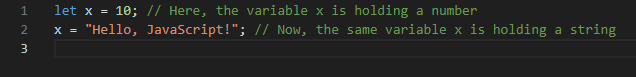
# **1-3 انقیاد نوع در جاوااسکریپت و زمان وقوع آن**

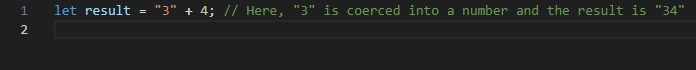
در جاوا اسکریپت، انقیاد نوع(type binding) یک مفهوم مستقیم مانند انقیاد داده نیست. با این حال، جاوا اسکریپت یک زبان تایپ پویا است، به این معنی که متغیرها می توانند مقادیر هر نوع را بدون هیچ گونه اجرای نوع نگه دارند. این در مقایسه با زبان های استاتیک تایپ شده مانند جاوا یا C ++، که در ان متغیرها با یک نوع خاص اعلام می شوند و فقط می توانند مقادیر ان نوع را نگه دارند کاملا متفاوت است.

هنگامی که شما یک مقدار را به یک متغیر در جاوا اسکریپت اختصاص می دهید، نوع متغیر با نوع مقدار اختصاص داده شده تعیین می شود. در اینجا یک مثال ساده است:



در مثال بالا، متغیر x در ابتدا به یک عدد محدود شد، اما بعدا، بدون هیچ مشکلی به یک رشته بازگشت. این انعطاف پذیری یکی از ویژگی های تایپ پویا در جاوا اسکریپت است.

اجبار نوع(Type coercion) نیز یک ویژگی رایج در جاوا اسکریپت است، جایی که نوع یک مقدار به طور خودکار به نوع دیگری بر اساس عملیات انجام شده تبدیل می شود. به عنوان مثال:



در این مورد، رشته "3" به طور خودکار به یک عدد مجبور می شود زمانی که به عدد 4 اضافه می شود.

## **1-1-3 چه زمانی انقیاد نوع در جاوا اسکریپت انجام می شود؟**

انقیاد نوع یا تعیین نوع به صورت پویا در زمان اجرا در جاوا اسکریپت اتفاق می افتد. این به این دلیل است که جاوا اسکریپت یک زبان تفسیری است و تعیین نوع با اجرای کد رخ می دهد.

نوع متغیر می تواند در طول اجرای برنامه تغییر کند، همانطور که در مثال قبلی نشان داده شده است. این ماهیت پویای انقیاد نوع برای انعطاف پذیری جاوا اسکریپت اساسی است.

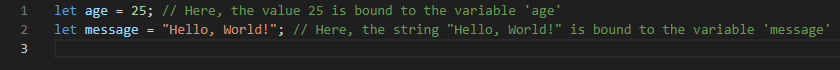
در حالی که تایپ پویا انعطاف پذیری را فراهم می کند و همچنین رخ دادن مسائل غیر منتظره مربوط به نوع را در زمان اجرا را به همراه دارد، به همین دلیل درک انقیاد نوع جاوا اسکریپت و مدیریت کردن نوع‌ها به طور موثر برای نوشتن کد قوی و قابل پیش بینی بسیار مهم است.

بنابراین، به طور خلاصه، انقیاد نوع در جاوا اسکریپت بیشتر در مورد ماهیت پویای سیستم تایپ زبان است و به صورت پویا در زمان اجرا رخ می دهد زیرا مقادیر به متغیرها اختصاص داده می شود و عملیات بر روی ان مقادیر انجام می شود.

## **2-1-3 انقیاد مقدار در جاوااسکریپت و زمان وقوع آن**

در جاوا اسکریپت، مفهوم انقیاد مقدار یک اصطلاح استاندارد در خود زبان نیست، اما می تواند مربوط به ایده "انقیاد یک مقدار به یک متغیر" یا "انقیاد یک مقدار به یک ویژگی یا عنصر" در زمینه چارچوب ها یا کتابخانه هایی باشد که انقیاد داده به UI را اداره می کنند، مانند توسعه وب.

در جاوا اسکریپت، اتصال یک مقدار به یک متغیر از طریق تخصیص متغیر انجام می شود. هنگامی که شما یک مقدار را به یک متغیر اختصاص می دهید، اساسا ان مقدار را به نام متغیر متصل می کنید. به عنوان مثال:



عمل اختصاص یک مقدار به یک متغیر، روش اساسی برای انقیاد مقادیر در جاوا اسکریپت است.

## **3-1-3 انقیاد مقادیر در زمینه UI**

هنگامی که به توسعه وب و چارچوب هایی مانند Angular، React یا Vue.js اشاره می‌شود، مفهوم انقیاد مقدار اغلب به اتصال یک مقدار از مدل داده به یک عنصر UI مانند یک فیلد ورودی، یک منطقه متن یا هر جزء دیگر اشاره دارد.

به عنوان مثال، در React، شما می توانید یک مقدار را به یک عنصر ورودی با استفاده از ویژگی مقدار(value attribute) متصل کنید، مانند این:



در این مثال، inputValue به مقدار عنصر ورودی محدود می شود، به این معنی که هنگامی که inputValue تغییر می کند، عنصر ورودی به روز می شود و هنگامی که کاربر در فیلد ورودی تایپ می کند، inputValue را به روز می کند.

## **4-1-3 چه زمانی انقیاد داده در جاوا اسکریپت انجام می شود؟**

انقیاد داده در زمینه UI یا اتصال داده ها معمولا زمانی انجام می شود که شما می خواهید تغییرات در مدل داده خود را به UI منعکس کنید و بالعکس. در چارچوب هایی مانند Angular، React یا Vue.js، این اغلب به عنوان بخشی از فرایند رندر اتفاق می افتد. هر زمان که مقدار محدود تغییر می کند، باعث به روز رسانی عنصر UI مرتبط می شود و هنگامی که کاربر با UI تعامل می کند، می تواند مقدار محدود را در مدل داده به روز کند.

زمان بندی انقیاد داده بستگی به چارچوب خاص و چرخه عمر ان دارد. به عنوان مثال، در React، به روز رسانی وضعیت یک جزء باعث رندر مجدد جزء و عناصر محدود ان می شود و UI را برای منعکس کردن حالت جدید به روز می کند. در Angular، تغییرات در مدل باعث ایجاد یک چرخه تشخیص تغییر می شود و دیدگاه را با مقادیر جدید به روز می کند.

به طور خلاصه، انقیاد داده در جاوا اسکریپت می تواند به اتصال مقادیر به متغیرها یا در زمینه توسعه وب، مقادیر متصل شده به عناصر یا اجزای UI به منظور حفظ UI در همگام سازی با مدل داده های اساسی اشاره کند. این یک جنبه اصلی ساخت رابط های کاربری پویا و تعاملی است و نقش کلیدی در توسعه وب مدرن دارد.

# 2-3 آیا تعاریف متغیرها ضمنی است یا صریح و یا هر دو نوع تعریف وجود دارد؟ با ذکر مثال توضیح داده شود.

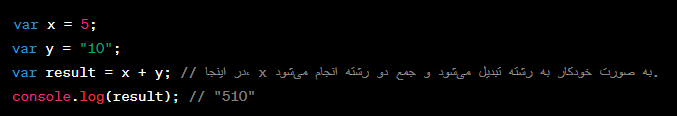
در جاوا اسکریپت، تعاریف متغیر می تواند هم ضمنی و هم صریح باشد و تمایز بین این دو ریشه در نحوه تعریف و مقداردهی اولیه متغیرها است.

در جاوااسکریپت، انقیاد نوع (Type coercion) به تبدیل یک نوع داده به نوع دیگر اشاره دارد. این انقیاد ممکن است به صورت خودکار (Implicit) یا اختیاری (Explicit) انجام شود.

انقیاد نوع خودکار (Implicit Type Coercion):

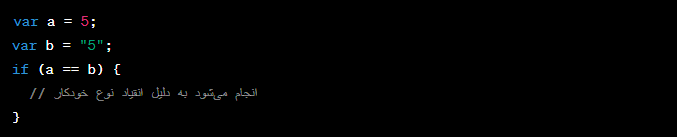
در عملیات ریاضی:

در جاوااسکریپت، اگر یک عملیات ریاضی بین دو نوع داده مختلف انجام شود، موتور جاوااسکریپت تلاش می‌کند تا به صورت خودکار نوع داده‌ها را تبدیل کند. برای مثال:



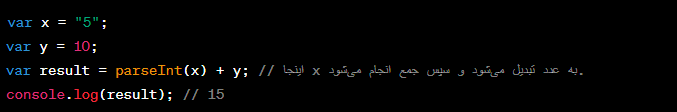
در مقایسه:

در بعضی مواقع، مقایسه اشیاء با انواع مختلف باعث انجام انقیاد نوع خودکار می‌شود. برای مثال:



انقیاد نوع اختیاری (Explicit Type Coercion):

برنامه‌نویس می‌تواند به صورت اختیاری نیز انقیاد نوع انجام دهد. این کار با استفاده از توابعی مانند parseInt، parseFloat، String و ... انجام می‌شود. برای مثال:

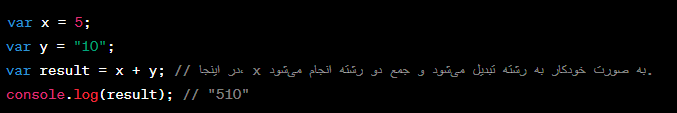


در کل، انقیاد نوع در جاوااسکریپت به صورت پویا و در زمان اجرا انجام می‌شود، و بستگی به عملیاتی که انجام می‌شود، ممکن است انقیاد نوع خودکار رخ دهد یا برنامه‌نویس مستقیماً انقیاد نوع انجام دهد.

انقیاد نوع در جاوااسکریپت به دلیل این که یک زبان تایپ دینامیک و از نوع خیلی ضعیف است، ممکن است در برخی مواقع به مشکلاتی منجر شود که برنامه‌نویسان باید مواظب آن باشند.

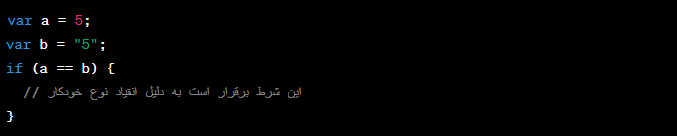
انقیاد نوع در عملیات ریاضی:

در عملیات ریاضی، انقیاد نوع خودکار می‌تواند به نتایج غیر منتظره منجر شود:



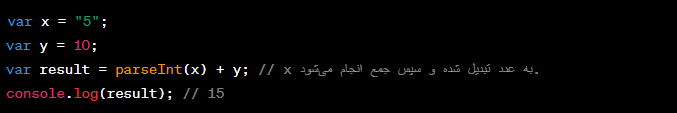
انقیاد نوع در مقایسه:

مقایسه اعداد با رشته‌ها ممکن است منجر به نتایج غیر منتظره شود:



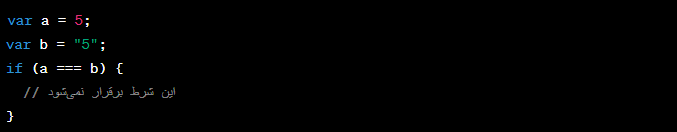
انقیاد نوع اختیاری:

برنامه‌نویسان می‌توانند از توابعی مانند parseInt یا parseFloat برای انجام انقیاد نوع اختیاری استفاده کنند:



تجنب انقیاد نوع ناخواسته:

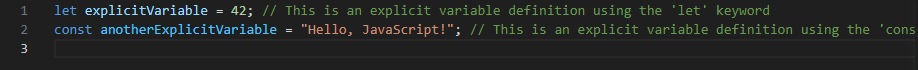
برای جلوگیری از انقیاد نوع ناخواسته، برنامه‌نویسان معمولاً از اپراتور === (برابری به همه جهات و نوع داده) به جای == (برابری به همه جهات) در مقایسه‌ها استفاده می‌کنند. این کار باعث می‌شود که نوع داده‌ها هم بررسی شود و فقط در صورت برابری نوع‌ها، مقایسه به درستی انجام شود:



در کل، انقیاد نوع در جاوااسکریپت باید با دقت مدیریت شود تا اشکالات مربوط به نوع‌ها به وجود نیاید و کد به درستی اجرا شود.

تعریف صریح متغیر

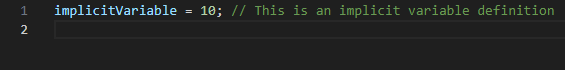
تعریف صریح متغیر زمانی اتفاق می افتد که یک توسعه دهنده، یک متغیر را با استفاده از کلمات کلیدی let، const یا var اعلام می کند و یک مقدار اولیه را در زمان اعلان اختصاص می دهد. به عنوان مثال:



در مثال بالا، هر دو متغیر به شکل صریح تعریف شده‌اند. انواع آن‌ها بر اساس نوع مقادیر اختصاص داده شده به آن‌ها (عدد و رشته و ...) تعیین می شوند.

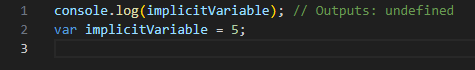
تعریف ضمنی متغیر

تعریف ضمنی متغیر زمانی اتفاق می افتد که یک مقدار به یک متغیر اختصاص داده می شود بدون اینکه به صراحت آن را با استفاده از let، const یا var اعلام کند. هنگامی که یک متغیر به طور ضمنی تعریف می شود، زمانی ایجاد می شود که شما یک مقدار را بدون اعلام صریح به ان اختصاص می دهید. به عنوان مثال:



در این مثال، implicitVariable به طور ضمنی تعریف می شود زمانی که مقدار 10 به ان اختصاص داده می شود. جاوا اسکریپت به طور خودکار نوع متغیر را بر اساس مقدار اختصاص داده شده تعیین می کند. این یک مثال از تایپ پویا است و قابلیت تعریف متغیر ضمنی جاوا اسکریپت را نشان می دهد.

مهم است که توجه داشته باشید که تمام تعریف متغیرها در جاوا اسکریپت به بالای تابع یا اسکریپت حاوی آن‌ها محدود میشوند. این بدان معنی است که تخصیص متغیرها در جایی که انتظار دارید اتفاق می افتد، تعریف واقعی به بالا محدود می رود. به عنوان مثال:



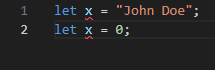
در این مورد، متغیر implicitVariable به بالای دامنه خود محدود میشود، که منجر به کنسول می شود و دستور console.log خطا را پرتاب نمی کند، بلکه خروجی تعریف نشده است.

نقش let، const و var

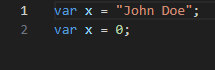
* کلمه کلیدی let در ES6 معرفی شد (2015).
* متغیرهای تعریف شده با let نمی توانند دوباره تعریف شوند.
* متغیرهای تعریف شده با let باید قبل از استفاده تعریف شوند.
* متغیرهای تعریف شده با let دارای محدوده بلوک هستند.

نمیتوان دوباره تعریف کرد

متغیرهای تعریف شده با let نمی توانند دوباره تعریف شوند. شما نمی توانید به طور تصادفی یک متغیر تعریف شده با let را دوباره تعریف کنید. به طور مثال تعریف متغیر به شکل زیر اشتباه است :

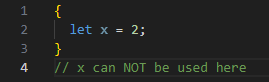


اما تعریف متغیر به شکل زیر صحیح است :

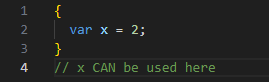


محدوده بلوک

قبل از ES6 (2015)، جاوا اسکریپت دارای محدوده جهانی و محدوده تابع بود. در ES6 دو کلمه کلیدی مهم جدید جاوا اسکریپت را معرفی شد: let و cons. این دو کلمه کلیدی محدوده بلوک را در جاوااسکریپت معرفی میکنند. متغیرهای اعلام شده در داخل یک بلوک { } از خارج از بلوک قابل دسترسی نیستند:

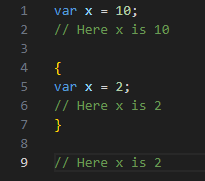


متغیرهای اعلام شده با کلمه کلیدی var محدوده بلوک ندارند. متغیرهای اعلام شده در داخل یک بلوک { } می توانند از خارج از بلوک قابل دسترسی باشند.

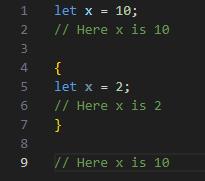


بازتعریف متغیرها

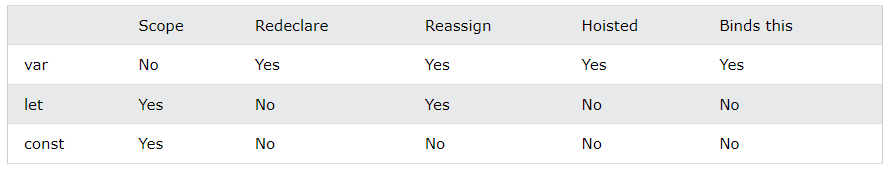
بازتعریف یک متغیر با استفاده از کلمه کلیدی var می تواند مشکلاتی را تحمیل کند. بازتعریف یک متغیر در داخل یک بلوک همچنین متغیر خارج از بلوک را دوباره تعریف خواهد کرد:



بازتعریف یک متغیر با استفاده از کلمه کلیدی let می تواند این مشکل را حل کند. بازتعریف یک متغیر در داخل یک بلوک، متغیر خارج از بلوک را دوباره تعریف نخواهد کرد:

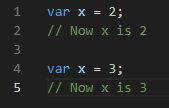


تفاوت بین var، let و const

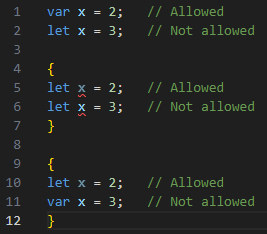


تعریف مجدد

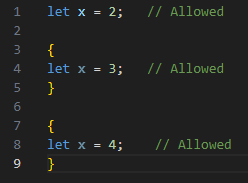
بازتعریف یک متغیر جاوا اسکریپت با var در هر جایی از یک برنامه مجاز است:



با let، بازتعریف یک متغیر در همان بلوک مجاز نیست:



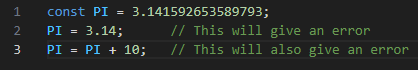
تعریف مجدد یک متغیر با let، در یک بلوک دیگر، مجاز است:



* کلمه کلیدی const در ES6 (2015) معرفی شد.
* متغیرهای تعریف شده با const نمی توانند مجددا تعریف شوند.
* متغیرهای تعریف شده با const نمی توانند دوباره اختصاص داده شوند.
* متغیرهای تعریف شده با const دارای محدوده بلوک هستند.

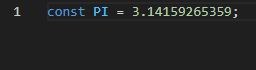
نمیتوانند دوباره تخصیص داده شوند

یک متغیر const را نمیتوان دوباره اختصاص داد:

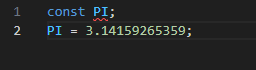


باید تخصیص داده شوند

به متغیرهای const جاوا اسکریپت باید در زمان تعریف یک مقدار اختصاص داده شود. به طور مثال تعریف متغیر به شکل زیر صحیح است:

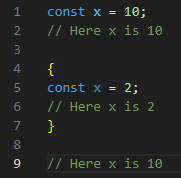


اما تعریف متغیر به شکل زیر غلط است:



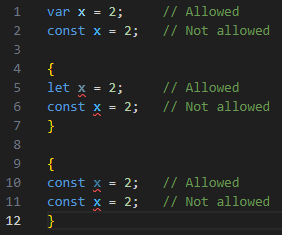
محدوده بلوک

رفتار const در بلوک مانند رفتار let است.

