانید

[ویدئو] اکماسکریپت | آموزش جاواسکریپت به زبان ساده

اکماسکریپت یک استاندارد هست که براساس اون زبان‌هایی مثل جاواسکریپت طراحی شدن



. . .

https://medium.com/java...

https://en.wikipedia.org/wiki...

https://dev.to/mac...

https://en.wikipedia.org/wik...

https://stackoverflow.com/qu...

https://www.w3schools.com/jsre...

https://developer.mozilla.org/en-...

https://codeburst.io/learn-u...

https://stackoverflow.com/que...

41. Local Storage چیه؟

Local Storage یک مکان ذخیره‌سازی هست که توی **مرورگر** کاربر وجود داره و این امکان رو به ما میده تا با جاوا اسکریپت یه سری اطلاعات رو ذخیره و استفاده کنیم. در واقع عملیات ثبت، خواندن، بروزرسانی و حذف (CRUD) رو می‌تونیم با فضایی که در اختیار داریم انجام بدیم. اطلاعاتی که در Local Storage ذخیره میشه دائمی هست. یعنی اگه کاربر تب مرورگر و یا حتی خود مرورگر رو ببنده این اطلاعات حذف **نمیشن**. برای آشنایی بیشتر پیشنهاد می‌کنم این مقاله رو بخونید:

آموزش استفاده از Local Storage در جاوا اسکریپت

42. چه چیزایی به جاوا اسکریپت توسط ES6 اضافه شده؟

۱۴ مورد زیر توسط ES6 به جاوا اسکریپت اضافه شده که ۷ آیتم رو توی این مقاله بررسی می‌کنیم:

- Arrow Functions
- Classes
- Template Strings یا Template Literals
- Enhanced Object literals
- Object Destructuring
- Promises
- Generators
- Modules
- Symbol
- Proxies
- Sets
- پارامترهای پیشفرض توابع
- عملگرهای Rest and Spread
- Let و Const

43. var و let و const چه تفاوت‌هایی با هم دارن؟

سه راه برای تعریف کردن متغیر توی جاوااسکریپت هستن که let و const توسط ES6 معرفی شدن. متغیرهایی که با var تعریف میشن به اصطلاح Function Scope هستن. یعنی سرتاسر یک تابع قابل دسترسی هستن:

```
function avoir() {
    if (true) {
        var x = 5;
    }

    console.log(x);
}

avoir() // 5
```

همونطور که می‌بینید متغیر x داخل if تعریف شد و بیرون از if هم قابل دسترسی هست. اما اگه متغیر x رو با let یا const تعریف می‌کردیم، بیرون از if یا هر بلاک دیگه‌ای قابل استفاده نبود. به فضای بین دو براکت میگن بلاک:

```
function avoir() {
    if (true) {
        let x = 5;
    }

    console.log(x);
}

avoir(); // Error: x is not defined
```

چرا x قابل دسترسی نیست؟ چون متغیرهایی که با let و const تعریف میشن فقط توی محدوده‌ی همون بلاک قابل دسترسی هستن.

تفاوت let و const

وقتی متغیری با let تعریف میشه بعدا میشه مقدار اون رو تغییر داد. اما متغیرهایی که با const تعریف میشن همون ابتدای تعریف باید مقداردهی بشن و مقدارش هم دیگه قابل تغییر نیست.

پیشنهاد می‌کنم این مقاله اختصاصی درباره تفاوت‌های var و let و const رو بخونین.

44. Arrow Function چیه؟

یک راه جدید برای تعریف توابع توی جاوااسکریپت استفاده از Arrow Function ها هست که توی ES6 معرفی شده. این نوع توابع تعریف ساده‌تری دارن. توی مثال زیر آیتم اولی یک تابع معمولی هست و آیتم بعدی دقیقا مشابه همون هست که با Arrow Function نوشته شده:

```
// Old functions
function alphaville() {
    return 'Forever Young';
}

// Arrow Function
alphaville = () => 'Forever Young';
```

ما توی آیتم اولی برای فرستادن اون متن به خروجی از کلمه کلیدی return استفاده کردیم. اما توی آیتم دومی متن مورد نظرمون رو دقیقا بعد Arrow یا `=>` نوشتیم که دقیقا همون کار return رو می‌کنه. اگه لازم باشه کارهای بیشتری توی تابع انجام بدیم چطور؟🤔

از براکت جلوی Arrow استفاده می‌کنیم:

```
alphaville = () => {
    let song = 'Forever Young';
    // ...
    // ...
    // ...

    return song;
}
```

اگه تابع ما پارامتر داشته باشه اون رو داخل پرانتزی که بعد از مساوی و `=>` هست می‌ذاریم:

```
play = (song) => {
    return song;
}

play(your_favorite_song);
```