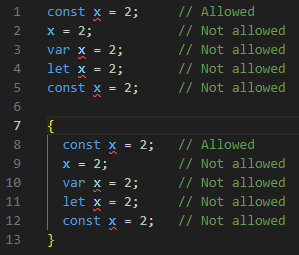


بازتعریف

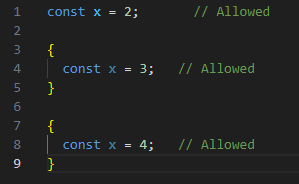
تعریف مجدد var موجود یا متغیر let برای const، در همان محدوده، مجاز نیست:



تخصیص مجدد یک متغیر const موجود، در همان محدوده، مجاز نیست:



بازتعریف یک متغیر با const، در محدوده دیگری، و یا در یک بلوک دیگر، مجاز است:



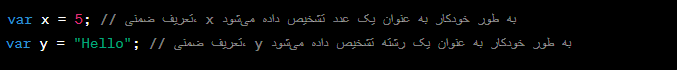
استفاده از let و const متغیر را به شکل صریح تعریف میکنند که اغلب به عنوان بهترین روش در نظر گرفته می شود زیرا به جلوگیری از انتقال مجدد تصادفی مقادیر متغیر ها کمک می کند و محدوده سطح بلوک را فراهم می کند و به افزایش پیش بینی کد کمک می کند.

به طور خلاصه، جاوا اسکریپت از تعاریف متغیر ضمنی و صریح پشتیبانی می کند. با تعاریف صریح، شما از کلمات کلیدی let یا const برای اعلام یک متغیر استفاده می کنید. در تعاریف ضمنی، یک متغیر زمانی ایجاد می شود که یک مقدار به ان اختصاص داده شود. درک تفاوت های ظریف تعریف متغیر در جاوا اسکریپت می تواند در نوشتن کد قابل پیش بینی و قابل نگهداری کمک کند.

در جاوااسکریپت، تعریف متغیرها می‌تواند به صورت ضمنی (Implicit) یا صریح (Explicit) انجام شود. همچنین، اغلب متغیرها به صورت دینامیک (Dynamic) تعریف می‌شوند، که به این معناست که نوع متغیر ممکن است در طول زمان تغییر کند.

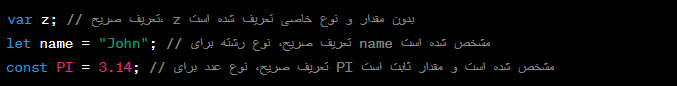
تعریف ضمنی:

تعریف ضمنی متغیرها به معنای این است که موتور جاوااسکریپت خودکار نوع متغیر را تشخیص داده و مقدار مناسب را تعیین می‌کند. این موضوع بیشتر در زمان انتساب (assignment) اتفاق می‌افتد.



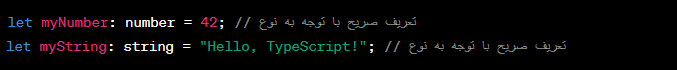
تعریف صریح:

تعریف صریح متغیرها به معنای این است که برنامه‌نویس به صورت صریح نوع متغیر را مشخص می‌کند. این کار با استفاده از کلمه‌های کلیدی مانند var، let، یا const انجام می‌شود.



تعریف صریح با توجه به نوع:

تا ES6، تعریف صریح نوع در جاوااسکریپت امکان‌پذیر نبود. اما با معرفی let و const در ES6، برنامه‌نویسان می‌توانند به صورت صریح نوع متغیر را مشخص کنند.

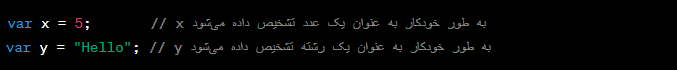


در مثال بالا، myNumber به عنوان یک عدد و myString به عنوان یک رشته تعریف شده‌اند. این قابلیت اغلب در سیستم‌های جاوااسکریپت توسعه یافته‌تر، مانند TypeScript، مورد استفاده قرار می‌گیرد.

تعریف متغیرها در جاوااسکریپت:

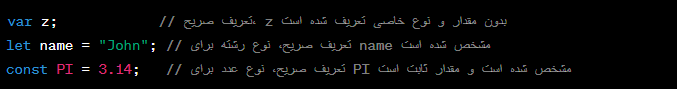
1. تعریف ضمنی:

در تعریف ضمنی، موتور جاوااسکریپت خودکار نوع متغیر را تشخیص داده و مقدار مناسب را تعیین می‌کند. این به این معناست که شما نیازی به صراحتًا مشخص کردن نوع متغیر ندارید.



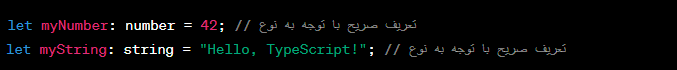
2. تعریف صریح:

تعریف صریح به این معناست که برنامه‌نویس نوع متغیر را به صورت صریح مشخص می‌کند. این عمل با استفاده از کلمات کلیدی مانند var، let یا const انجام می‌شود.



3. تعریف صریح با توجه به نوع:

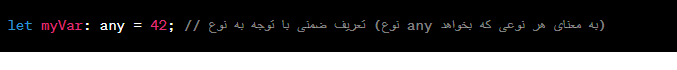
در برخی از توسعه‌های جدید جاوااسکریپت، مانند TypeScript، این امکان وجود دارد که برنامه‌نویس نوع متغیر را صریحاً مشخص کند.



در اینجا، myNumber به عنوان یک عدد و myString به عنوان یک رشته با توجه به نوع مشخص شده‌اند.

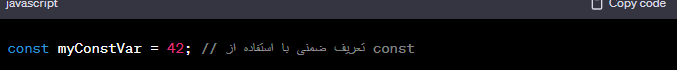
4. تعریف ضمنی با توجه به نوع:

از ES6 به بعد، با معرفی let و const، تعریف ضمنی نیز می‌تواند با توجه به نوع انجام شود.



5. تعریف ضمنی با استفاده از: const

از آنجایی که const متغیرها را ثابت می‌کند، تعریف ضمنی با استفاده از const ممکن است در برخی موارد باعث ایجاد کد کمتری با خطا شود.



با این توضیحات، شما می‌توانید با استفاده از تعریف ضمنی یا صریح، به وفور با متغیرها در جاوااسکریپت کار کنید و از آن‌ها به عنوان داده‌های دینامیک و قابل تغییر بهره‌مند شوید.

# **3-3 آیا در جاوااسکریپت متغیرهای ایستا در پشته وجود دارد؟ توضیح دهید.**

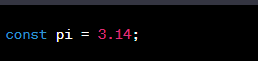
در جاوااسکریپت، متغیرهای ایستا به مفهوم متغیرهایی که در پشته (stack) ذخیره می‌شوند، معنا نمی‌شود. پشته اصطلاحا برای ذخیره داده‌ها و مراجع به توابع در زمان اجرا استفاده می‌شود، اما متغیرها اصولا در حافظه (heap) ذخیره می‌شوند.

در جاوااسکریپت، متغیرها در حافظه ذخیره می‌شوند و به طور مستقیم از پشته استفاده نمی‌کنند. پشته بیشتر برای مدیریت فرآیندهای توابع و استفاده از حافظه موقتی برای ذخیره مقادیر محلی توابع استفاده می‌شود.

در جاوااسکریپت، مفهوم "متغیرهای ایستا" به صورت صریح وجود ندارد. جاوااسکریپت یک زبان برنامه‌نویسی دینامیک است و متغیرهای آن به صورت دینامیک در حافظه ذخیره می‌شوند.

متغیرهای ایستا به طور کلی به متغیرهایی اشاره دارند که مقدار آن‌ها در طول زمان تغییر نمی‌کند. در جاوااسکریپت، این مفهوم به صورت صریح وجود ندارد. با این حال، می‌توانید از کلمه کلیدی `const` برای تعریف متغیرهای ثابت استفاده کنید که مقدار آن‌ها پس از تخصیص قابل تغییر نباشد.

مثال:



در این مثال، `pi` به عنوان یک متغیر ثابت تعریف شده است و مقدار آن پس از اولین تخصیص قابل تغییر نیست. این متغیر به نوعی به مفهوم متغیرهای ایستا نزدیکتر است، اما همچنان به عنوان یک متغیر دینامیک در جاوااسکریپت در نظر گرفته می‌شود.

بنابراین، مفهوم متغیرهای ایستا که به معنای ذخیره در پشته باشد، در جاوااسکریپت صحیح نیست. متغیرها در این زبان به صورت دینامیک در حافظه ذخیره می‌شوند و ممکن است در طول اجرا تغییر کنند.

آیا در جاوااسکریپت متغیرهای پویا در پشته وجود دارد؟ توضیح دهید.

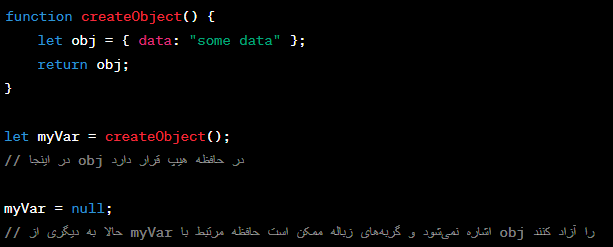
در جاوااسکریپت، مفهوم "متغیرهای پویا در پشته" به طور کلی درست نیست. در زبان‌های برنامه‌نویسی مانند جاوا و C++، مفهوم پویا بودن (dynamic) به ارتباط با تخصیص حافظه در زمان اجرا مرتبط است. این زبان‌ها می‌توانند به صورت پویا حافظه را تخصیص دهند یا آن را آزاد کنند.

در جاوااسکریپت، متغیرها به صورت دینامیک در حافظه ذخیره می‌شوند، اما در جاوااسکریپت، ما نیازی به مدیریت مستقیم حافظه نداریم و تمام برنامه ها به شکل پویا عمل میکنند.

در جاوااسکریپت، متغیرهای اصطلاحاً در حافظه هیپ (Heap) قرار می‌گیرند و مفهوم "متغیرهای پویا در پشته" به صورت مستقیم در این زبان وجود ندارد. در جاوااسکریپت، حافظه پویا به صورت خودکار توسط سطل زباله (garbage collectors) مدیریت می‌شود.

وقتی یک متغیر در جاوااسکریپت تعریف می‌شود و به آن مقداری اختصاص داده می‌شود، متغیر و مقدارش در حافظه هیپ قرار می‌گیرند. اگر دیگر به آن متغیر ارجاعی وجود نداشته باشد، گربه‌های زباله پیش می‌روند و متغیرها و اشیاء غیرقابل دسترس را از حافظه حذف می‌کنند.

مثال:



در این مثال، `obj` یک متغیر است که در حافظه هیپ قرار دارد. هنگامی که `myVar` به `null` تنظیم می‌شود، دیگر به `obj` اشاره نمی‌شود و گربه‌های زباله ممکن است حافظه مرتبط با `obj` را آزاد کنند. این فرآیند مدیریت حافظه به صورت خودکار و توسط محیط اجرایی جاوااسکریپت انجام می‌شود، بنابراین نیاز به دخالت دستی بر روی پشته وجود ندارد.

بنابراین، می‌توان گفت که در جاوااسکریپت، متغیرها به صورت پویا در حافظه ذخیره می‌شوند، اما نه به صورت مستقیم در پشته. اطلاعات مربوط به توابع و محیط اجرایی در پشته قرار دارند، اما این با مفهوم متغیرهای پویا به طور مستقیم مرتبط نیست.

# **4-3 آیا در جاوااسکریپت متغیرهای پویادر هیپ به طور صریح وجود دارد؟ توضیح دهید.**

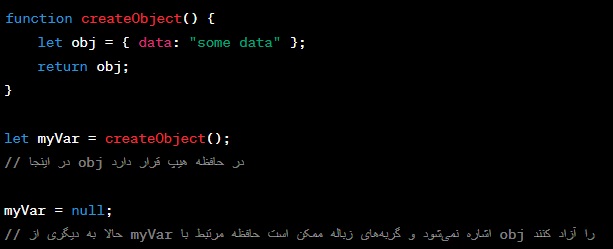
در جاوااسکریپت، تا حد زیادی مدیریت حافظه به صورت خودکار انجام می‌شود و توسعه‌دهندگان نیازی به مستقیم تخصیص یا آزاد کردن حافظه ندارند. در واقع، مفهوم "متغیرهای پویا در هیپ" به طور مستقیم در جاوااسکریپت وجود ندارد.

متغیرها در جاوااسکریپت به صورت دینامیک در حافظه ذخیره می‌شوند و به صورت خودکار توسط گربه‌های زباله مدیریت می‌شوند. گربه‌های زباله به صورت دوره‌ای اجرا می‌شوند و اشیاءی که دیگر به آن‌ها ارجاعی وجود ندارد، حذف می‌شوند.

به عبارت دیگر، توسعه‌دهندگان در جاوااسکریپت نیازی به تخصیص یا آزاد کردن حافظه به صورت صریح ندارند و این عملیات‌ها به صورت اتوماتیک انجام می‌شوند. در نتیجه، متغیرها در هیپ (heap) ذخیره می‌شوند و مدیریت حافظه به صورت متقابل با سطل زباله(Garbage Collector) انجام می‌شود.

در جاوااسکریپت، مفهوم "متغیرهای پویا در هیپ" به صورت صریح وجود ندارد. در این زبان، متغیرها به صورت دینامیک در حافظه ذخیره می‌شوند و تخصیص و آزادسازی حافظه به صورت خودکار توسط سطل زباله (garbage collectors) انجام می‌شود.

مثال:



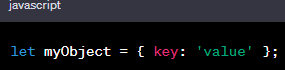
در این مثال، `obj` یک شیء است که در حافظه هیپ قرار دارد. هنگامی که `myVar` به `null` تنظیم می‌شود، دیگر به `obj` اشاره نمی‌شود و گربه‌های زباله ممکن است حافظه مرتبط با `obj` را آزاد کنند. این فرآیند توسط محیط اجرایی جاوااسکریپت به صورت خودکار انجام می‌شود و نیازی به مدیریت صریح حافظه در هیپ نداریم.

بنابراین، متغیرها در جاوااسکریپت به صورت پویا در حافظه ذخیره می‌شوند و مفهوم متغیرهای پویا در هیپ به صورت صریح وجود ندارد.

# **5-3 آیا در جاوااسکریپت متغیرهای پویادر هیپ به طور ضمنی وجود دارد؟ توضیح دهید.**

بله، در جاوااسکریپت متغیرهای پویا (dynamic variables) در حافظه هیپ (heap) به طور ضمنی وجود دارند. زبان‌هایی که از مدیریت حافظه خودکار استفاده می‌کنند، مانند جاوااسکریپت، اشیاء (objects) به صورت پویا در حافظه هیپ ایجاد می‌شوند.

وقتی یک متغیر از نوع شیء (object) در جاوااسکریپت تعریف می‌شود، این شیء در حافظه هیپ قرار می‌گیرد. به عنوان مثال:

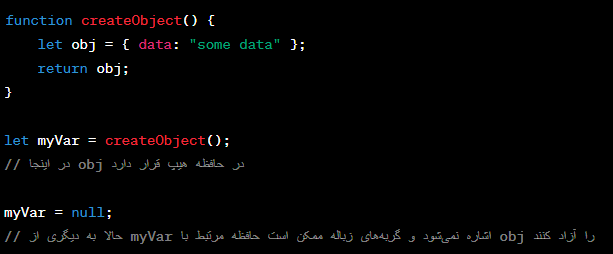


در اینجا، `myObject` یک متغیر است که یک شیء را نمایندگی می‌کند و این شیء در حافظه هیپ ذخیره می‌شود. گربه‌های زباله جاوااسکریپت به طور دوره‌ای حافظه را پیمایش کرده و اشیاءی که دیگر به آن‌ها ارجاعی وجود ندارد را تشخیص داده و حذف می‌کنند.

از آنجا که تخصیص و آزادسازی حافظه به صورت خودکار انجام می‌شود، توسعه‌دهندگان نیازی به نگرانی در مورد مدیریت حافظه به صورت صریح ندارند و متغیرهایشان به صورت ضمنی در حافظه هیپ ذخیره می‌شوند.

بله، در جاوااسکریپت متغیرها و اشیاء در حافظه هیپ به طور ضمنی وجود دارند. زمانی که شیء‌هایی ایجاد می‌شوند و به متغیرها اختصاص داده می‌شوند، این شیء‌ها در حافظه هیپ قرار می‌گیرند و مدیریت حافظه توسط گربه‌های زباله انجام می‌شود.

مثال:



در این مثال، `obj` یک شیء است که به آن اختصاص داده شده و در حافظه هیپ قرار دارد. وقتی `myVar` به `null` تنظیم می‌شود، دیگر به `obj` اشاره نمی‌شود و گربه‌های زباله ممکن است حافظه مرتبط با `obj` را آزاد کنند.

این روند بهبود کارایی و مدیریت حافظه را ساده‌تر می‌کند و توسعه‌دهندگان نیازی به مدیریت صریح حافظه ندارند. حافظه‌هایی که دیگر به آن‌ها اشاره ایجاد نشده باشد، توسط گربه‌های زباله حذف می‌شوند.

# **6-3 متغیر های ایستا در جاوااسکریپت چگونه پیاده سازی شده اند؟**

در جاوااسکریپت، متغیرهای ایستا با استفاده از const پیاده‌سازی می‌شوند. کلمه‌ی const برای تعریف متغیرها به کار می‌رود که مقدار آن‌ها در طول اجرای برنامه تغییر نخواهد کرد. این کلمه کلیدی در ES6 (ECMAScript 2015) معرفی شد و تا کنون به عنوان روش اصلی برای تعریف متغیرهای ایستا در جاوااسکریپت مورد استفاده قرار می‌گیرد.



در این مثال، pi به عنوان یک متغیر ایستا با استفاده از const تعریف شده است. این متغیر نمی‌تواند در طول اجرای برنامه تغییر کند و اگر تلاش کنید مقدار آن را تغییر دهید، با یک خطا مواجه خواهید شد.

مزایا و کاربردها:

ثبات کد (Code Stability): استفاده از متغیرهای ایستا باعث می‌شود که مقدار متغیرها در طول زمان تغییر نکند و کد ثابت‌تر و قابل‌ترکیب‌تری ارائه شود.

پیش‌بینی مقدار (Value Predictability): اگر یک متغیر با const تعریف شود، برنامه‌نویسان و تا اندازه‌ای حتی خوانندگان کد می‌توانند انتظار داشته باشند که مقدار آن تا زمان اجرای برنامه تغییر نخواهد کرد.

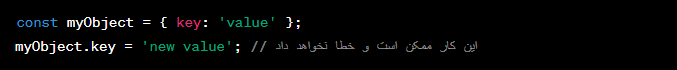
کد خواناتر (Readability): تعریف متغیرهای ایستا با const می‌تواند کد را خواناتر کند و افراد دیگری که کد شما را مشاهده می‌کنند را در درک مفهوم و مقصود کمک کند.

نکات مهم:

در صورتی که متغیری با const تعریف شود، باید به یکبار مقداردهی شود. این یعنی شما نمی‌توانید یک const بدون مقداردهی اولیه اعلام کنید.



متغیرهای ایستا با const می‌توانند اشیاء (Objects) و آرایه‌ها را نیز در خود نگه‌داری کنند. با این حال، تغییر ویژگی‌های داخلی یک شیء یا آرایه با const ممکن است باعث شود خطا رخ دهد، اما شناخته‌شده‌اند.



در کل، متغیرهای ایستا با استفاده از const یکی از ابزارهای مهم برای بهبود قابلیت خواندن و پیش‌بینی کد در جاوااسکریپت هستند.

به چند نکته بیشتر در مورد متغیرهای ایستا (const) در جاوااسکریپت بپردازیم:

1. تفاوت با let و var:

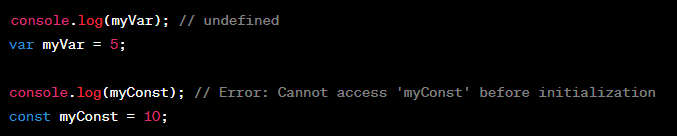
let برای تعریف متغیرهای محلی استفاده می‌شود که می‌توانند در یک بلوک مشخص (مثل یک حلقه یا یک شرط) تغییر کنند.

const نیز برای تعریف متغیرهای محلی استفاده می‌شود، اما مقدار آن‌ها نمی‌تواند تغییر کند.

var که در ES5 معرفی شد، نحوه‌ی تعریف متغیرها را مشخص می‌کند و می‌تواند دچار مشکلاتی مانند hoisting شود.

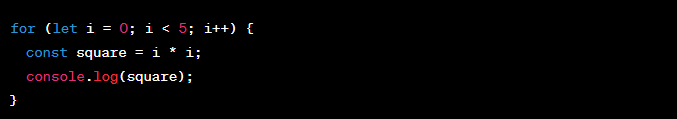
2. Hoisting در متغیرهای const:

متغیرهای const همچنان تحت hoisting هستند. این بدان معناست که می‌توانید یک متغیر const را پیش از تعریف آن استفاده کنید، اما به دلیل خصوصیات const که یک بار مقداردهی شده‌اند نمی‌توانند مقدار دیگری را دریافت کنند.



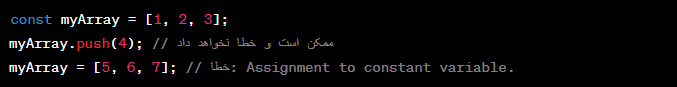
3. استفاده در حلقه‌ها:

استفاده از const در حلقه‌ها می‌تواند کمک کند تا از تغییر ناخواسته مقادیر جلوگیری شود.



4. آرایه‌ها و: const

در صورتی که یک آرایه با const تعریف شود، می‌توانید عناصر آرایه را تغییر دهید، اما نمی‌توانید یک آرایه جدید نسبت به متغیر ایجاد کنید.



5. تفاوت با ایستایی در زبان‌های دیگر:

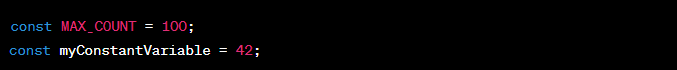
در زبان‌های دیگر مفهوم "ایستا" به معنایی مشخص تر از آن چیزی است که در جاوااسکریپت با const تعبیه شده است. در جاوااسکریپت، const به معنای "قابل تغییر نیست" است، اما این به معنای این نیست که مقدار آن برابر با یک مقدار ثابت و ایستا است.

6. توصیه برای استفاده از const:

استفاده از const به عنوان روش اولویتی برای تعریف متغیرها تا جای ممکن توصیه می‌شود. این کمک می‌کند کد شما خواناتر و قابل‌ترکیب‌تر باشد و در صورتی که مقدار یک متغیر تغییر نخواهد کرد، از این امکان بهره‌مند شوید.

7. نامگذاری متغیرهای ایستا:

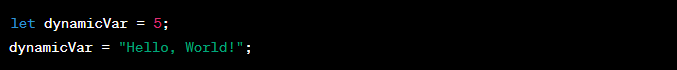
بهتر است متغیرهای ایستا با const نام‌هایی داشته باشند که از حروف بزرگ تشکیل شده و به صورت snake\_case یا camelCase نوشته شوند تا در کد به راحتی قابل تشخیص باشند.



با استفاده از متغیرهای ایستا، می‌توانید کد خود را از اشکالات مرتبط با تغییر ناخواسته مقادیر در طول زمان محافظت کنید و قابلیت خواندن و نگهداری کد خود را افزایش دهید.

# **7-3 متغیر های پویا در جاوااسکریپت چگونه پیاده سازی شده اند؟**

در جاوااسکریپت، متغیرهای پویا با استفاده از let یا var پیاده‌سازی می‌شوند. این کلمات کلیدی برای تعریف متغیرهای قابل تغییر در طول اجرای برنامه به کار می‌روند. این متغیرها به عبارتی قابلیت تغییر نوع و مقدار را دارند.

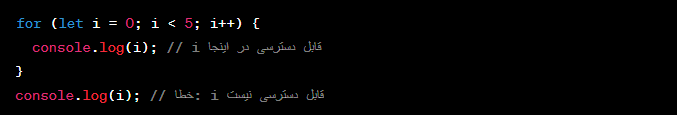


در این مثال، dynamicVar با استفاده از let تعریف شده و ابتدا به یک عدد اختصاص داده شده و سپس مقدار آن به یک رشته تغییر کرده است.

نکات مهم:

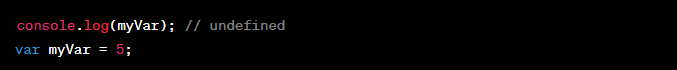
let در ES6:

کلمه کلیدی let در ES6 معرفی شد و به خوبی از var جایگزین شده است. از let به دلیل مزایای بیشتری مانند داشتن scope بلوکی (block-scoped) استفاده می‌شود.

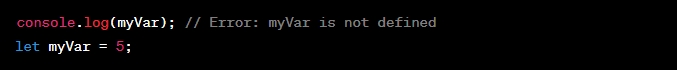


var و Hoisting:

var نیز همچنان در جاوااسکریپت استفاده می‌شود، اما دارای خصوصیت hoisting است که به این معناست که متغیرها با var ممکن است پیش از تعریف آن‌ها مورد دسترسی قرار گیرند.

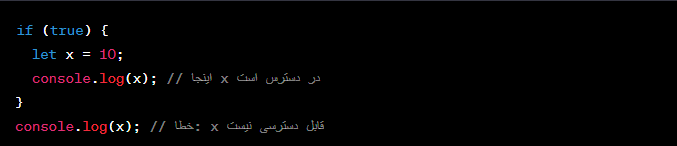


در مقایسه با let که اینگونه کار نمی‌کند:



استفاده در محدوده‌های مختلف:

let اطلاعات را در scope بلوکی (بین {}) نگه می‌دارد و این به معنای آن است که می‌توانید متغیرهایی با همین نام را در بلوک‌های مختلف بسازید.

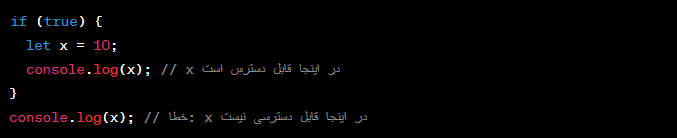


استفاده از let به عنوان روش اولویتی برای تعریف متغیرها تا جای ممکن توصیه می‌شود، زیرا این کلمه کلیدی اصطلاحاً مزایای بیشتری نسبت به var دارد و اشکالات زیادی که در var وجود دارد را برطرف کرده است. استفاده از let باعث ایجاد کدی خواناتر و قابل‌ترکیب‌تر می‌شود.

متغیرهای پویا در جاوااسکریپت با استفاده از let و گاهی هم var تعریف می‌شوند. دو کلمه‌ی کلیدی let و var به شما اجازه می‌دهند تا متغیرها را تعریف کرده و به آن‌ها مقدار اختصاص دهید. این متغیرها می‌توانند در طول اجرای برنامه تغییر کنند، به خصوص اگر با let تعریف شده باشند.

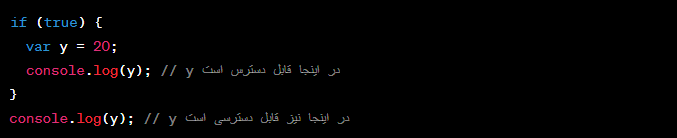
1. استفاده از let:

let به شما این امکان را می‌دهد که متغیرها را در scope بلوکی (block scope) تعریف کنید. این به معنای این است که متغیرها تا انتهای بلوک (بین {}) محدود می‌شوند.



2. استفاده از var:

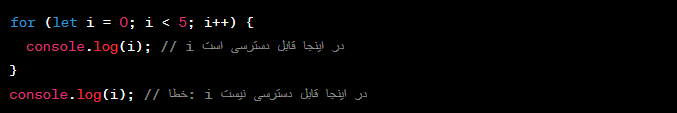
var همچنان در جاوااسکریپت وجود دارد، اما به دلیل خصوصیاتی نظیر hoisting و قابل دسترسی در خارج از scope بلوکی که تعریف شده‌اند، کمتر مورد استفاده قرار می‌گیرد.



به عنوان یک نکته مهم، var ها hoisting دارند که به این معنی است که متغیرها می‌توانند پیش از تعریف آن‌ها مورد استفاده قرار گیرند.

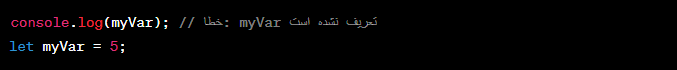
3. مبنای Block Scope در let:

let به شما این امکان را می‌دهد که scope بلوکی (بین {}) را ایجاد کنید. این بهبود از نظر اصول برنامه‌نویسی و ایمنی کد است.



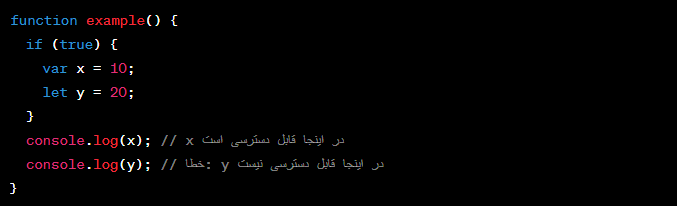
4. Hoisting با let:

let نیز hoisting دارد، اما با این تفاوت که اگر متغیر قبلاً تعریف شده باشد ولی هنوز مقداردهی نشده باشد، به undefined تنظیم می‌شود.



5. تفاوت در محدودیت‌های Block Scope:

let به عنوان یک بهترین عملکرد در محدوده‌های بلوکی استفاده می‌شود، در حالی که var به محدودیت‌های قدیمی‌تری از قبیل محدوده‌های تابعی (function scope) محدود می‌شود.



در کل، به منظور ایجاد متغیرهای پویا که مقدار و نطام تغییرات در طول اجرا دارند، از let برخوردار از Block Scope استفاده کنید. استفاده از let به کدتان انعطاف بیشتری می‌دهد و اشکالات مرتبط با var را کاهش می‌دهد.

# **8-3 مقایسه سرعت تخصیص متغیر های ایستا و پویا را در جاوااسکریپت**

در جاوااسکریپت، سرعت تخصیص متغیرهای ایستا (با `const`) و متغیرهای پویا (با `let` یا `var`) تأثیر مستقیمی بر عملکرد ندارد، زیرا جاوااسکریپت یک زبان اجرایی (interpreted language) است و نه یک زبان کامپایل شده (compiled language). در زبان‌های کامپایل شده، تفاوت در تخصیص متغیرها ممکن است به دلیل فاصله زمانی کمتر بین تولید کد و اجرا مشهود باشد، اما در جاوااسکریپت، این تفاوت به دلیل ماهیت دینامیک و تفسیری زبان کمتر قابل مشاهده است.

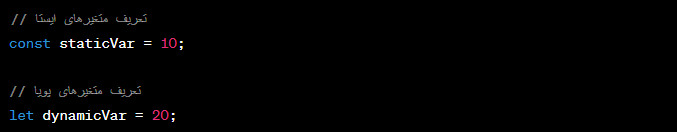
در واقع، در محیط اجرایی جاوااسکریپت، به طور کلی اهمیت زیادی به نحوه تعریف متغیر (ایستا یا پویا) نمی‌دهند و بیشتر به الگوها و عملکرد کد توجه می‌کنند.

متغیرهای ایستا با `const` تعریف شده‌اند و می‌توانند بهترین عملکرد را در برخی موارد ارائه دهند. از طرف دیگر، متغیرهای پویا با `let` یا `var` قابلیت انجام تغییرات دارند که در برخی موارد انعطاف‌پذیری بیشتری ارائه می‌دهد.

توصیه می‌شود که در انتخاب نوع متغیر بر اساس نیازها و اهداف کد نظر داشته باشید تا برنامه‌ی شما خواناتر و قابل‌ترکیب‌تر باشد. اهمیت اصلی باید به خواناتری و نگهداری کد داده شود تا بهینه‌سازی‌های جزئی در این زمینه تأثیر محدودی داشته باشند.

سرعت تخصیص متغیرها در جاوااسکریپت به طور کلی از نحوه اجرای کد و ماهیت زبان تفسیری (interpreted) آن تاثیر می‌پذیرد. در زبان‌های تفسیری مانند جاوااسکریپت، کد به صورت مستقیم توسط موتور اجرایی (مانند موتور V8 در مرورگرها یا موتور Node.js) تفسیر می‌شود و نیازی به یک مرحله کامپایل (compile) قبل از اجرا ندارد.

بنابراین، تفاوت در تخصیص متغیرهای ایستا (const) و پویا (let یا var) در جاوااسکریپت به نظر می‌رسد که تأثیر مستقیمی بر عملکرد کد نداشته باشد. مهمترین نکته این است که متغیرها به صورت دینامیک تایپ (Dynamic Typing) در جاوااسکریپت تعریف می‌شوند که به برنامه‌نویس اجازه می‌دهد تا در طول زمان نوع متغیرها را تغییر دهد.



در این مثال، هر دو staticVar و dynamicVar به صورت پویا تعریف شده‌اند، اما staticVar با const تعریف شده که نشان‌دهنده این است که مقدارش تا زمان اجرای برنامه تغییر نخواهد کرد. این تغییر در تعریف متغیرها به تنهایی تأثیر چندانی بر سرعت اجرا نخواهد گذاشت.

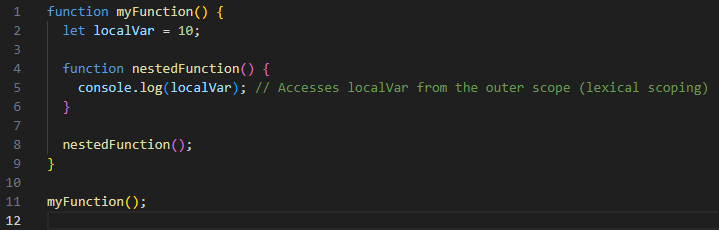
برای بهینه‌سازی کد، می‌توانید تلاش کنید که از متغیرهای ایستا (const) در جاهایی که مقدار ثابت استفاده می‌شود و از متغیرهای پویا (let یا var) در جاهایی که نیاز به تغییر مقدار وجود دارد، بهره ببرید. اما تفاوت در سرعت تخصیص به میزان قابل توجهی نبوده و بیشتر توجه به خواناتری و نگهداری کد باید داده شود.

# **9-3 آیا حوزه تعریف در زبان جاوااسکریپت ایستا است یا پویا؟ با ذکر مثال توضیح دهید.**

در جاوا اسکریپت، دامنه تعاریف متغیر در درجه اول استاتیک است، زیرا بر اساس جایی که متغیرها در کد اعلام می شوند و نه در شرایط زمان اجرا تعیین می شوند. با این حال، ماهیت پویای جاوا اسکریپت همچنین رفتارهای خاصی را معرفی می کند که می تواند به صورت پویا در مورد این کلمه کلیدی و توابع تو در تو در نظر گرفته شود.

محدوده استاتیک

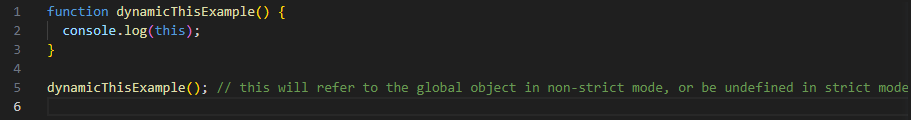
در محدوده استاتیک (همچنین به عنوان محدوده واژگانی شناخته می شود)، دامنه یک متغیر با مکان ان در کد منبع و زمینه واژگانی که در ان اعلام شده است تعیین می شود. موتور جاوا اسکریپت در زمان کامپایل می داند که باید به کدام محدوده متغیر برای حل یک متغیر نگاه کند. به عنوان مثال:



در این مورد، NestedFunction به دلیل محدوده استاتیک به localVar دسترسی دارد. محیط واژگانی NestedFunction شامل دامنه myFunction است که به ان اجازه دسترسی به localVar را می دهد.

محدوده داینامیک

در حالی که محدوده متغیر عمدتا استاتیک است، استفاده از کلمه کلیدی this در جاوا اسکریپت نوعی رفتار دامنه پویا را معرفی می کند. مقدار this به صورت پویا بر اساس چگونگی فراخوانی یک تابع تعیین می شود، نه جایی که تعریف می شود. این رفتار به انعطاف پذیری جاوا اسکریپت کمک می کند. مثال زیر را در نظر بگیرید:



مقدار this در dynamicThisExample بستگی به این دارد که چگونه تابع فراخوانی می شود، و ان را به یک شکل از محدوده پویا در زمینه اجرای تابع تبدیل می کند.

به طور خلاصه، دامنه تعاریف متغیر در جاوا اسکریپت در درجه اول استاتیک است. با این حال، رفتار پویای کلمه کلیدی this، شکلی از محدوده پویا را در توابع معرفی می کند. درک تعامل بین محدوده واژگانی، کلمه کلیدی this برای نوشتن کد جاوا اسکریپت قابل پیش بینی و قابل نگهداری بسیار مهم است.

# **10-3 پیاده‌سازی حوضه تعریف پویا در جاوا اسکریپت :**

اضافه کردن حوزه تعریف پویا (Dynamic Scope) به جاوااسکریپت یک تغییر اساسی در طراحی زبان است و نیاز به تغییرات گسترده‌ای دارد. در حال حاضر، جاوااسکریپت از حوزه تعریف لکسیکال (Lexical Scope) استفاده می‌کند که به عنوان Scope یا Scope Chain نیز شناخته می‌شود. در حوزه تعریف لکسیکال، Scope ها بر اساس مکان تعریف توابع در کد قرار دارند و از این رو، دسترسی به متغیرها بر اساس نحوه تو در تویی از یکدیگر تعیین می‌شود.