دو نکته که باید درنظر داشته باشیم اینه که Arrow Function ها برخلاف توابع معمولی متغیری به اسم arguments ندارن. بجای اون می‌تونیم از پارامتر rest (سه نقطه) استفاده کنیم. همچنین این توابع دارای مقدار مستقل this نیستن و این مقدارش رو از حوزه‌های بالاتر می‌گیره. برای درک بهتر مورد پیشنهاد می‌کنم پست‌های زیر رو بخونین:

بیشتر بدانید

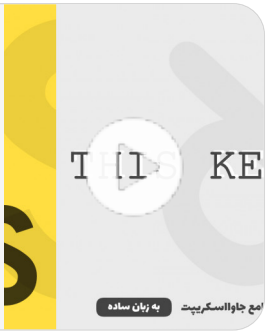
[ویدئو] توابع - قسمت 5 | Arrow Functions

Arrow Function ها یک راه جدید برای داشتن Function Expression توی جاوااسکریپت هستن که توی این قسمت با اونها آشنا می‌شیم

بیشتر بدانید

[ویدئو] کلمه‌کلیدی this | آموزش جاوااسکریپت به زبان ساده

یکی از مهمترین موضوع‌ها توی جاوااسکریپت کلمه‌کلیدی this هست که امروز با اون آشنا می‌شیم



45. تفاوت نوشتن کلاس‌ها توی ES5 و ES6؟

قبل از ES6 برای داشتن یک چیزی مثل کلاس که بشه شامل متد و پراپرتی باشه و بشه ازش نمونه ساخت از Constructor Functions مثل زیر استفاده می‌کردیم:

```
function Person(firstname, age) {
  this.firstName = firstname;
  this.age = age;
}

var jack = new Person("Jack", 35);
var mark = new Person("Mark", 45);

console.log(jack.age); // 35
console.log(mark.age); // 45
```

همونطور که می‌بینیم اینجا چیزی به اسم کلاس وجود نداره و یک تابع هست که توسط کلمه کلیدی new داره پیاده‌سازی میشه.

با معرفی شدن ES6 ساخت کلاس توی جاوا اسکریپت معنادار تر شد:

```
class Person {
    constructor(firstname, age){
        this.firstName = firstname;
        this.age = age;
    }

    static self() {
        return this;
    }

    toString(){
        return "[object Person]";
    }

    getInfo(){
        return `${this.firstname} is ${this.age} years
old`;
    }
}
```

46. Template Literals چیه؟

قبلا برای نوشتن یک تک‌کوتیشن توی یک رشته‌ای که با تک‌کوتیشن نوشته شده از یک بک‌اسلش مثل زیر استفاده می‌کردیم:

```
var str = 'I\'m Bond, James Bond';
```

همین قضیه برای دابل‌کوتیشن‌ها صدق می‌کنه.

Template Literals یک راه جدید برای ساختن رشته‌ها توی جاوااسکریپت هستن که علاوه بر اینکه مشکل بالا رو حل می‌کنه یک مزیت دیگه هم داره که توی ادامه بهش اشاره می‌کنم. توی رشته‌های Template Literals بجای کوتیشن، از Backtick یا ` استفاده می‌کنیم. (این کارکتر روی کیبرد من بالای Tab هست).

```
var str = `I'm Bond, James Bond "'"'"'"'"'"`;
```

توی روش‌های قدیمی ساخت رشته توی جاوااسکریپت وقتی می‌خواستیم یک متغیر رو وسط یک رشته بچسبونیم، باید رشته رو قطع می‌کردیم، متغیر رو اضافه می‌کردیم و ادامه رشته رو بهش می‌چسبوندیم:

```
var age = 105;
var str = "He is " + age + " years old";
```

اما توی روش جدید و با Template Literals کافیه متغیر رو بصورت زیر اضافه کنیم:

```
var age = 105;
var str = `He is ${age} years old`;
```

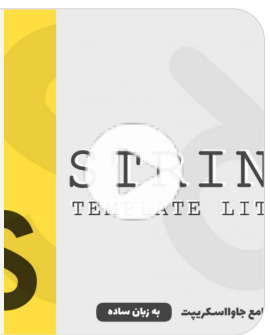
داخل `${...}` هر عبارتی می‌تونه قرار بگیره. مثلا:

```
var age = 105;

console.log(`He is ${(age * 2) + 2} years old`);

console.log(`He is ${person.getAge()} years old`);

```

بیشتر بدانید

[ویدئو] رشته‌ها - قسمت 2 | رشته‌های مدرن Template Literals

توی این قسمت با یک روش جدید و کاربردی برای ساختن رشته‌ها توی جاوااسکریپت آشنا می‌شیم



47. Object Destructuring چیه؟

Object Destructuring یک راه جدید برای گرفتن یا استخراج کردن مقدارهای یک آرایه یا آبجکت و داشتن اونها توی متغیرها هست.

قبلا برای اینکه مقدارهای یک آبجکت یا آرایه رو توی متغیرها داشته باشیم به شکل زیر عمل می‌کردیم:

```
const employee = {
    firstName: "Job",
    lastName: "Offer",
    position: "Software Developer",
    yearHired: 1954
};

var firstName = employee.firstName;
var lastName = employee.lastName;
var position = employee.position;
var yearHired = employee.yearHired;
```

به چهار خط آخر کد بالا دقت کنین؛ برای هر آیتم یک متغیر ساختیم. برای آبجکت‌های بزرگ‌تر، تعداد خط‌ها و متغیرها همینطوری بیشتر میشن. یک راه بهتر استفاده از Object Destructuring هست که به صورت زیر اون رو می‌نویسیم:

```
let { firstName, lastName, position, yearHired } =
employee;


console.log(firstName); // Job
console.log(lastName); // Offer
console.log(position); // Software Developer
console.log(yearHired); // 1954
```

خب همین یک خط، کار چهار خط بالا رو انجام میده. توی این روش، مقدار key های آبجکت، توی متغیرهایی که همنام با key های آبجکت هستن ریخته میشه. دقت کنین ترتیب مهم نیست. فقط باید اسم متغیرهایی که توی براکت استفاده می‌کنیم، توی آبجکت وجود داشته باشن. اگه بخوایم از نام دلخواه خودمون استفاده کنیم به روش زیر عمل می‌کنیم:

```
let { firstName: fName, lastName: lName, position,
yearHired } = employee;

console.log(fName); // Job
console.log(lName); // Offer
```

و اگه آبجکت ما یک پراپرتی خاص رو نداشت، می‌تونیم مثل زیر یک مقدار پیشفرض بدیم:

```
let { age = 3, lastName: lName, position, yearHired } =
employee;
```

48. ماژول‌ها توی ES6 چی هستن؟

> *نویسندگان خوب، کتاب‌ها رو به بخش‌ها و فصل‌های مجزا تقسیم‌بندی میکنن. برنامه‌نویس های خوب، برنامه رو به بخش‌های مختلف تقسیم‌بندی میکنن.*

درست مثل یک کتابی که به فصل‌های مجزا تقسیم‌بندی میشه، یک برنامه میتونه به قسمت‌های مجزا تقسیم بندی بشه که به هر یک از این قسمت‌ها میگن ماژول. یک ماژول معمولا مستقل از بخش‌های برنامه عمل میکنه. میتونه بدون مشکل به یک ماژول دیگه و یا به هسته برنامه اضافه یا برداشته بشه. ماژول‌های توی حوزه خودشون اجرا میشن. یعنی اجزای ماژول (متغیرها، توابع و کلاس‌ها) فقط توی خود ماژول قابل دسترسی هستن و Global نیستن. این اجزا تنها زمانی قابل دسترسی بصورت Global هستن که ما اونها رو export کنیم. پس به همین شکل، اگه بخوایم از یک ماژول و اجزای اون استفاده کنیم، باید اونها رو import (وارد) کنیم

برای آشنایی بیشتر با ماژول‌های جاوااسکریپت این مقاله رو بخونید.

49. آبجکت Set چیه؟

آبجکت Set توی جاوااسکریپت یه راه جدید برای نگهداری مقدارهای یکتا هست. یعنی توی ست، هیچ آیتم تکراری وجود نداره. نحوه‌ی نوشتن ست بصورت زیر هست:

```
const alpha = new Set(['a', 'b', 'b', 'b', 'c']);

console.log(alpha); // Set { 'a', 'b', 'c' }
```

چیزی که ست به عنوان ورودی قبول می‌کنه باید از نوع Iterable باشه. یعنی مثل آرایه یا آبجکت که بشه اونها رو پیمایش کرد. همونطور که توی خط ۳ می‌بینید مقدارهای تکراری توی ست حذف میشن.

اضافه کردن به ست

برای اضافه کردن مقدار به ست، از متد add استفاده می‌کنیم:

```
const alpha = new Set(['a', 'b', 'b', 'b', 'c']);

alpha.add('d');
alpha.add('e').add('e').add('e').add('f');

console.log(alpha); // Set { 'a', 'b', 'c', 'd', 'e', 'f'
}
```

حذف از ست

برای حذف از ست از متد delete استفاده می‌کنیم. اگه حذف با موفقیت انجام بشه خروجی true هست. در غیر این صورت false:

```
const alpha = new Set(['a', 'b', 'b', 'b', 'c']);

alpha.delete('b'); // true
alpha.delete('z'); // false

console.log(alpha); Set { 'a', 'c' }
```

برای بررسی اینکه یک آیتم توی ست وجود داره از has استفاده می‌کنیم:

```
const alpha = new Set(['a', 'b', 'b', 'b', 'c']);

console.log(alpha.has('h')); // false
console.log(alpha.has('b')); // true
```