برای ایجاد حوزه تعریف پویا، نیاز به تغییرات در نحوه عملکرد Scope ها و مکانیسم تعیین متغیرها داریم. این تغییرات به شدت تأثیرگذار بوده و احتمالاً با ساختار فعلی جاوااسکریپت تداخل دارد. با این حال، می‌توانیم یک سناریو ساده را برای نمونه توضیح دهیم:

سناریو ساده:

فرض کنید که می‌خواهیم یک ویژگی تعریف پویا به جاوااسکریپت اضافه کنیم. برای این کار، یک نحوه اجرا و تعیین Scope دیگر بر اساس اجرا‌ها ایجاد می‌کنیم.

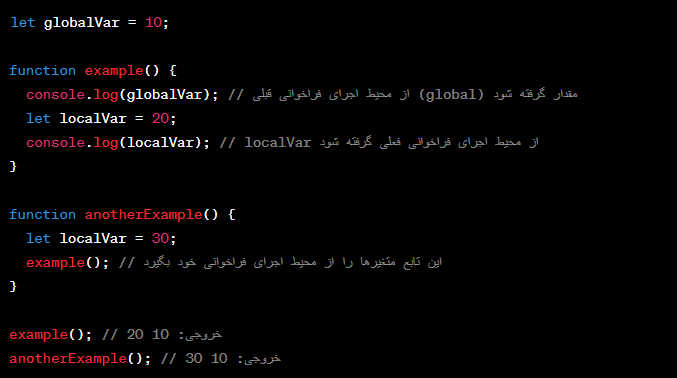
ایده اصلی:

متغیرهای تعریف شده در یک تابع به صورت پویا بر اساس محیط اجرای فراخوانی‌ها تغییر کنند.

تابع‌ها اطلاعات Scope خود را در هر فراخوانی ذخیره کنند.

اگر یک متغیر در محیط اجرا وجود نداشته باشد، به محیط اجرای قبلی نگاه کند.

پیاده‌سازی ساده:



این کد ساده نمایانگر یک ایده مبتنی بر حوزه تعریف پویا است. در هر فراخوانی تابع، متغیرها از محیط اجرای آن تابع گرفته شده و اگر یک متغیر در محیط اجرای فعلی وجود نداشته باشد، به محیط اجرای قبلی نگاه می‌کنند.

جاوااسکریپت در حال حاضر از حوزه تعریف لکسیکال استفاده می‌کند که به این معناست که Scope ها بر اساس مکان تعریف توابع در کد قرار دارند. این به این معناست که دسترسی به متغیرها بر اساس نحوه تو در تویی از یکدیگر تعیین می‌شود. حتی اگر یک تابع در داخل یک تابع دیگر تعریف شده باشد، Scope آن تابع داخلی محدود به Scope تابع بیرونی خود خواهد بود.

برای اجرای این تغییر و افزودن حوزه تعریف پویا، نیاز به تغییرات گسترده‌ای در موتورهای اجرای جاوااسکریپت (مثل V8 که توسط Google برای مرورگر Chrome ایجاد شده است) و استانداردهای جاوااسکریپت داریم. این تغییرات بسیار پیچیده و تأثیرگذار هستند و نیاز به تعیین استانداردهای جدید، تغییرات در موتورهای جاوااسکریپت، و حتی اصلاح یا تغییر در کد‌های موجود دارند.

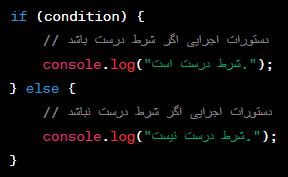
یک پیاده‌سازی واقعی این تغییرات بسیار پیچیده و بسیار وسیع خواهد بود و احتمالاً نیاز به مشارکت جامع از سوی توسعه‌دهندگان و تصمیم‌گیران جاوااسکریپت دارد. در زمینه توسعه زبان‌های برنامه‌نویسی، تغییرات اساسی مانند این نیاز به بررسی دقیق، آزمایش، و تأثیرات کلان روی اکوسیستم جاوااسکریپت دارند.

در حال حاضر، استفاده از حوزه تعریف لکسیکال در جاوااسکریپت به‌عنوان یک استاندارد تقریباً ثابت شده است و تغییر آن به یک سیاق بزرگتر نیاز دارد که از ابتدا در طراحی زبان لحاظ شود.

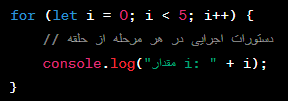
# **11-3 بلوک ها در جاوااسکریپت چگونه تعریف شده اند؟**

در جاوااسکریپت، بلوک‌ها (blocks) به صورت گروهی از ادامه دادن دستورات می‌باشند که می‌توانند به عنوان یک واحد اجرایی در نظر گرفته شوند. بلوک‌ها در اغلب موارد با دستورات شرطی (مانند `if` یا `else`) و حلقه‌ها (مانند `for` یا `while`) مرتبط هستند. یک بلوک با استفاده از زوج آکولاد `{}` تعریف می‌شود.

مثال از یک بلوک در یک دستور شرطی (`if`):



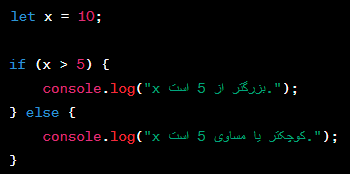
یک مثال دیگر از یک بلوک در یک حلقه (`for`):



در اینجا، بلوک داخلی هر دستور `if` یا `for` توسط آکولاد‌ها `{}` تعیین می‌شود. دستورات داخل بلوک باید توسط آکولاد‌ها محصور شوند، حتی اگر تنها یک دستور باشند. استفاده از آکولادها از ابهام جلوگیری می‌کند و کد را برای خوانایی بهتر افزایش می‌دهد.

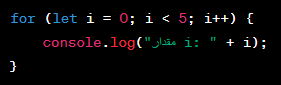
در جاوااسکریپت، بلوک‌ها با استفاده از آکولادها `{}` تعریف می‌شوند و دستورات داخل آکولادها به عنوان یک واحد اجرایی در نظر گرفته می‌شوند. زمانی که یک دستور شرطی یا یک حلقه تعریف می‌شود، دستورات متعلق به آن دستور داخل آکولادها قرار می‌گیرند.

مثال ۱: دستور شرطی (if-else)



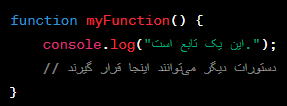
در این مثال، دستورات `console.log` داخل بلوک `if` و `else` قرار دارند.

مثال ۲: حلقه (for)



در این مثال، دستور `console.log` داخل بلوک حلقه `for` قرار دارد. بلوک اجرایی حلقه با `{}` شروع و پایان می‌یابد.

مثال ۳: تعریف تابع



در تعریف یک تابع نیز، دستورات تابع داخل یک بلوک آکولاد قرار دارند.

استفاده از بلوک‌ها باعث بهبود خوانایی کد، حفظ ساختار و قابلیت اطمینان در اجرای بخش‌های مختلف کد می‌شود. آکولادها به تعداد یکتا دستورات مربوطه را مشخص می‌کنند و از اشکالات جلوگیری می‌کنند.

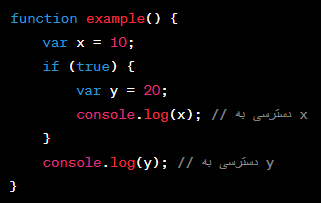
# **12-3 ایا کلمات کلیدی ویژه ای برای اعمال تغییر در حوزۀ تعریف متغیرها در جاوااسریپت وجود دارند؟**

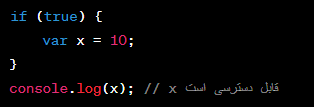
در جاوااسکریپت، متغیرها می‌توانند با استفاده از سه کلمه کلیدی `var`، `let` و `const` تعریف شوند. این کلمات کلیدی تفاوت‌هایی در تعریف متغیرها دارند:

1. var:

`var` یک متغیر را تعریف می‌کند و از آن به عنوان متغیر تابعی استفاده می‌کند (function-scoped).

ممکن است به هر نقطه‌ای در داخل یک تابع دسترسی داشته باشد، حتی اگر داخل بلوک شرطی یا حلقه باشد.

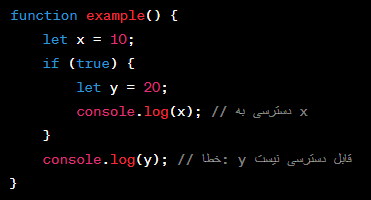


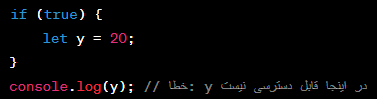


2. let:

`let` متغیرها را به عنوان متغیرهای بلوکی تعریف می‌کند (block-scoped).

این به این معناست که دسترسی به `let` در محدوده تعریف شده (بلوک، شرط، حلقه) محدود است.

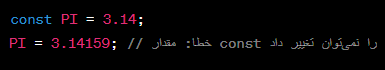




3. const:

`const` نیز متغیرها را به عنوان متغیرهای بلوکی تعریف می‌کند.

اما با این تفاوت که یکبار مقدار دهی شده‌اند، نمی‌توان آن‌ها را مجدداً تغییر داد.

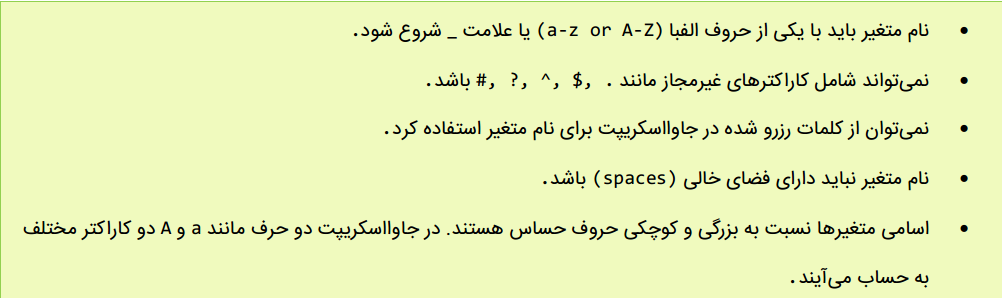


به کمک این کلمات کلیدی، توسعه‌دهندگان در جاوااسکریپت می‌توانند متغیرهای خود را با توجه به نیازهای مختلف و محدودیت‌ها تعریف کنند.

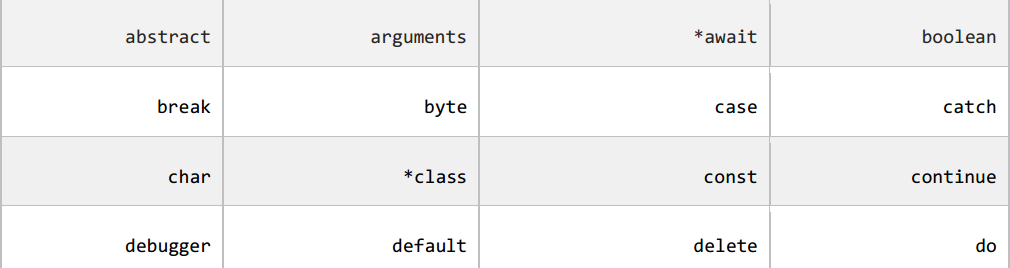
**متغیر**

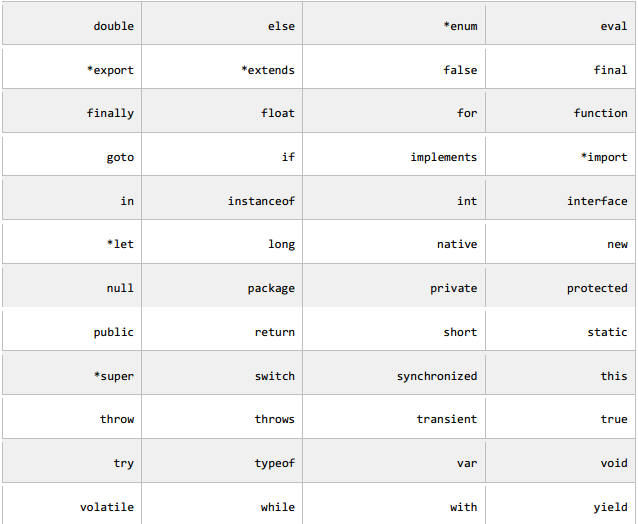
متغیر مکانی از حافظه است که شما میتوانید مقادیری را در آن ذخیره کنید. میتوان آن را به عنوان یک ظرف تصور کرد که دادههای خود را در آن قرار دادهاید. محتویات این ظرف میتواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن میتوان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار میباشد که میتواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر میتواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع دادهای که در آن ذخیره میشود یکی است.

متغیر دارای عمر نیز هست که از روی آن میتوان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما میگوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده میکنیم. هنگامی که یک برنامه ایجاد میکنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا دادههایی که توسط کاربر وارد میشوند، داریم. این مکان، همان متغیر است.



دو متغیر با نامهای my Number و My Number دو متغیر مختلف محسوب میشوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع میشود. متغیر دارای نوع هست که نوع دادهای را که در خود ذخیره میکند را نشان میدهد. در مورد انواع داده ها در جاوااسکریپت توضیح می دهیم. لیست کلمات کلیدی جاوااسکریپت، که نباید از آنها در نامگذاری متغیرها استفاده کرد در زیر آمده است:





متغیرهای جاوا اسکریپت را می توان به 4 روش اعلام کرد:

بطور خودکار

با استفاده از var

با استفاده از let

با استفاده از const

کلمه کلیدی var در تمام کدهای جاوا اسکریپت از سال 1995 تا 2015 استفاده شده است.کلمات کلیدی let و const در سال 2015 به جاوا اسکریپت اضافه شدند.کلمه کلیدی var فقط باید در کد نوشته شده برای مرورگرهای قدیمی استفاده شود.

چه زمانی از var، let یا const استفاده کنیم؟

1. همیشه متغیرها را اعلام کنید

2. اگر مقدار نباید تغییر کند، همیشه از const استفاده کنید

3. اگر نوع نباید تغییر کند (آرایه ها و اشیا) همیشه از const استفاده کنید.

4. فقط اگر نمی توانید از const استفاده کنید از let استفاده کنید

5. فقط در صورتی از var استفاده کنید که باید از مرورگرهای قدیمی پشتیبانی کنید.

همه متغیرهای جاوا اسکریپت باید با نام های منحصر به فرد شناسایی شوند.این اسامی منحصر به فرد را شناسه می نامند.شناسه ها می توانند نام های کوتاه (مانند x و y) یا نام های توصیفی تر (سن، مجموع، حجم کل) باشند.قوانین کلی برای ساخت نام برای متغیرها (شناسه های منحصر به فرد) عبارتند از:

نام ها می توانند شامل حروف، اعداد، زیرخط و علامت دلار باشند.

نام ها باید با یک حرف شروع شوند.

نام ها همچنین می توانند با $ و \_ شروع شوند (اما در این آموزش از آن استفاده نمی کنیم).

نام ها به حروف بزرگ و کوچک حساس هستند و متغیرهای متفاوتی هستند.

کلمات رزرو شده (مانند کلمات کلیدی جاوا اسکریپت) را نمی توان به عنوان نام استفاده کرد.

شناسه های جاوا اسکریپت به حروف بزرگ و کوچک حساس هستند. در جاوا اسکریپت، نوع‌های داده متغیرها به طور پویا تعیین می‌شوند، یعنی یک متغیر می‌تواند در طول اجرای برنامه از یک نوع به نوع دیگر تغییر کند. این ویژگی اغلب به عنوان "تایپ پویا" یا "dynamic typing" شناخته می‌شود.

# **13-3 انواع داده جاوا اسکریپت**

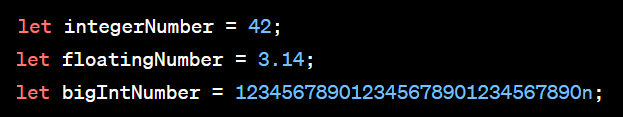
جاوا اسکریپت دارای 10 نوع داده است:

1. String

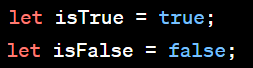


2. Number: اعداد صحیح و اعشاری

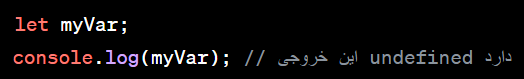
3. Bigint: برای اعداد صحیح بزرگ



4. Boolean



5. Undefined: نیز یک مقدار خاص در جاوا اسکریپت است که نشان‌دهنده این است که یک متغیر تعریف شده است، اما به آن هیچ مقداری اختصاص نیافته است یا یک ویژگی موجود در یک شیء تعیین نشده است. در موارد بسیاری، undefined به عنوان مقدار پیش‌فرض برای متغیرها استفاده می‌شود، اگر مقداری به آن اختصاص داده نشده باشد. این همچنین ممکن است در مواردی که یک تابع یک مقدار را بازگردانده نشود، به عنوان مقدار بازگردانده‌شده توسط تابع نیز دیده شود. در کل، **undefined** به نوعی نشان‌دهنده‌ی عدم تعریف یا تخصیص مقدار به یک متغیر یا ویژگی است.Top of Form



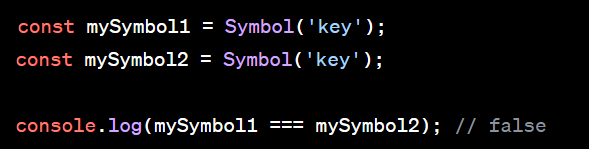
6. Null: در جاوا اسکریپت یک مقدار خاص است که به عنوان نشانه‌ای برای نشان دادن عدم وجود یک مقدار یا مقدار خالی استفاده می‌شود. در واقع، **null** نشان دهنده‌ی این است که یک متغیر یا یک ویژگی از یک شیء به صورت هندسی مقداردهی نشده است.



7. Symbol: نیز یک نوع داده در جاوا اسکریپت است. این نوع داده برای ایجاد یک مقدار یکتا (unique) و غیرقابل تغییر (immutable) به منظور استفاده در عناوین ویژگی‌های شیء یا ایجاد کلیدهای منحصر به فرد در مجموعه‌ها (Sets) استفاده می‌شود.



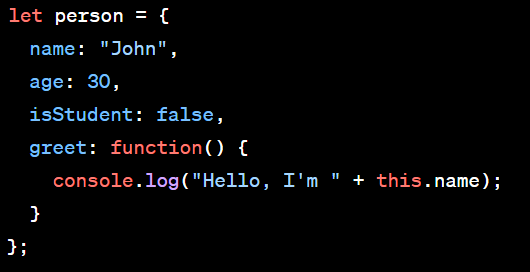
Symbol به دلیل یکتایی و غیرقابل تغییر بودن، می‌تواند به خوبی برای جلوگیری از تداخل نام در ویژگی‌ها یا کلیدهای مجموعه‌ها مورد استفاده قرار گیرد.



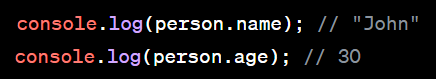
در مثال بالا، هر **Symbol** به عنوان یک مقدار منحصر به فرد در نظر گرفته می‌شود.

Top of Form

8. Object: اشیاء یا شیء (Objects) در جاوا اسکریپت یک نوع داده است که به شما این امکان را می‌دهد که یک مجموعه از ویژگی‌ها یا ویژگی و مقدارهای مرتبط را در یک واحد ذخیره کنید. این به شما این امکان را می‌دهد که اطلاعات را به صورت ساختاری و منظم نگه دارید.



در این مثال، **person** یک شیء است که ویژگی‌های **name**، **age**، و **isStudent** دارد. همچنین یک تابع به نام **greet** نیز به عنوان یکی از ویژگی‌ها در آن وجود دارد. می‌توانیم به ویژگی‌های یک شیء به صورت مستقیم دسترسی پیدا کنید:

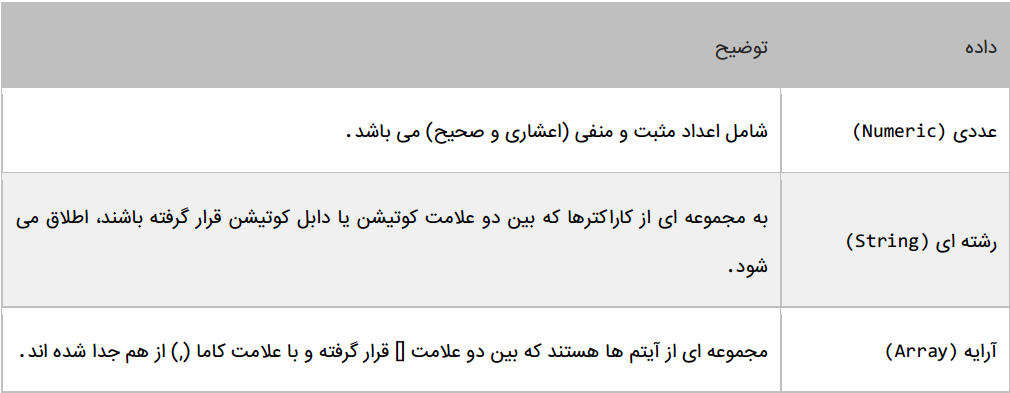


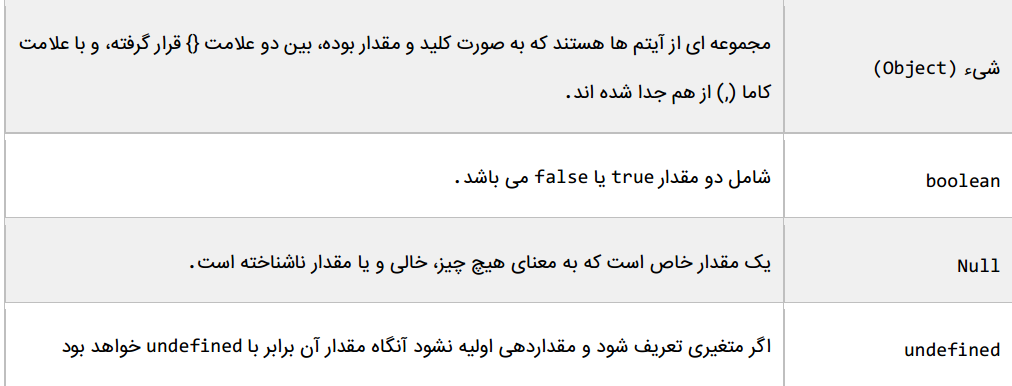
همچنین می‌توانید توابع را از طریق شیء صدا بزنید:



9.Array: یک مجموعه از عناصر.





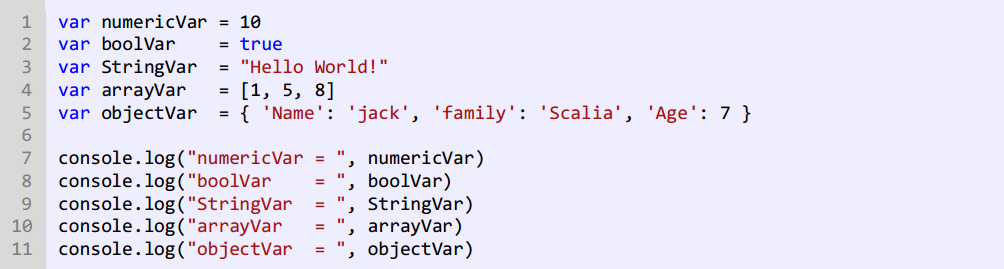


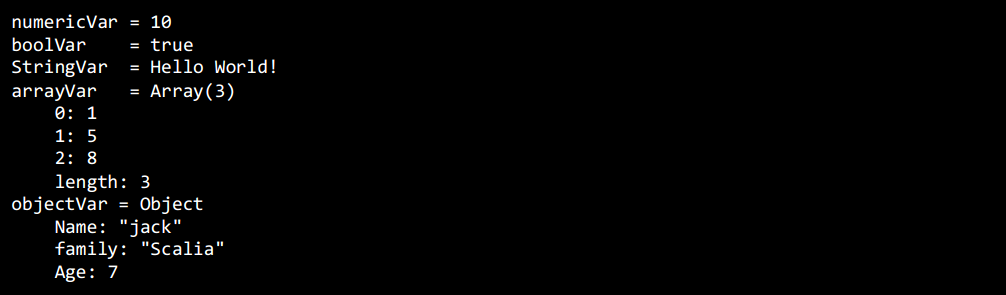
## **3-13-1 استفاده از متغیرها**

بر خلاف زبان هایی مثل جاوا و سی شارپ، که هنگام تعریف متغیر باید نوع متغیر را هم مشخص می کردیم، در JavaScript کافیست که ابتدا کلمه کلیدی var و سپس نام متغیر را نوشته و به وسیله علامت مساوی یک مقدار به آن اختصاص دهیم :

var variableName = Value

در مثال زیر نحوه تعریف و مقداردهی متغیرها نمایش داده شده است :





در خطوط 5-1 ،متغیر ها تعریف شده اند. اما نوع این متغیرها چیست؟ JavaScript نوع متغیرها را بسته به مقداری که به آنها اختصاص داده می شود در نظر می گیرد. مثال نوع متغیر StringVar در خط 3 از نوع رشته است، چون یک مقدار رشته ای به آن اختصاص داده شده است. به خطوط 4 و 5 کد بالا توجه کنید. در خط 4 یک متغیر تعریف شده است و نوع داده ای که به آن اختصاص داده شده است از نوع array است. همانطور که در درس قبل اشاره شد، برای تعریف array علامت [] به کار می رود و آیتم های داخل آن به وسیله کاما از هم جدا می شوند :

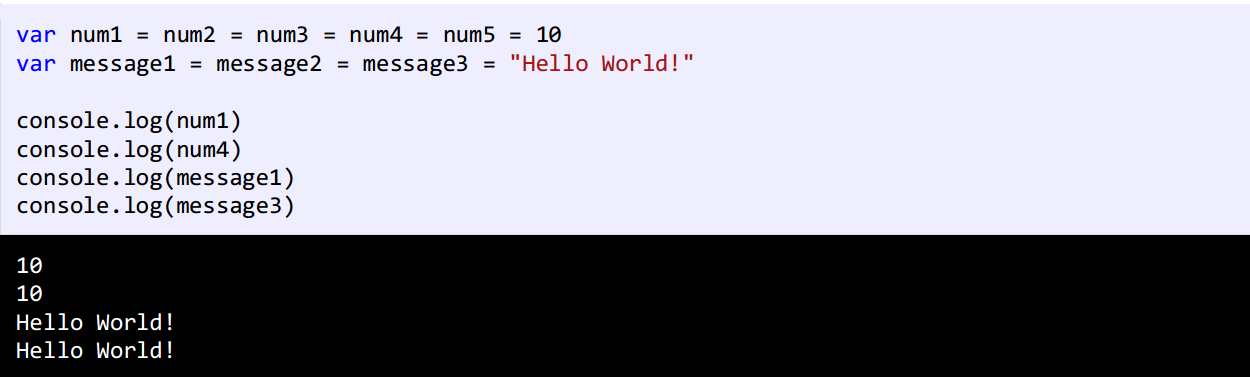
var arrayVar = [1,5,8]

در خط 5 هم یک متغیر تعریف شده است و یک مقدار از نوع Object به آن اختصاص داده شده است. در تعریف Object به جای علامت [] از {} استفاده می شود. آیتم ها به صورت کلید/مقدار تعریف می شوند. بین کلید و مقدارعلامت: و بین هر دو کلید/مقدار، هم علامت , قرار می گیرد.

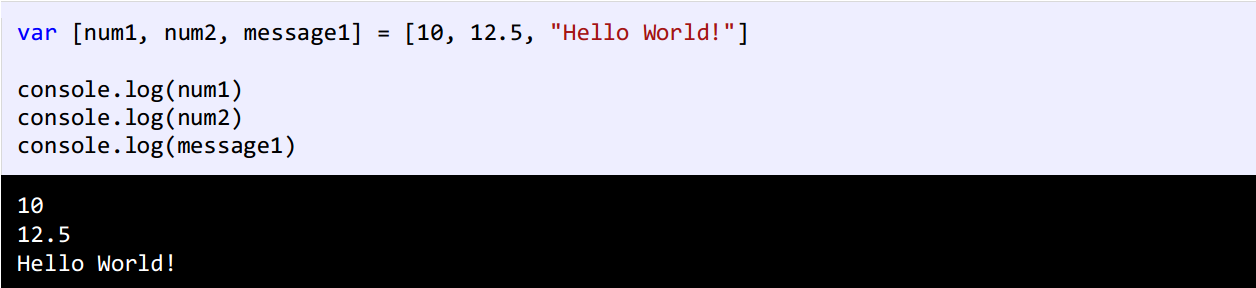
var object Var = {Key1: Value1, Key2: Value2, Key3: Value3}

مثلا در مثال بالا یک Object تعریف کرده ایم که سه آیتم یا کلید/مقدار دارد که بین آنها عالمت کاما ),( قرار داده ایم. ولی بین یک کلید و مقدار مربوط به آن علامت : قرار گرفته است. برای اختصاص یک مقدار به چند متغیر می توان به صورت زیر عمل کرد :

var identifier1 = identifier2 = ...indentifierN = Value



دقت کنید که برای متغیرهای تعریف شده در حالت بالا یک خانه حافظه تخصیص داده می شود، یعنی مقدار 10 در حافظه ذخیره شده ومتغیرهای num1 و num2 و num3 و num4 و num5 به آن خانه از حافظه اشاره می کنند. همچنین می توان چند متغیر را تعریف کرد و برای هر یک از آن ها مقدار جداگانه ای مشخص نمود: یک متغیر جاوا اسکریپت می تواند هر نوع داده ای را در خود نگه دارد. توجه داشته باشید که جاوا اسکریپت یک زبان پویا و تایپ پویا است، بنابراین متغیرها می‌توانند در طول اجرای برنامه از یک نوع به نوع دیگر تغییر کنند.



در کد بالا مقدار num1 برابر 10 ،num2 برابر 5.12 و message1 برابر World Hello !می باشد. در JavaScript متغیر ها هم باید تعریف و هم مقداردهی شوند. یعنی اگر متغیری را تعریف کرده و به آن مقداری را اختصاص ندهید و برنامه را اجرا کنید با خطا مواجه می شوید :



یک رشته در اصل یک مجموعه از کاراکترهاست که در داخل عالمت ”“ یا ’ ’ قرار دارند. هر کدام از این کاراکترها دارای یک اندیس است که به وسیله آن اندیس قابل دسترسی هستند. اندیس کاراکترها در رشته از 0 شروع می شود.

var message = "Hello World!"

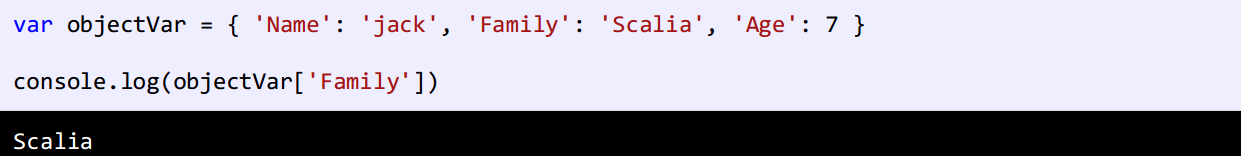
در رشته بالا اندیس کاراکتر O برابر 4 است. برای درک بهتر به شکل زیر توجه کنید :



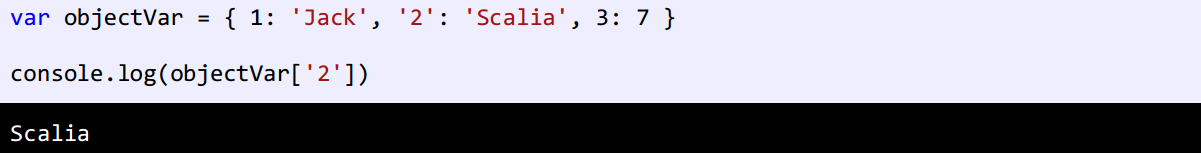
حال برای چاپ یک کاراکتر )مثال W )از این رشته کافیست که به صورت زیر عمل کنیم :



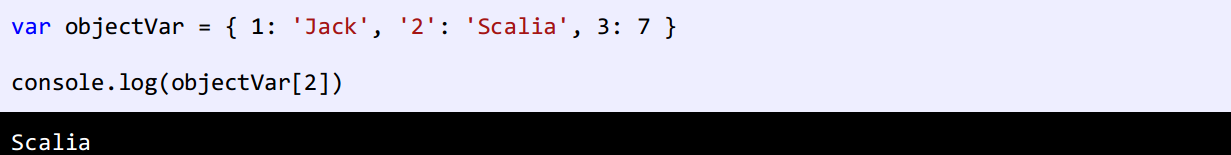
و اما در مورد Object ،شما باید نام کلید را بنویسید تا مقدار آن برای شما نمایش داده شود:



نکته ای که بهتر است در همین جا به آن اشاره کنیم این است که کلید/مقدارها در Object می توانند از هر نوعی باشند و شما برای چاپ مقدار مربوط به یک کلید باید نام کلید را دقیق بنویسید. به مثال زیر توجه کنید :



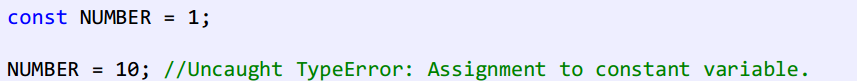
در مثال باال ما مقدار کلید ’2 ’را چاپ کرده ایم. حال اگر به جای ’2 ’عدد 2 را بنویسیم، نتیجه همان است :



## **3-13-2 ثابت ها**

هی اولیه شوند و اگر مقدار دهی آنها فراموش ثابتها، انواعی هستند که مقدار آنها در طول برنامه تغییر نمیکند. ثابتها شود در برنامه خطا به وجود میآید. بعد از این که به ثابتها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمیتوان آن را تغییر داد. برای تعریف ثابتها باید از کلمه کلیدی const استفاده کرد. معموال نام ثابتها را طبق قرارداد با حروف بزرگ مینویسند تا تشخیص آنها در برنامه راحت باشد. نحوه تعریف ثابت در زیر آمده است :

const data\_type identifier = initial\_value;

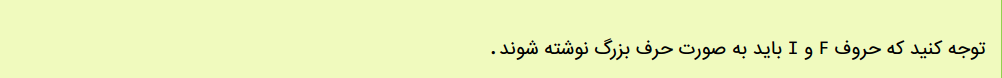


در این مثال میبینید که مقدار دادن به یک ثابت، که قبال مقدار دهی شده برنامه را با خطا مواجه میکند. ممکن است این سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابتها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمیکنند بهتر است که آنها را به صورت ثابت تعریف کنید. این کار هر چند کوچک کیفیت برنامه شما را بالا میبرد.

## **3-13-3 تبدیل انواع داده**

در زبان جاوااسکریپت امکان تبدیل یک نوع به نوع دیگر وجود دارد. این زبان دارای مجموعهای از توابع از پیش تعریف شده است، که میتوانند مقادیر را از یک نوع بـه نوع دیگر تبدیل کنند. جاوااسکریپت دو متد برای تبدیل انواع غیر عددی به عددی فراهم کرده است:

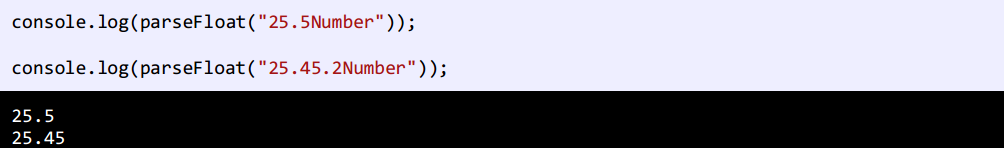
* praseInt()
* parseFloat()



این متدها فقط بر روی رشتههای حاوی عدد کار میکنند و بر روی بقیه انواع مقدار NaN را بر میگردانند. متد ()praseInt از اولین کاراکتر رشته شروع میکند اگر عدد بود آن را بر میگرداند در غیر این صورت مقدار NaN را بر میگردانند. این روند تا آخرین کاراکتر ادامه پیدا میکند تا اینکه به کاراکتری غیر عددی برسد. به مثال زیر توجه کنید:



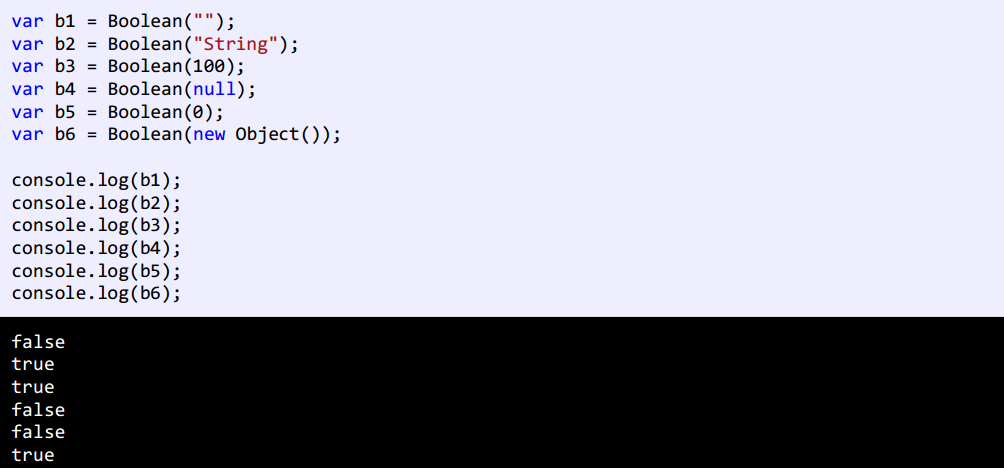
متد ()parseFloat نیز همانند ()parseInt عمل کرده و از اولین کاراکتر شروع به جستجو میکند. البته در این متد اولین کاراکتر نقطه حساب نمیشود و آن را به همان صورت میگرداند. به مثال زیر توجه کنید:



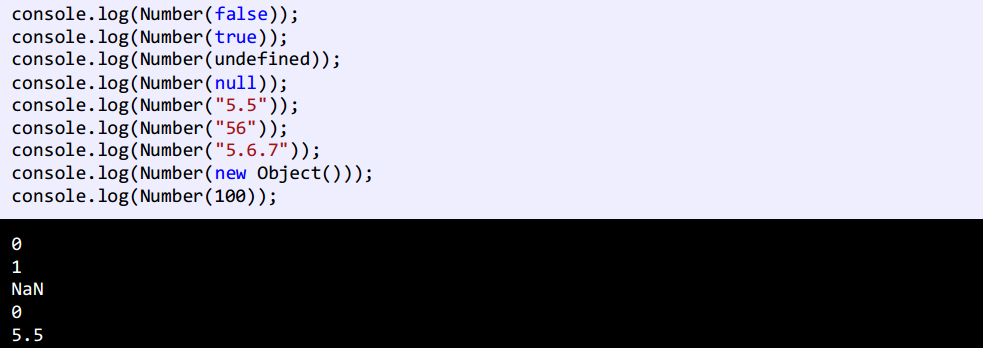
در جاوااسکریپت امکان استفاده از روشی موسوم به Casting Type برای تبدیل انواع وجود دارد. سه متد برای Casting Type وجود دارد:

* Boolean
* Number
* String

متد ()Boolean زمانی مقدار true را بر میگرداند که پارامتر دریافتیاش، رشتهای شامل حداقل یک کارکتر، یک عدد غیر از صفر و یا یک شیء باشد. مقدار false را نیز زمانی بر میگرداند که پارامتر دریافتیاش رشتهای تهی، عدد صفر یا یکی مقادیر null و undefinedباشد:



متد ()Number کاری شبیه به متدهای ()parseInt و ()parseFloat انجام میدهد ولی تفاوتهایی هم با این دو متد دارد. اگر به یاد داشته باشید متدهای ()parseInt و ()parseFloat آرگومان دریافتی را فقط تا اولین کاراکتر بی ارزش بر میگرداندند.