خالی کردن ست

برای خالی کردن ست از متد clear استفاده می‌کنیم:

```
const alpha = new Set(['a', 'b', 'b', 'b', 'c']);

alpha.clear();
console.log(alpha); // Set {}
```

اندازه‌ی ست

برای اینکه تعداد آیتم‌های ست رو بدست بیاریم از پراپرتی size استفاده می‌کنیم:

```
const alpha = new Set(['a', 'b', 'b', 'b', 'c']);

console.log(alpha.size); // 3
```

خب الان که با ست آشنا شدیم، می‌تونیم از اون استفاده کنیم برای اینکه مقدارهای تکراری رو از توی یک آرایه حذف کنیم:

```
const numbers = [ 1, 2, 3, 4, 5, 6, 6, 7, 8, 8, 5 ];
const uniqueNums = [...new Set(numbers)];

console.log(uniqueNums); // [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

برای آشنایی کامل با آبجکت Set جاوااسکریپت، می‌تونید پست زیر رو بخونید:

بیشتر بدانید

همه چیز از آبجکت Set جاوااسکریپت

با یکی دیگه از ویژگی‌های جالب جاوااسکریپت که از ES6 اضافه شده آشنا می‌شیم

50. Callback Function چیه؟

به تابعی میگن که بعد از اینکه یک عملیات خاص اتفاق افتاد، اجرا میشه. مثال زیر رو درنظر بگیرید:

```
$('div').click(function() {
    alert('Hi');
});
```

توی این مثال که با جی‌کوئری نوشته شده، وقتی روی div های توی صفحه کلیک کنیم، تابعی که توی متد click قرار داره اجرا میشه. به اون تابع می‌گن Callback Function. به طور کلی این توابع بعد از اینکه کار یک عملیات به پایان رسید اجرا میشن. یعنی برای مثال اگه بخوایم یک تکه کد بعد از یک عملیات خاص اجرا بشه از Callback Function استفاده می‌کنیم.

بیشتر بدانید

[ویدئو] Callback Function چیه؟

این توابع با رخ‌دادن یک رویداد خاص توی برنامه اجرا میشن

. . .

https://dev.to/...

https://www.w3schools.com/js/j...

https://codeburst.io/java...

51. await/async چی هستن؟

اول باید بدونیم عملیات ناهمگام چی هستن. عملیات ناهمگام به عملیاتی گفته میشه که جدا از روند اصلی برنامه اجرا میشن. یعنی برنامه منتظر تموم شدن اونها **نمی‌مونه** و به کار خودش ادامه میده. مثلا اتصال به یک منبع خارجی با Ajax. توی جاوا اسکریپت برای مدیریت کردن عملیات ناهمگام راه‌های مختلفی وجود داره. مثل کال‌بک فانکشن‌ها، پرامیس‌ها و Async/await.

Async/await یک راه حل برای مدیریت کردن عملیات ناهمگام هست که خیلی راحت‌تر، خواناتر و بهتر از روش‌های دیگه هست. بهتره که بدونیم Async/await توی پشت پرده خودش از پرامیس استفاده می‌کنه. با پرامیس‌ها توی این مقاله آشنا بشید.

فرض کنیم با تابع fetch می‌خوایم یه سری اطلاعات رو از یک منبع خارجی دریافت و پردازش کنیم. اگه با پرامیس‌ها بنویسیم کد ما به صورت زیر خواهد بود:

```
function callApi() {
    return fetch("url/to/api/endpoint")
        .then(resp => resp.json())
        .then(data => {
            // do something
        }).catch(err => {
            // error
        });
}

```

و اگه با Async/await بنویسیم بصورت زیر:

```
async function callApi() {
  try {
    const resp = await fetch("url/to/api/endpoint");
    const data = await resp.json();
    //do something
  } catch (e) {
    //
  }
}
```

به کلمه‌ی کلیدی async به اول تابع دقت کنین. وقتی این کلمه کلیدی اول یک تابع قرار بگیره، خروجی این تابع یک پرامیس خواهد بود. یعنی با چیزی توی این تابع return میشه، میشه مثل پرامیس‌ها رفتار کرد:

```
async function api() {
    return "I'm Api";
}

api().then(alert);
```

و یک کلمه کلیدی دیگه داریم به اسم await. این کلمه کلیدی فقط توی توابعی استفاده میشه که کلمه async اول اونها قرار داشته باشه. از await پشت یک عملیات ناهمگام که یک پرامیس هست استفاده میشه و باعث میشه تا ادامه‌ی برنامه صبر کنه تا این پرامیس resolve بشه. مثال زیر رو ببینید:

```
function externalCall() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('done')
        }, 2000);
    });
}


async function getApi() {
    let result = await externalCall();
    console.log(result);
}

getApi();
```

تابع externalCall مثلا قراره یک کاری انجام بده که ۲ ثانیه زمان میبره. این تابع رو توی تابع getApi استفاده کردیم و قبل از اون از کلمه کلیدی await هم استفاده کردیم. با این کار، خط ۱۲ تا زمانی که کار خط ۱۱ تموم نشده، اجرا نخواهد شد.

برخلاف پرامیس‌ها که باعث به‌وجود اومدن متدهای زنجیره‌ای و کدهای تو در تو میشن، استفاده از Async/await باعث میشه کدهای ساده‌تر، خواناتر و مسطح‌تری داشته باشیم و همچنین خطایابی توی کدها ما راحت‌تر میشه. توضیح کامل و اختصاصی async/await در جاوااسکریپت

52. عملگرهای Spread و Rest چه تفاوت‌هایی دارن؟

این دو عملگر ظاهر کاملا یکسانی دارن و هر دو بصورت سه‌نقطه (...) هستن.

عملگر Spread

کلمه Spread یعنی گسترش دادن و پخش کردن. عملگر Spread برای زمانی هست که می‌خوایم مقدارهای یک آرایه یا هر چیز Iterable رو گسترش بدیم و

پخش کنیم توی یک چیز دیگه. کد زیر رو درنظر بگیرید:

```
function sum(x, y, z) {
  return x + y + z;
}

const numbers = [1, 2, 3];
```

تابع sum سه تا آرگومان نیاز داره. متغیر numbers رو در نظر بگیرید. اگه بخوایم از روش قبلی استفاده کنیم، اعداد رو باید به شکل زیر پاس بدیم:

```
sum(numbers[0], numbers[1], numbers[2]);
```

اما روش باحال‌تر استفاده از عملگر Spread هست:

```
sum(...numbers);
```

این عملگر دقیقا کار مثال قبلی رو انجام داد. مقدارهای آرایه numbers رو پخش کرد توی تابع sum. همونطور که می‌بینید کد ما چقدر کوتاه‌تر و خواناتر شد.

عملگر Rest

این عملگر که توی توابع استفاده میشه، به تابع کمک می‌کنه تا بی‌نهایت آرگومان داشته باشه. در واقع یک روش جدیدتر بجای کلمه کلیدی arguments توی توابع هست. تابع sum مثال بالا رو درنظر بگیرید که فقط سه تا آرگومان قبول می‌کرد. با استفاده از پارامتر rest می‌تونیم یک کاری کنیم که بی‌نهایت آرگومان قبول کنه:

```
function sum(...numbers) {
    return numbers.reduce((total, current) => total +
current);
}

sum(1, 2, 3); // 6
sum(1, 2, 3, 11, 12, 13); // 42
```

53. پارامتر پیشفرض چیه؟

پارامتر پیشفرض که توی اکثر زبان‌های برنامه‌نویسی وجود داره، تا قبل از معرفی ES6 توی جاوا اسکریپت وجود نداشت. یعنی توابع تو جاوا اسکریپت نمی‌تونستن پارامترهای پیشفرض داشته باشن. روش قدیمی برای کنار اومدن با این مشکل استفاده از روش زیر بود:

```
function add(a, b) {
    var a = a || 0;
    var b = b || 0;

    return a + b;
}
```

باید مقدار پیشفرض رو داخل خود تابع می‌نوشتیم. اما با استفاده از امکانی که ES6 به ما میده می‌تونیم کد بالا رو بصورت زیر بنویسیم:

```
function add(a = 0, b = 0){
  return a + b;
}

add(1); // 1
```

می‌تونیم از Object Destructuring هم استفاده کنیم:

```
function getFirst([first, ...rest] = [0, 1]) {
  return first;
}
```

54. آبجکت Wrapper چیه؟

یک سری نوع‌های داده‌ای توی جاوااسکریپت مثل رشته، عدد و بولین دارای متد و پراپرتی هستن درحالیکه آبجکت نیستن. مثلا پراپرتی length توی همه‌ی رشته‌ها هست یا متد toUpperCase:

```
var mein = "dein";

console.log(typeof mein); // string
console.log(mein.length); // 4
console.log(mein.toUpperCase()); // DEIN
```

چرا اینطوری هستن؟🤔

دلیلش وجود آبجکت‌های Wrapper هست. وقتی از نوع‌های داده‌ای Primitive مثل رشته، عدد و بولین (بجز null و undefiend) یک پراپرتی یا متد رو فراخونی می‌کنیم، اون رشته یا عدد بطور موقت داخل یک آبجکت قرار می‌گیره. به این آبجکت موقت میگن Wrapper object. یعنی یک چیزی شبیه به این اتفاق میوفته:

```
var mein = "dein";

alert(new String(mein).toUpperCase()); // DEIN
```

به این ترتیب رشته‌ی ما شبیه به یک آبجکت میشه که شامل یک سری متدها و پراپرتی‌های پرکاربرد هست. آبجکت Wrapper بعد از فراخونی پراپرتی یا متد از بین میرن.

55. Implicit Coercion و Explicit Coercion چی هستن؟

این‌ها دو روش تبدیل نوع داده توی جاوااسکریپت هستن.

Implicit Coercion یا ضمنی

توی این روش برای تبدیل نوع داده لازم نیست کار خاصی انجام بدیم:

```
1  console.log(1 + '6'); // 16 (type of string)
2  console.log(false + true); // 1 (type of number)
3  console.log(6 * '2'); // 12 (type of number)
```

توی مثال اول، عدد 1 ابتدا تبدیل به رشته میشه و بعد به یک **رشته** با مقدار 6 می‌چسبه. دقت کنین اینجا محاسبه ریاضی انجام نشد. بلکه با 1 مثل رشته رفتار شد. ما اینجا برای تبدیل نوع هیچ کار خاصی انجام ندادیم و جاوااسکریپت بصورت خودکار اینکار رو برای ما انجام داد. هرچند توی بعضی از زبان‌ها ممکنه خطا بگیریم.
توی مثال دوم هم خروجی از نوع number خواهد بود با اینکه داریم دو بولین رو جمع می‌کنیم.
توی مثال سوم قبل از اینکه عمل ضرب انجام بشه، رشته‌ی با مقدار 2 تبدیل میشه به عدد. که باعث میشه خروجی ۱۲ و از نوع number بشه.

Explicit Coercion یا صریح

توی این روش باید بطور صریح خودمون تبدیل نوع رو انجام بدیم:

```
1  console.log(1 + parseInt('6'));
```

ما خودمون بطور صریح رشته با مقدار 6 رو به عدد تبدیل کردیم که باعث شد خروجی 7 بشه.

این لینک رو ببینید خیلی بدردتون می‌خوره :)

56. NaN چیه؟

مخفف عبارت Not a Number هست. وقتی یک عملیات ریاضی مثل ضرب روی مقادیر غیر عددی اعمال بشه خروجی NaN خواهد بود. خروجی عبارت‌های زیر

NaN هستن:

```
{} * undefined;
{} * null;
'' * 'a';
1 * {};

undefined + null;
1 + NaN;
1 + undefined;
```

برای اینکه ببینیم یک مقدار NaN هست از تابع isNaN استفاده می‌کنیم:

```
isNaN({} * undefined); // true
isNaN({} * null); // true
isNaN('' * 'a'); // true
isNaN(1 * {}); // true

isNaN(undefined + null); // true
isNaN(1 + NaN); // true
isNaN(1 + undefined); // true
```


57. چطوری ببینیم یک مقدار آرایه هست؟

آبجکت سراسری Array یک متد داره به اسم isArray که بررسی می‌کنه که آیا یک مقدار آرایه هست یا نه:

```
console.log(Array.isArray(5));  // false
console.log(Array.isArray("")); // false
console.log(Array.isArray()); // false
console.log(Array.isArray(null)); // false
console.log(Array.isArray({ length: 5 })); // false

console.log(Array.isArray([])); // true
```

58. چند روش برای پیمایش روی اعضای آرایه با استفاده از حلقه‌ی for توی جاوااسکریپت وجود داره؟

۳ روش وجود داره. یکی حلقه‌ی for معمولی هست که بصورت زیر نوشته میشه:

```
var brands = ['Samsung', 'Apple', 'Huawei',
'Blackberry'];

for (var i = 0; i < brands.length; i++) {
   console.log(brands[i]);
}
```

اگه ما با یک آبجکت که key/value هست سر و کار داشته باشیم، روش‌های بهتر استفاده از حلقه for وجود داره.

for ... in

اگه یک آبجکت داشته باشیم، با این نوع for می‌تونیم روی key (پراپرتی) های اون پیمایش کنیم. نحوه نوشتن اون به صورت زیر هست:

```
let person = {
    name: 'John',
    lastname: 'Doe',
    age: 6
}

for (var key in person) {
    console.log(key); // name, lastname, age
}
```

for ... of

راحت‌ترین راه برای پیمایش روی یک آرایه معمولی استفاده از این نوع حلقه هست:

```
let cars = ['Toyota', 'Suzuki', 'Honda', 'Mazda'];

for (var car of cars) {
    console.log(car);
}

// Toyota
// Suzuki
// Honda
// Mazda
```

پیشنهاد می‌کنم این مقاله اختصاصی درباره روش‌های نوشتن حلقه for رو بخونید.

59. چطور ببینیم یک پراپرتی توی یک آبجکت وجود داره؟

برای این کار دو راه وجود داره.

راه اول استفاده از عملگر in هست:

```
const person = {
    age: 89,
    height: 2.30
};

console.log("age" in person); // true
console.log("weight" in person); // false
```

راه دوم استفاده از متد hasOwnProperty هست که توی همه‌ی آبجکت‌ها وجود داره:

```
console.log(person.hasOwnProperty("age")); // true
console.log(person.hasOwnProperty("weight")); // false
```

60. AJAX چیه؟

یک ابزار برای نمایش اطلاعات بصورت غیرهمگام هست. یعنی با Ajax می‌تونیم اطلاعات رو به سرور بفرستیم و یا دریافت کنیم، بدون اینکه صفحه رفرش بشه و همچنین روند اجرای برنامه‌ی ما منتظر کار Ajax نمی‌مونه. Ajax مخفف Asynchronous JavaScript and XML هست.

. . .

https://dev.to/gafi...

https://dev.to/mac...

https://itnext.io/a-b...

https://javascript.info/async...

61. چند راه برای ساختن آبجکت توی جاوااسکریپت می‌شناسید؟

برای ساختن یک آبجکت توی جاوا اسکریپت سه راه وجود داره:

Object Literal

که همون روش رایج هست که به صورت زیر هست:

```
const person = {
    name: "Mark",
    sleep() {
        return "ZzZzZ";
    }
}
```

توابع Constructor

توی این روش با کلمه کلیدی new می‌تونیم از یک تابع معمولی یک آبجکت داشته باشیم:

```
function Person(name) {
  this.name = name;

  this.sleep = function() {
    return 'ZzZzZz';
  }
}

const mark = new Person("Mark");

alert(mark.sleep()); // ZzZzZz
```

بیشتر بدانید

[ویدئو] توابع Constructor | آموزش جاوااسکریپت به زبان ساده

Constructor Function که به اون توابع سازنده هم گفته میشه، یک راه سنتی برای داشتن کلاس‌ها توی جاوااسکریپت هست

متد Object.create

با استفاده از این متد می‌تونیم از یک آبجکتی که از قبل وجود داره، یک آبجکت جدید بسازیم:

```
const person = {
  sleep() {
    return "ZzZzZz";
  }
}

const mark = Object.create(person, {
  name: {
    value: 'Mark',
  },
});

alert(mark.sleep()); // ZzZzZz
```

در واقع آبجکت person به عنوان پروتوتایپ (والد) آبجکت جدید در نظر گرفته میشه.

62. فرق متدهای freeze و seal چیه؟

این متدها برای آبجکت سراسری Object هستن. seal به معنی بستن و خاتمه‌دادن هست. متد **seal** مانع اضافه شدن پراپرتی جدید به آبجکت میشه. همچنین یک پراپرتی نمی‌تونه حذف بشه. فقط میشه مقدار یک پراپرتی رو ویرایش کرد. متد freeze شبیه به seal هست اما با این تفاوت که این متد اجازه ویرایش مقدار یک پراپرتی رو نمیده.

63. فرق بین عملگر in و متد hasOwnProperty چیه؟

همونطور که توی قسمت ششم بررسی کردیم، هر دو مورد برای بررسی کردن وجود یک پراپرتی توی یک آبجکت استفاده میشه. عملگر in علاوه بر اینکه بررسی می‌کنه که آیا پراپرتی یا متد مورد نظر ما توی آبجکت وجود داره، بررسی می‌کنه که آیا پراپرتی مورد نظر ما توی prototype اون آبجکت هم وجود داره یا نه. اما متد hasOwnProperty دیگه دنبال پراپرتی مورد نظر ما توی prototype نمی‌گرده:

```
const person = {
    name: "Chris Spheeris"
}

console.log("name" in person); // true
console.log("toString" in person); // true

console.log(person.hasOwnProperty("name")); // true
console.log(person.hasOwnProperty("toString")); // false
```

64. برای مدیریت کردن عملیات ناهمگام چه راه‌هایی رو می‌شناسید؟

توی خود جاوااسکریپت با ۳ روش می‌تونیم این کار رو انجام بدیم.

- Callback Function
- پرامیس‌ها
- async/await

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

65. آیا جاوااسکریپت به حروف بزرگ و کوچیک حساسه؟

بله. همه‌ی متغیرها و توابع توی جاوااسکریپت به حروف بزرگ و کوچیک حساس هستن. یعنی دقیقا باید همونطوری استفاده بشن که تعریف شدن:

```
const Car = "Ford";
const car = "Toyota";

console.log(Car); // Ford
console.log(car); // toyota


parseFloat("20.33 sd"); // 20.33
parsefloat("20.33 sd"); // ReferenceError: parsefloat is not defined
```

بیشتر بدانید

[ویدئو] آشنایی با ساختار جاوااسکریپت | آموزش جاوااسکریپت به زبان ساده

با ساختار کلی زبان جاوااسکریپت توی قسمت آشنا می‌شیم

66. متد pop چکار می‌کنه؟

این متد آخرین المنت یک آرایه رو حذف می‌کنه. خروجی این متد، اون آیتم حذف شده هست:

```
const persons = ["Mike", "Ali", "John", "Sarah"];

console.log(persons.pop()); // Sarah
console.log(persons); // ["Mike", "Ali", "John"]
```

67. Memoization چیه؟

Memoization یعنی تابعی بسازیم که قادر باشه مقدار یا خروجی قبلی پردازش شده‌ی خودش رو به یاد داشته باشه. از خوبی استفاده از Memoization اینه که هر دفعه که تابع با آرگومان‌های مشخص فراخونی بشه دیگه عملیات اضافی صرف محاسبه‌ی مجدد خروجی نمیشه که اینکار باعث صرفه‌جویی در زمان میشه؛ اما ممکنه مقدار بیشتری از حافظه اشغال بشه.

68. یک تابع Memoization پیاده‌سازی کنین

همونطور که گفتیم یک تابع Memoization خروجی قبلی خودش با یک آرگومان خاص رو ذخیره می‌کنه. نوشتن این تابع خیلی سادس. اینجا کلوژرها به کار ما میان. این تابع یک آرگومان می‌گیره. این آرگومان می‌تونه هر تابعی باشه؛ هر تابعی که می‌خوایم خروجی اون توی حافظه ذخیره بشه:

```
function memoize(fn) {
  const cache = {};

  return function (param) {
    if (cache[param]) {
      console.log('cached');
      return cache[param];
    } else {
      const result = fn(param);
      cache[param] = result;
      console.log(`not cached`);

      return result;
    }
  }
}

const toUpper = function (str) {
  return str.toUpperCase();
}

const toUpperMemoized = memoize(toUpper);

toUpperMemoized("abcdef"); // not cached
toUpperMemoized("abcdef"); // cached
```

تابع memoize ما اول بررسی می‌کنه که آیا تابعی که پاس داده شده با یک آرگومان مشخص، از قبل توی حافظه (اینجا متغیر cache) وجود داره یا نه که اگه وجود نداشت، ابتدا تابع پاس داده شده رو با آرگومانش اجرا می‌کنه و سپس خروجی اون رو میریزه توی حافظه. پس اگه دوباره این تابع با این آرگومان دوباره فراخونی شد، خود تابع اجرا نمیشه و فقط خروجی اون که توی حافظه هست برگردونده میشه.

69. چرا typeof null یک آبجکت هست؟ چطوری null بودن رو بررسی کنیم؟

خروجی کد زیر در عین ناباوری توی جاوااسکریپت true هست:

```
typeof null === 'object' // true
```

اینکه چرا اینطوری هست باید از سازنده‌های این زبان بپرسیم. برای بررسی کردن اینکه یک عبارت null هست یا نه، از عملگر سه‌مساوی (===) استفاده می‌کنیم:

```
var value = null;

value === null; // true
```

70. کلمه کلیدی new چکار می‌کنه؟

اگه با شی‌گرایی آشنایی داشته باشید، می‌دونیم که برای نمونه ساختن از یک کلاس، باید از کلمه کلیدی new استفاده کنیم. این کار به ما یک آبجکت از کلاس مورد نظر ما برمی‌گردونه. توی جاوااسکریپت ES5 چیزی به اسم کلاس وجود نداره که بشه از اون نمونه گرفت. اما به هر حال این کار قابل اجرا هست. کلمه‌ی کلیدی new پشت یک تابع قرار می‌گیره و ما به جاوااسکریپت می‌گیم که این تابع رو اجرا کن و یک آبجکت به ما برگردون:

```
function Employee(name, position, yearHired) {
  this.name = name;
  this.position = position;
  this.yearHired = yearHired;
};

const emp = new Employee("Mario", "Software Developer",
2017);

console.log(emp); // Object { name: "Mario", position:
"Software Developer", yearHired: 2017 }
```

به توابعی که با کلمه کلیدی new فراخونی میشن، میگن Constructor functions.

. . .

https://hackernoon.com/und...

https://developer.mozilla.org/en-US...

https://dev.to/mac...

https://stackoverflow.com/qu...

خب دوستان، ۷۰ سوال مصاحبهٔ جاوااسکریپت رو با هم بررسی کردیم. باید بدونیم که اگه بخوایم جاوااسکریپت رو کامل یاد بگیریم، شاید نیاز به بررسی هزار تا سوال دیگه باشه. بهرحال امیدوارم استفاده کنین و توی مسیر موفقیت و خوشبختی باشین. علاوه بر وبسایت دیتی، می‌تونین از طریق تلگرام و لینکدین با من در ارتباط باشین ✌😉

۱۴۰۳/۰۱/۲۱