سادهترین متد هم ()String است که همان چیزی را که میگیرد به عنوان رشته بر میگرداند:



در جاوااسکریپت یک متد به نام ()typeof وجود دارد که از آن برای تشخیص نوع متغیر استفاده میشود. به مثال زیر توجه کنید :



## **4-13-3 ذخیره داده**

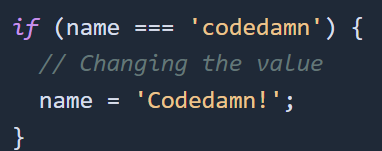
بر خلاف زبان های سطح پایین مانند C، جاوا اسکریپت به طور خودکار حافظه را در هنگام ایجاد اشیاء اختصاص می دهد و زمانی که دیگر استفاده نمی شود آن را آزاد می کند (جمع آوری زباله). C دارای تکنیک های مدیریت حافظه مانند malloc () و free () است. این بدان معنا نیست که ما نیازی به نگرانی در مورد مدیریت حافظه در جاوا اسکریپت نداریم.

چرخه عمر حافظه: صرف نظر از زبان برنامه نویسی، چرخه عمر حافظه مراحل زیر را دنبال می کند:

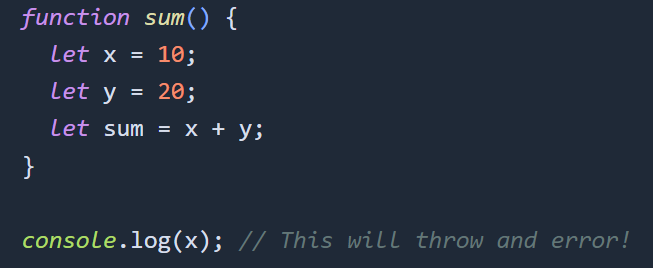
1. حافظه مورد نیاز ما را تخصیص می دهد: جاوا اسکریپت حافظه را به شی ایجاد شده اختصاص می دهد.

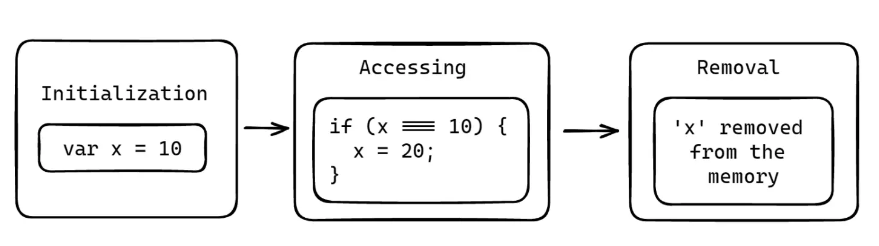


1. از حافظه اختصاص داده شده استفاده کنید.



1. زمانی که از حافظه استفاده نمی شود، حافظه را آزاد کنید: زمانی که حافظه اختصاص داده شده آزاد شد، برای مقاصد دیگر استفاده می شود. حذف، تخصیص حافظه رزرو شده توسط متغیرها است. پس از توزیع، نمی‌توانیم از کد جاوا اسکریپت خود به متغیر دسترسی پیدا کنیم. به عنوان مثال، ما نمی توانیم به متغیرهای ایجاد شده در داخل یک تابع دسترسی داشته باشیم.

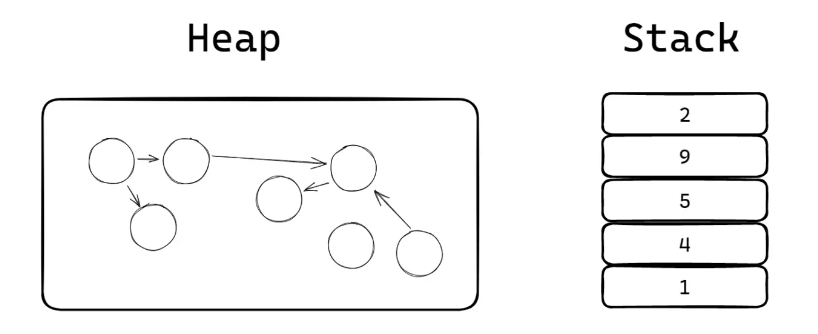




مرحله دوم برای همه زبان ها یکسان است. با این حال، مرحله اول و آخر در زبان های سطح بالا مانند جاوا اسکریپت به صورت ضمنی است. توجه: "اشیاء" در این زمینه نه تنها به معنی اشیاء در جاوا اسکریپت است، بلکه به معنی توابع و محدوده عملکرد نیز می باشد. در جاوا اسکریپت، تخصیص و مدیریت حافظه توسط موتور جاوا اسکریپت انجام می شود که بخشی از محیط زمان اجرا است که کد جاوا اسکریپت شما در آن اجرا می شود. مشخصات نحوه تخصیص انواع داده در حافظه ممکن است بین موتورهای مختلف جاوا اسکریپت متفاوت باشد. مقداردهی اولیه ایجاد متغیر است. در جاوا اسکریپت، ما این کار را با استفاده از کلمات کلیدی let، const یا var انجام می دهیم. در این مرحله، موتور جاوا اسکریپت حافظه را برای متغیرها ذخیره می کند و مقادیر را در فضای حافظه اختصاص داده شده ذخیره می کند.

# **14-3 جاوا اسکریپت چگونه متغیرها را ذخیره می کند؟**

ما در مورد تخصیص و واگذاری صحبت کردیم، اما سوال اینجاست که این تخصیص ها و واگذاری ها کجا اتفاق می افتد؟

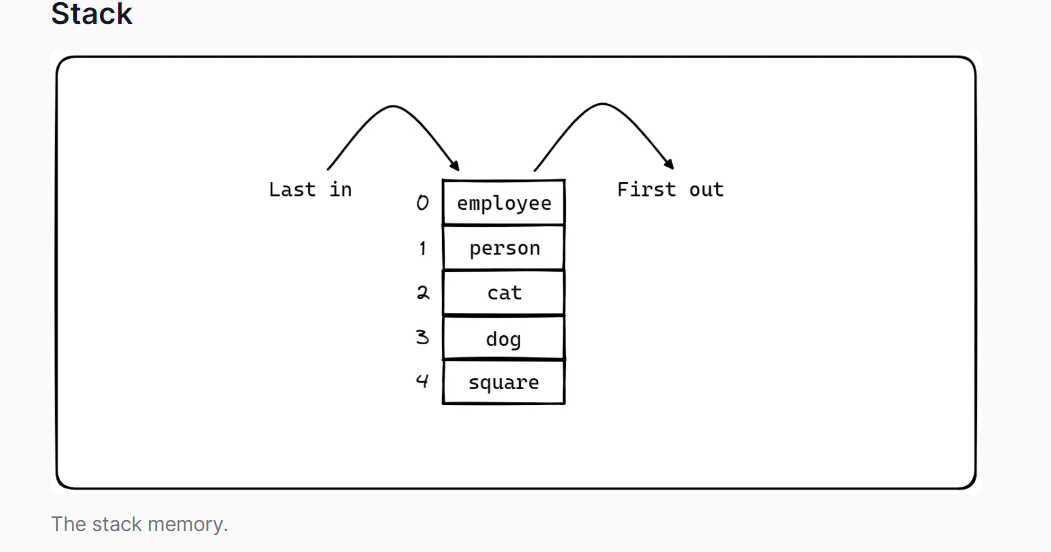


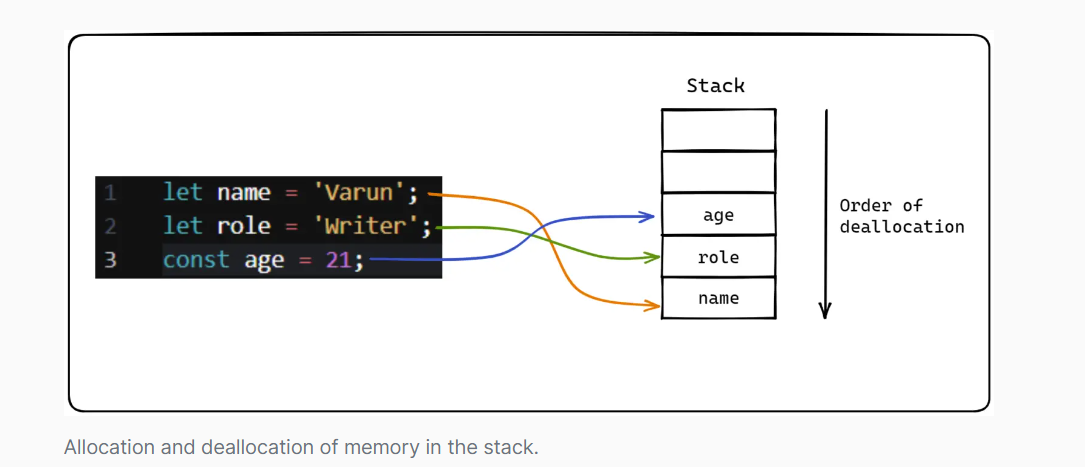
جاوا اسکریپت از دو نوع فضای حافظه یعنی Stack و Heap برای ذخیره متغیرها تشکیل شده است. تفاوت بین آنها این است که چه متغیرهایی در کدام نوع ذخیره سازی ذخیره می شوند. بیایید نگاهی دقیق تر به پشته ها و پشته ها بیندازیم.

موتورهای جاوا اسکریپت دو مکان برای ذخیره داده ها دارند:

پشته: ساختار داده ای است که برای ذخیره داده های استاتیک استفاده می شود. داده های استاتیک به داده هایی اطلاق می شود که اندازه آنها توسط موتور در طول زمان کامپایل شناخته می شود. در جاوا اسکریپت، داده های ایستا شامل مقادیر ابتدایی مانند رشته ها، اعداد، بولی، تهی و تعریف نشده است. ارجاعاتی که به اشیا و توابع اشاره می کنند نیز گنجانده شده است. مقدار ثابتی از حافظه برای داده های استاتیک اختصاص داده شده است. این فرآیند به عنوان تخصیص حافظه ثابت شناخته می شود.

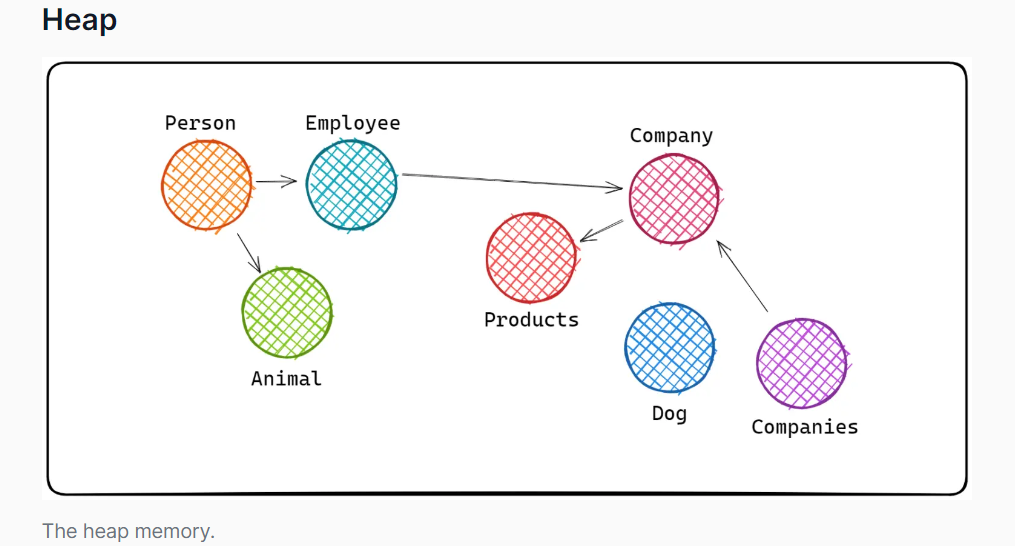
Heap: برای ذخیره اشیا و توابع در جاوا اسکریپت استفاده می شود. موتور مقدار ثابتی از حافظه را تخصیص نمی دهد. در عوض، فضای بیشتری را در صورت نیاز اختصاص می دهد.





فرض کنید چند متغیر دارید که یکی یکی به پشته اضافه می کنید. آخرین متغیری که فشار داده شده در بالا است، به این معنی که وقتی حافظه تخصیص داده می شود، اولین متغیری است که حذف می شود.

جاوا اسکریپت از ساختار داده پشته برای ذخیره داده های ثابت استفاده می کند. این شامل تمام اعداد، رشته ها، بولی ها و سایر انواع داده های اولیه می شود. این نوع داده ها دارای اندازه ثابتی هستند که در زمان کامپایل شناخته می شوند. متغیرهایی مانند اشیا، آرایه ها و غیره در پشته ذخیره نمی شوند زیرا اندازه آنها در طول زمان اجرا متفاوت است.

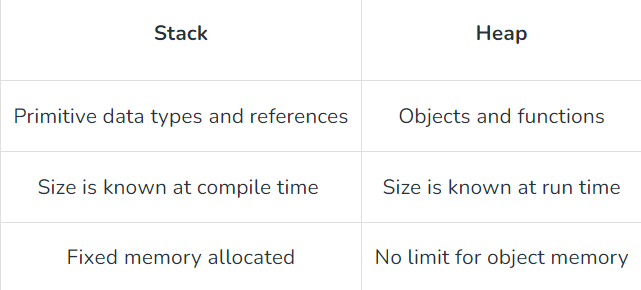


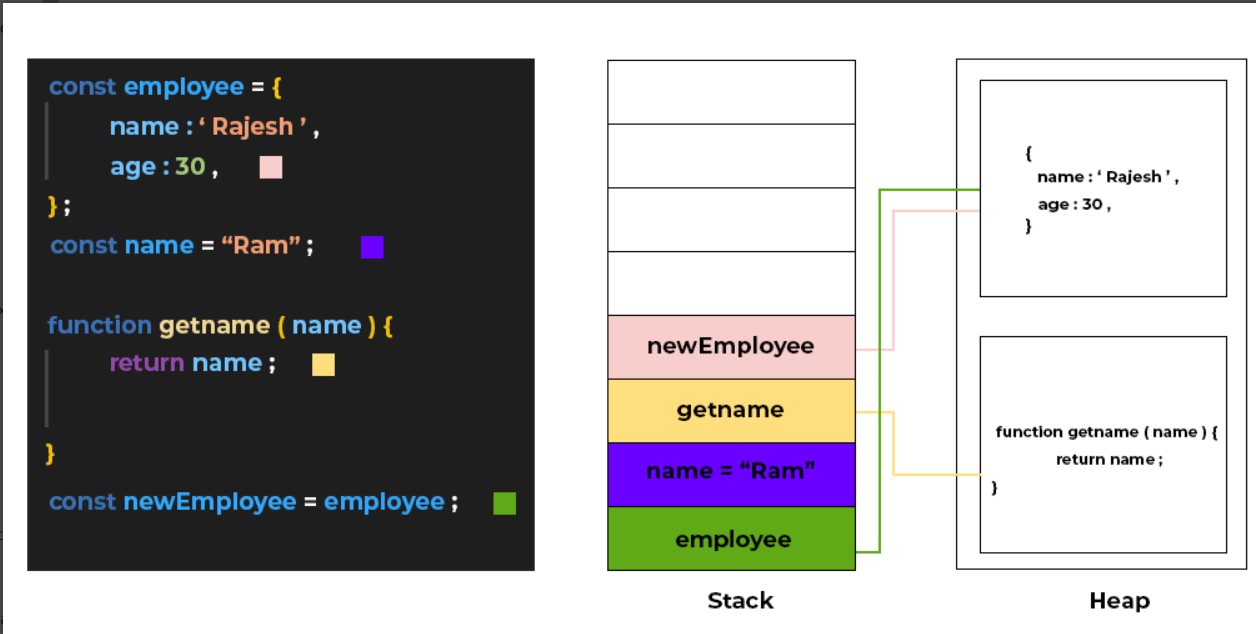
از آنجایی که Stack فقط می تواند داده های ثابت یا ثابت را ذخیره کند، جاوا اسکریپت داده های پویا را در کجا ذخیره می کند؟ پاسخ Heap است. حافظه پشته، بر خلاف حافظه پشته، محدودیتی در اندازه ثابت ندارد، یعنی حافظه به صورت پویا تخصیص می یابد.

جاوا اسکریپت از یک پشته برای ذخیره متغیرهایی استفاده می کند که اندازه آنها در زمان کامپایل ناشناخته است یا ممکن است در زمان اجرا متفاوت باشد، مانند اشیا، آرایه ها، توابع و غیره.

موتور جاوا اسکریپت به صورت پویا حافظه را به پشته اختصاص می دهد. در ابتدا، اندازه پشته به حافظه سیستم در دسترس بستگی دارد و می توان آن را به صورت پویا بر اساس نیاز افزایش/کاهش داد.

* **Primitives (Numbers, Strings, Booleans, Undefined, Null):** اولیه ها مستقیماً در محل حافظه ای که متغیر به آن دسترسی دارد ذخیره می شود. آنها معمولاً انواع کوچک و ساده ای هستند که می توانند مستقیماً در مقدار ثابتی از حافظه ذخیره شوند.به عنوان مثال، یک عدد یا یک بولی در مقدار ثابتی از حافظه متناسب با اندازه آن نوع داده ذخیره می شود.
* **Objects (Objects, Arrays, Functions):** اشیاء ساختارهای داده پیچیده تری هستند و معمولاً به عنوان مرجع در حافظه ذخیره می شوند. داده های واقعی برای یک شی در جای دیگری در حافظه ذخیره می شود و متغیر یک مرجع (آدرس حافظه) را به آن مکان نگه می دارد.این مرجع به اشیا اجازه می دهد تا از نظر اندازه و محتوا انعطاف پذیرتر باشند.
* **Symbols:** نمادها منحصر به فرد و تغییر ناپذیر هستند و به روشی مشابه نمادهای اولیه ذخیره می شوند. موتور جاوا اسکریپت تضمین می کند که هر مقدار نماد منحصر به فرد است، بنابراین می توان آن را مستقیماً در حافظه ذخیره کرد.
* **Functions:** توابع اشیایی در جاوا اسکریپت هستند و مشابه اشیاء ذخیره می شوند.کد تابع در حافظه ذخیره می شود و متغیر مرجعی به آن کد دارد.





# **15-3 عبارات و عملگرها**

ابتدا با دو کلمه آشنا شوید :

* عملگر: نمادهایی هستند که اعمال خاص انجام میدهند.
* عملوند: مقادیری که عملگرها بر روی آنها عملی انجام میدهند.

مثال Y+X :یک عبارت است که در آن X و Y عملوند و عالمت + عملگر به حساب میآیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب میآیند. جاوااسکریپت دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسهای، منطقی و بیتی میباشد. از عملگرهای ساده ریاضی میتوان به عملگر جمع و تفریق اشاره کرد. سه نوع عملگر در جاوااسکریپت وجود دارد :

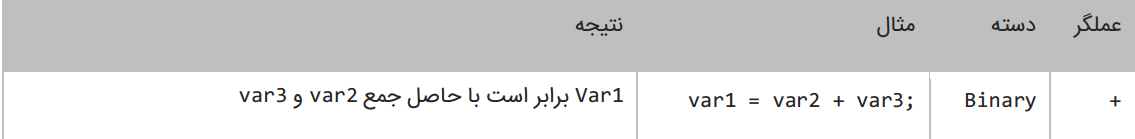
* یگانی – (Unary) به یک عملوند نیاز دارد
* دودویی – (Binary)به دو عملوند نیاز دارد
* سه تایی (Ternary) (به سه عملوند نیاز دارد

انواع مختلف عملگر که در این بخش مورد بحث قرار میگیرند، عبارتند از :

* عملگرهای ریاضی
* عملگرهای تخصیصی
* عملگرهای مقایسهای
* عملگرهای منطقی
* عملگرهای بیتی

## **1-15-3 عملگرهای ریاضی**

این زبان از عملگرهای ریاضی برای انجام محاسبات استفاده می کند. جدول زیر عملگرهای ریاضی JavaScript را نشان می دهد





دیگر عملگرهای JavaScript عملگرهای کاهش و افزایش هستند. این عملگرها مقدار 1 را از متغیرها کم یا به آنها اضافه می کنند. ازاین متغیرها اغلب در حلقه ها استفاده می شود :



به این نکته توجه داشته باشید که محل قرارگیری عملگر در نتیجه محاسبات تاثیر دارد.اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 اتفاق می افتد.چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می شود و سپس متغیر var2 افزایش یا کاهش می یابد. به مثال های زیر توجه کنید :







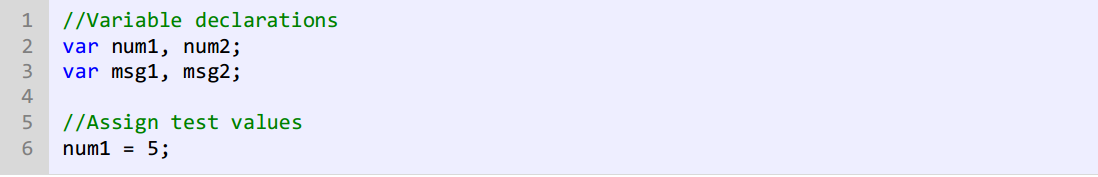
همانطور که در دو مثال باال مشاهده می کنید،درج عملگرهای -- و ++ قبل از عملوند y باعث می شود که ابتدا یک واحد از y کم و یا یک واحد به y اضافه شود و سپس نتیجه در عملوند x قرار بگیرد. حال به دو مثال زیر توجه کنید :

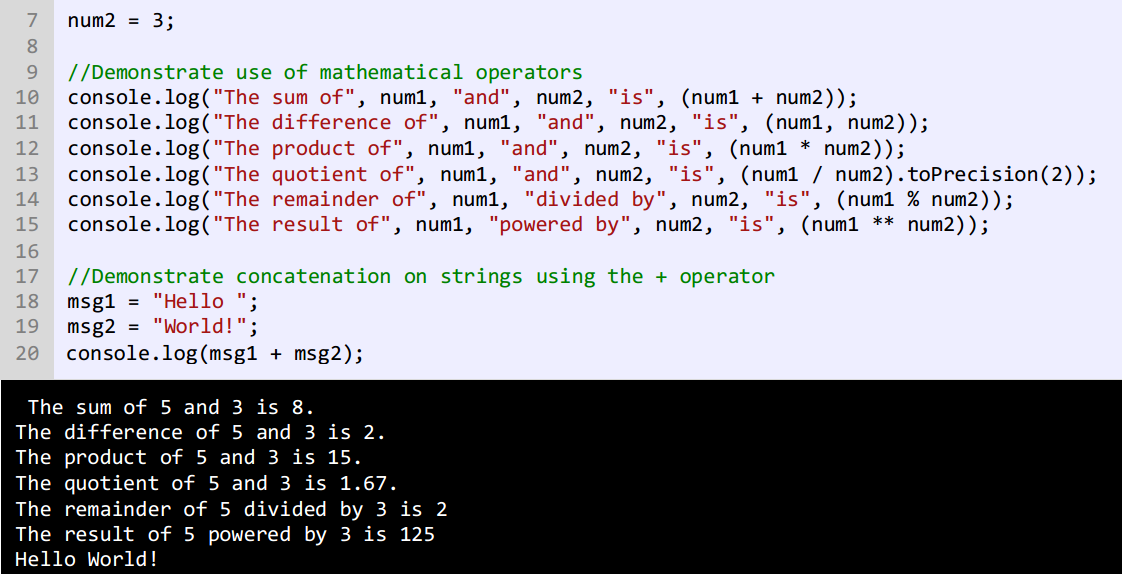




همانطور که در دو مثال باال مشاهده می کنید،درج عملگرهای — و ++ بعد از عملوند y باعث می شود که ابتدا مقدار y در داخل متغیر xقرار بگیرد و سپس یک واحد از y کم و یا یک واحد به آن اضافه شود. حال میتوانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی

در جاوااسکریپت را یاد بگیریم :



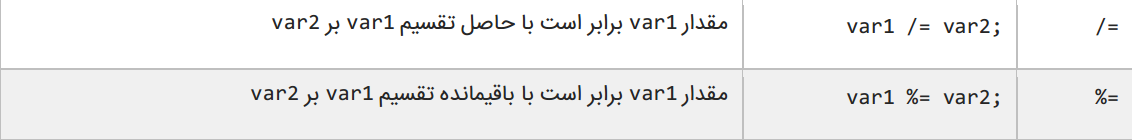


برنامه باال نتیجه هر عبارت را نشان میدهد. در این مثال و در خط 13 برای اینکه ارقام کسری بعد از عدد حاصل، دو رقم باشند، از متد toPrecision استفاده کرده ایم. عدد 2 در این جا بدین معناست که عدد را تا دو رقم اعشار نمایش بده. در خط 20 مشاهده میکنید که دو رشته به وسیله عملگر + به هم متصل شدهاند. نتیجه استفاده از عملگر + برای چسباندن دو کلمه ” Hello “و “World “!رشته “World Hello “!خواهد بود. به فاصلههای خالی بعد از اولین کلمه توجه کنید اگر آنها را حذف کنید از خروجی برنامه نیز حذف میشوند.

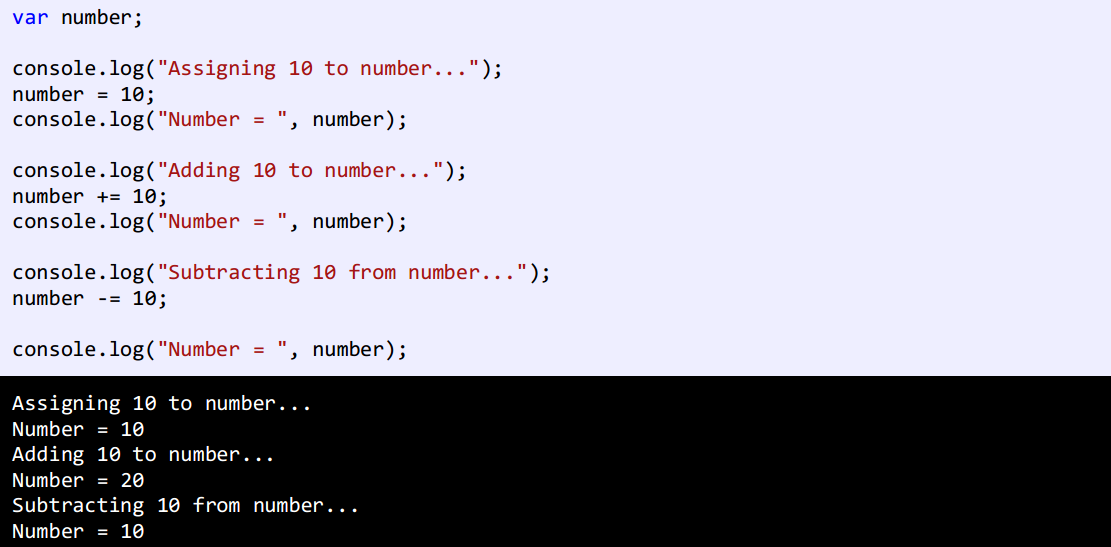
## **2-15-3 عملگرهای تخصیصی ( جایگزینی)**

نوع دیگر از عملگرهای Javascript عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار میدهند. جدول زیر انواع عملگرهای تخصیصی در Javascript را نشان میدهد:





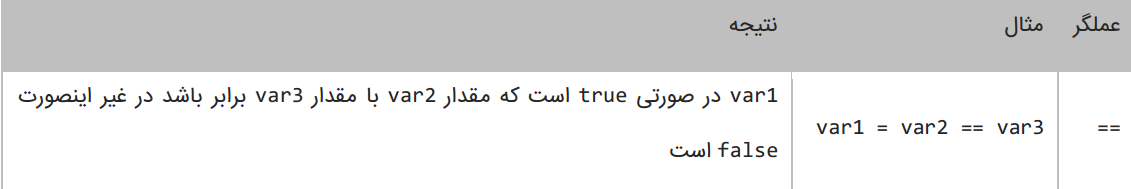
از عملگر =+ برای اتصال دو رشته نیز میتوان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خالصه نویسی در کد است. شکل اصلی کد var2 += var1 به صورت var2 + var1 = var1 میباشد. این حالت کدنویسی زمانی کارایی خود را نشان میدهد که نام متغیرها طوالنی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان میدهد:



در برنامه از 3 عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار 10 با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار 10 اضافه شده است. و در آخر به وسیله عملگر =- عدد 10 از آن کم شده است.

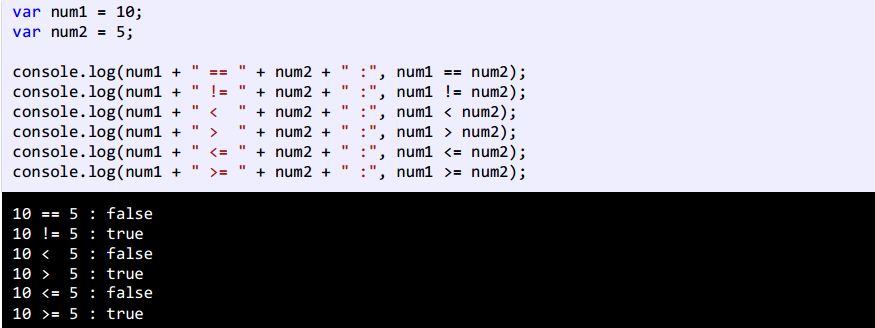
## **3-15-3 عملگرهای مقایسه ای**

از عملگرهای مقایسهای برای مقایسه مقادیر استفاده میشود. نتیجه این مقادیر یک مقدار بولی )منطقی( است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار trueو اگر نتیجه مقایسه اشتباه باشد مقدار false را نشان میدهند. این عملگرها به طورمعمول در دستورات شرطی به کار میروند به این ترتیب که باعث ادامه یا توقف دستور شرطی میشوند. جدول زیر عملگرهای مقایسهای در Javascript را نشان میدهد:





برنامه زیر نحوه عملکرد این عملگرها را نشان میدهد :



در مثال باال ابتدا دو متغیر را که میخواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص میدهیم. سپس با استفاده ازیک عملگر مقایسهای آنها را با هم مقایسه کرده و نتیجه را چاپ میکنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر ==به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند y = x مقدار y را در به x اختصاص میدهد. عملگر== عملگر مقایسهای است که دو مقدار را با هم مقایسه میکند مانند y==x و اینطور خوانده میشود x برابر است با .y

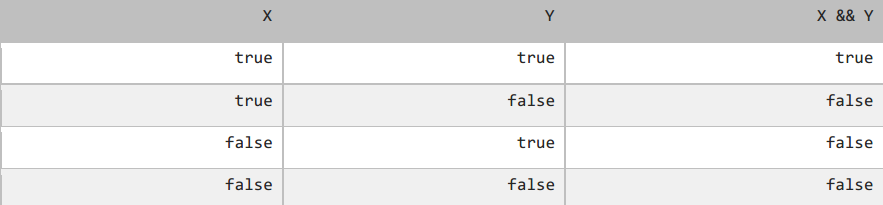
## **4-15-3 عملگرهای منطقی**

عملگرهای منطقی بر روی عبارات منطقی عمل می کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرط های پیچیده استفاده می شود. همانطور که قبال یاد گرفتید مقادیر بولی می توانند false یا true باشند. فرض کنید که var2 و var3 دو مقدار بولی هستند.



## **5-15-3 عملگر منطقی (&&)And**

اگر مقادیر دو طرف عملگر AND ،true باشد، عملگر AND مقدار true را بر می گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها false باشند مقدار false را بر می گرداند. در زیر جدول درستی عملگر AND نشان داده شده است :



برای درک بهتر تاثیر عملگر AND یاد آوری می کنم که این عملگر فقط در صورتی مقدار true را نشان می دهد که هر دو عملوند مقدارشان true باشد. در غیر اینصورت نتیجه تمام ترکیب های بعدی false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه ای است. به عنوان مثال نتیجه عبارت زیر درست (true (است اگر سن (age (بزرگتر از 18 و salary کوچکتر از 1000 باشد.

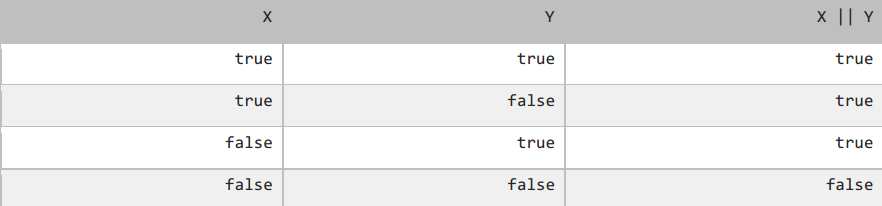


عملگر AND زمانی کارامد است که ما با محدود خاصی از اعداد سر و کار داریم. مثال عبارت 100 <= x <= 10 بدین معنی است که x میتواند مقداری شامل اعداد 10 تا 100 را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می توان از عملگر منطقی AND به صورت زیر استفاده کرد.



## **6-15-3 عملگر منطقی (||)or**

اگر یکی یا هر دو مقدار دو طرف عملگر OR ،درست (true (باشد، عملگر OR مقدار true را بر می گرداند. جدول درستی عملگر OR در زیرنشان داده شده است :

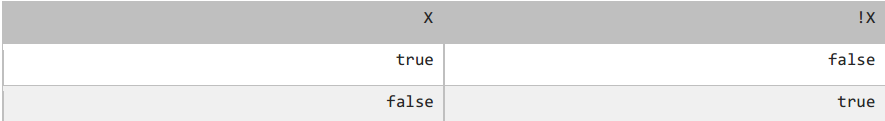


در جدول باال مشاهده می کنید که عملگر OR در صورتی مقدار false را بر می گرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (true (است که رتبه نهایی دانش آموز (finalGrade)بزرگتر از 75 یا یا نمره نهایی امتحان آن 100 باشد.



## **7-15-3 عملگر منطقی (!)Not**

برخالف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا اصطالح بولی را نفی می کند. مثال اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می کند. جدول زیر عملکرد اپراتور NOT را نشان می دهد :

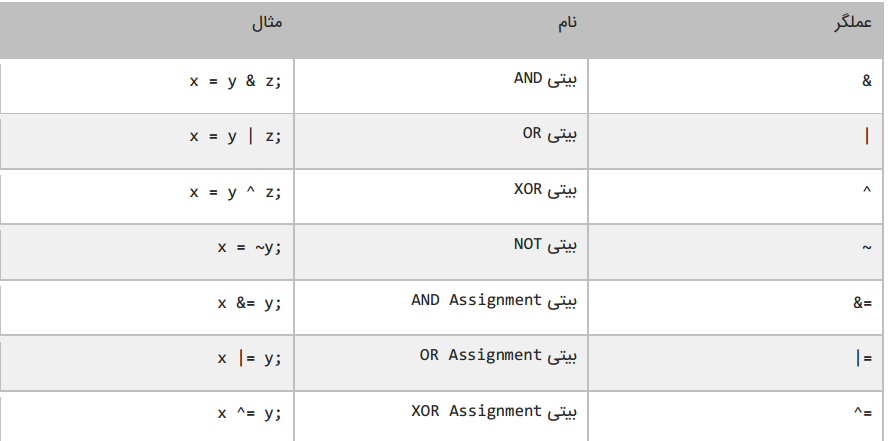


## **8-15-3 عملگرهای بیتی**

عملگرهای بیتی به شما اجازه میدهند که شکل باینری انواع دادهها را دستکاری کنید. در سیستم باینری )دودویی( که کامپیوتر از آن استفاده میکند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد 1 و برای نشان دادن حالت خاموش از عدد 0 استفاده میشود. بنابراین اعداد باینری فقط میتوانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای 2 و اعداد اعشاری را اعداد در مبنای 10 می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده 8 بیت است. به عنوان مثال برای یک داده از نوع int به 32 بیت یا 4 بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از 32 رقم 0 و 1 برای ذخیره استفاده میکنند. برای مثال عدد 100 وقتی به عنوان یک متغیر از نوع int ذخیره میشود در کامپیوتر به صورت زیر خوانده میشود:

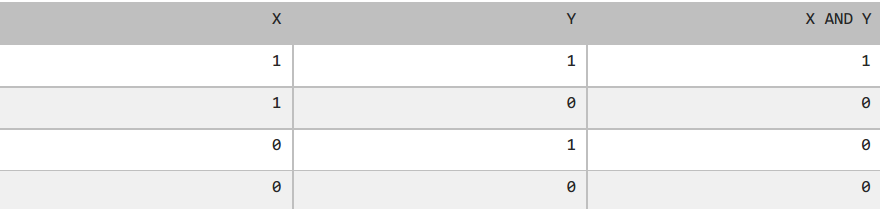


عدد 100 در مبنای ده معادل عدد 1100100 در مبنای 2 است. در اینجا 7 رقم سمت راست نشان دهنده عدد 100 در مبنای 2 است و مابقی صفرهای سمت چپ برای پر کردن بیت هایی است که عدد از نوع int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده میشوند. عملگرهای بیتی Javascript در جدول زیر نشان داده شده اند :



## **9-15-3 عملگر بیتی (&)And**

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام میدهد با این تفاوت که این عملگر بر روی بیتها کار میکند. اگر مقادیر دو طرف آن 1 باشد مقدار 1 را بر میگرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر میگرداند. جدول درستی عمگر بیتی AND در زیرآمده است:



در زیر نحوه استفاده از عملگر بیتی AND آمده است :



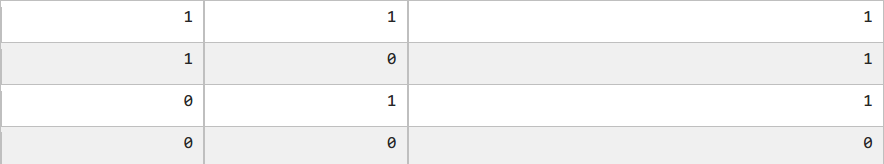
همانطور که در مثال باال مشاهده میکنید نتیجه عملکرد عملگر AND بر روی دو مقدار 5 و 3 عدد یک میشود. اجازه بدهید ببینیم که چطور این نتیجه را به دست میآید:



ابتدا دو عدد 5 و 3 به معادل باینریشان تبدیل میشوند. از آنجاییکه هر عدد صحیح 32) int (بیت است از صفر برای پر کردن بیتهای خالی استفاده میکنیم. با استفاده ازجدول درستی عملگر بیتی AND میتوان فهمید که چرا نتیجه عدد یک میشود.

## **10-15-3 عملگر بیتی (|)Or**

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه صفر در غیر اینصورت 1 خواهد شد. جدول درستی این عملگر در زیر آمده است :



نتیجه عملگر بیتی OR در صورتی صفر است که عملوندهای دو طرف آن صفر باشند. اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید :

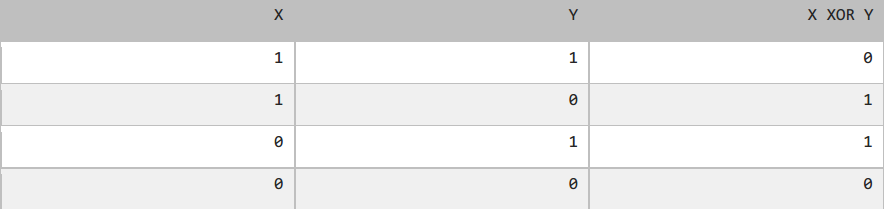


وقتی که از عملگر بیتی OR برای دو مقدار در مثال باال )7 و 9 )استفاده میکنیم نتیجه 15 میشود. حال بررسی میکنیم که چرا این نتیجه به دست آمده است؟



## **11-15-3 عملگر بیتی (^)XOR**

جدول درستی این عملگر در زیر آمده است :



در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک میشود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده میکنید :

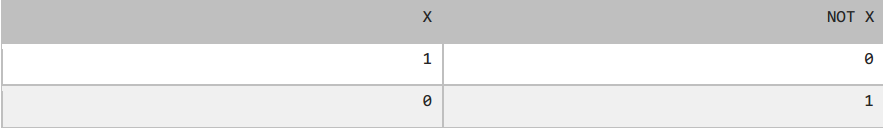






## **12-15-3 عملگر بیتی (~)NOT**

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. در زیر جدول درستی این عملگر آمده است:



## **13-15-3 عملگر بیتی تغییر مکان (shift)**

این نوع عملگرها به شما اجازه میدهند که بیتها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول میکنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت ها را نشان میدهد:





مشاهده میکنید که همه بیتها به اندازه دو واحد به سمت چپ منتقل شده اند. در این انتقال دو صفر از صفرهای سمت چپ کم میشود و در عوض دو صفر به سمت راست اضافه میشود.





هر بیت به اندازه 4 واحد به سمت راست منتقل میشود، بنابراین 4 بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه میشود.

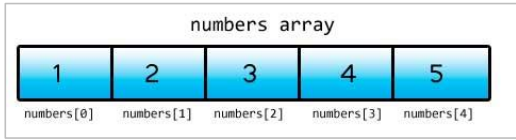
# **16-3 کار با آرایه ها**

آرایه نوعی متغیر است که لیستی از آدرسهای مجموعه ای از داده های همنوع یا غیر همنوع را در خود ذخیره می کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثال اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئنا تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می توان همه آنها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است :

آرایه‌ها در جاوا اسکریپت یکی از مهمترین ساختارهای داده هستند. یک آرایه می‌تواند مجموعه‌ای از عناصر را در خود نگه دارد. بیا نگاهی به چند مفهوم اصلی در کار با آرایه‌ها در جاوا اسکریپت بیندازیم:

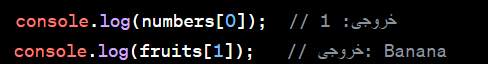
تعریف در آرایه: برای تعریف یک آرایه در جاوا اسکریپت از نماد **[ ]** استفاده می‌شود. مثلا:  


اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می شود.به عنوان مثال شما یک آرایه 5 عضوی دارید، اندیس آرایه از 0 تا 4 می باشد چون طول آرایه 5 است پس 1-5 برابر است با 4 .این بدان معناست که اندیس 0 نشان دهنده اولین عضو آرایه است و اندیس 1 نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید:



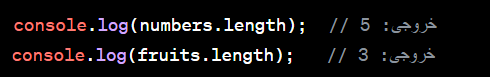
## **3-16-1 دسترسی به عناصر:**

عناصر آرایه با استفاده از ایندکس (شماره) دسترسی پیدا می‌کنند. در جاوا اسکریپت، ایندکس‌ها از صفر شروع می‌شوند. برای مثال:



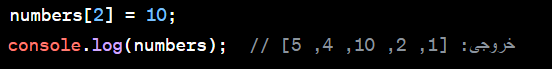
## **3-16-2 طول آرایه:**

طول یک آرایه با استفاده از ویژگی length قابل دریافت است**:**

****

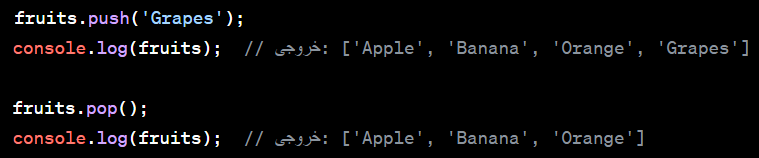
## **3-16-3 تغییر مقدار یک عنصر:**

می‌توانید مقدار یک عنصر را با استفاده از ایندکس تغییر دهید:

****

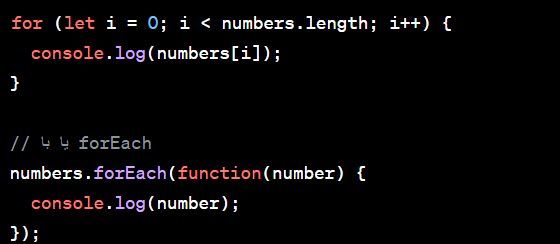
## **4-16-3 افزودن و حذف عناصر:**

برای افزودن عناصر به آرایه از push استفاده می‌شود و برای حذف از pop. مثال:

****

## **5-16-3 حلقه‌ها و آرایه‌ها:**

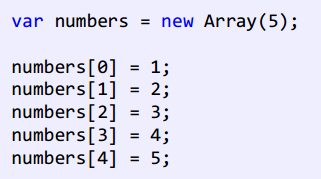
می‌توانید با استفاده از حلقه‌ها (مثل for یا forEach) از طول آرایه برای گردش و دسترسی به هر عنصر استفاده کنید:



یک روش دیگر برای تعریف آرایه استفاده از کلمه کلیدی new به صورت زیر است:



آرایه بالا را به صورت زیر هم می توان تعریف کرد:

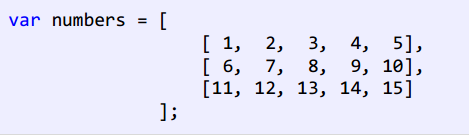
****

## **6-16-3 آرایه چند بعدی**

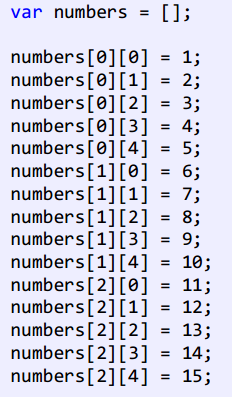
آرایههای چند بعدی آرایههایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را میتوان مانند یک جدول با تعدای ستون و ردیف تصور کنید. با افزایش اندیسها اندازه ابعاد آرایه نیز افزایش مییابد و آرایههای چند بعدی با بیش از دو اندیس به وجود میآیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است :

****

می توان گفت که یک آرایه دو بعدی، خود آرایه ای از آرایه هاست. یعنی هر عنصر این نوع آرایه، خود یک آرایه است. آرایه دو بعدی رو می توان به صورت یک جدول تصور کرد که دارای سطر و ستون می باشد. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم، یکی اندیس سطر و دیگری اندیس ستونی که آن عنصر در آن قرار دارد. یک مثال از آرایه دو بعدی در زیر آمده است :



در کد باال یک آرایه به نام numbers تعریف شده است که دارای سه عنصر است. البته هر کدام از این عناصر خود یک آرایه می باشند. یعنی آرایه numbers آرایه است از 3 آرایه. هر کدام از این آرایه ها هم 5 عنصر دارند. پس می توان گفت که آرایه numbers در واقع یک جدول با 3 سطر و 5 ستون می باشد. میتوان مقدار دهی به عناصر را به صورت دستی انجام داد مانند :



# **17-3 لیست ها**

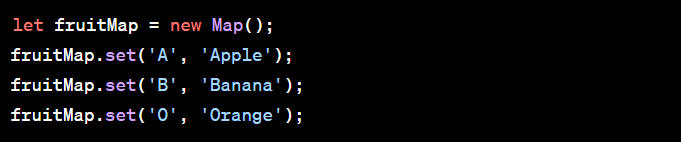
لیست‌ها در جاوا اسکریپت به واقعیت با آرایه‌ها تطابق دارند. در واقع، تا ES6 (ECMAScript 2015)، جاوا اسکریپت لیست‌ها به نام "آرایه" دارد. با اضافه شدن ویژگی‌های جدید به جاوا اسکریپت، مانند Map و Set، امکان استفاده از ساختارهای داده متنوع‌تری نیز وجود دارد.

آرایه‌ها (Arrays):

همانطور که قبلاً گفتیم، آرایه‌ها در واقع لیست‌ها هستند:



Map: یک ساختار داده‌ی کلید-مقدار است که می‌تواند به عنوان یک نوع لیست به حساب بیاید:

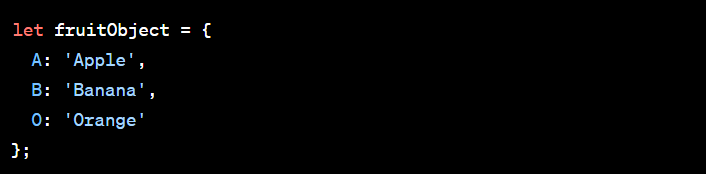


Set:یک مجموعه‌ی منحصربه‌فرد از مقادیر بدون تکرار است، معمولاً برای نگه‌داری لیست‌های متفاوت:



Objects:

اگرچه شئ‌ها در جاوا اسکریپت اصطلاحاً "لیست" نیستند، اما می‌توانند یک مجموعه از ویژگی‌ها (کلید-مقدار) را نگه دارند:



NodeList: وقتی با DOM (Document Object Model) در جاوا اسکریپت کار می‌کنید، گاهی اوقات یک NodeList را به عنوان لیستی از المان‌ها می‌بینید:



# **18-3 رشته ها**

کار با رشته‌ها (String) در جاوا اسکریپت یکی از پایه‌ای‌ترین وظایف است. رشته‌ها در جاوا اسکریپت به صورت اشیاء (objects) نمایان می‌شوند، که معنایی آن است که شما می‌توانید به ویژگی‌ها و توابع مرتبط با رشته دسترسی داشته باشید.

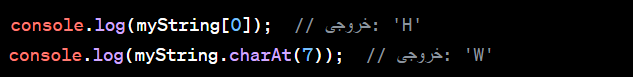
## **1-18-3 تعریف رشته**



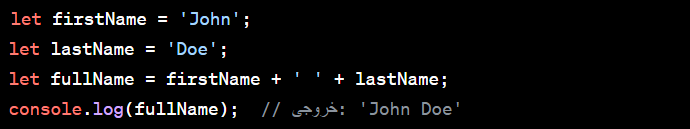
## **2-18-3 طول رشته**

****

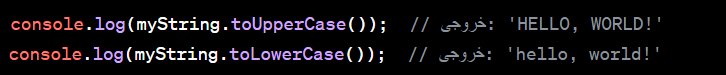
## **3-18-3 دسترسی به کاراکترها**

****

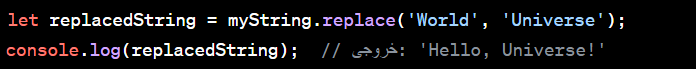
## **4-18-3 اتصال رشته‌ها (Concatenation)**



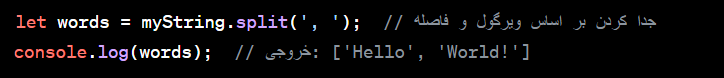
## **5-18-3 تغییر حالت حروف**



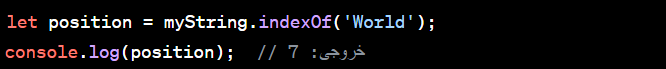
## **6-18-3 جایگزینی زیررشته**



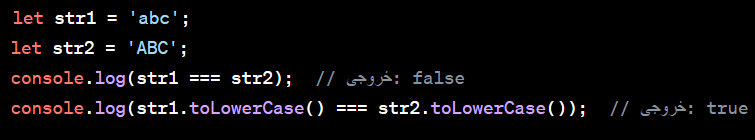
## **7-18-3 جدا کردن رشته**



## **8-18-3 جستجو در رشته**



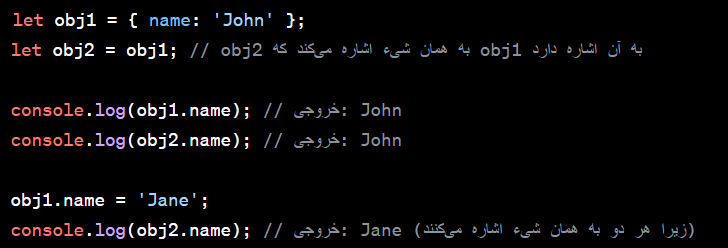
## **9-18-3 مقایسه رشته‌ها**



# **19-3 اشاره گر‌ها و متغیر های مرجع**

در جاوا اسکریپت، اشاره‌گرها به صورت مستقیم پشتیبانی نمی‌شوند، به این معنا که شما نمی‌توانید به طور مستقیم به یک مکان حافظه اشاره کنید یا از اشاره‌گرها استفاده کنید مانند زبان‌هایی مانند C یا C++.

اما، با وجود عدم پشتیبانی از اشاره‌گرها، شما همچنان با اشیاء (objects) و متغیرهای مرجع کار می‌کنید. هنگامی که یک متغیر به یک شیء یا آرایه اشاره می‌کند، شما در واقع یک نوع از "مرجع" را دارید.



در این مثال، هرچند ما اشاره‌گرها را به صورت مستقیم نداریم، اما obj1 و obj2 هر دو به یک شیء اشاره می‌کنند، بنابراین تغییراتی که در یکی از آنها انجام می‌شود، بر روی دیگری نیز تأثیر می‌گذارد.

در جاوا‌اسکریپت، مشکلات نشتی حافظه به صورت مستقیم قابل رفع نیستند زیرا زبان‌هایی مانند C یا C++ که اشاره‌گرهای دستی را مدیریت مستقیم حافظه ارائه می‌دهند، در جاوا‌اسکریپت وجود ندارند.

اما برخی از راهکارها و سازوکارهایی که می‌توانند کمک کننده باشند:

1. **استفاده از Garbage Collectionجمع‌آوری زباله:** جاوا‌اسکریپت از یک مکانیزم جمع‌آوری زباله برخوردار است که به طور خودکار اشیاء غیرقابل دسترس را تشخیص داده و از حافظه حذف می‌کند. این به طور کلی مشکلات نشتی حافظه را کاهش می‌دهد.
2. **رفع مشکلات در کد:** برنامه نویسی بهینه و پیشگیری از خطاهای مرتبط با مدیریت حافظه می‌تواند کمک کننده باشد. استفاده از متغیرهای محلی به جای متغیرهای سراسری، آزمون دقیق کد، و استفاده از الگوهای بهتر برنامه‌نویسی از جمله راهکارهایی هستند.
3. **استفاده از ابزارهای نظارتی (Monitoring Tools):** استفاده از ابزارهایی که مانیتورینگ برنامه و مصرف حافظه را فراهم می‌کنند، می‌تواند کمک کند. این ابزارها می‌توانند نقاط ضعف و نشتی حافظه را شناسایی کنند.
4. **استفاده از Strict Mode:** استفاده از Strict Mode در جاوا‌اسکریپت ممکن است باعث شناسایی و اعلام خطاهای احتمالی در کد شود که می‌تواند کمک کند تا مشکلات نشتی حافظه به سرعت شناسایی و رفع شوند.
5. **استفاده از اصول برنامه‌نویسی خوب:** اجتناب از انجام عملیات‌های پیچیده و غیرضروری، استفاده از متغیرهای محلی به جای متغیرهای سراسری، و به کارگیری اصول برنامه‌نویسی خوب می‌تواند کمک کند.

همچنین، بسیاری از مشکلات نشتی حافظه در برنامه‌های جاوا‌اسکریپت که در محیط مرورگر اجرا می‌شوند، به نحو معقولی توسط مرورگرها مدیریت می‌شوند.

Top of Form

# **20-3 آیا بازیافت کننده حافظه وجود دارد و در صورت وجود چگونه پیاده سازی شده است.**

بله، بازیافت کننده حافظه (Memory Allocator) عموماً یک جزء از سیستم مدیریت حافظه است که مسئول تخصیص و آزادسازی حافظه است. این مسئولیت شامل تخصیص بخش‌هایی از حافظه به برنامه‌ها در زمان اجرا (تخصیص دینامیک حافظه) و آزادسازی حافظه که دیگر نیازی به آن نیست، می‌شود.

پیاده‌سازی بازیافت کننده حافظه در زبان‌ها و محیط‌های مختلف می‌تواند متفاوت باشد، اما در اصول، یک بازیافت کننده حافظه عموماً دارای دو عملکرد اصلی است:

تخصیص حافظه (Allocation):

در زمان اجرا، بازیافت کننده حافظه مسئول تخصیص بخش‌هایی از حافظه به برنامه است. این تخصیص ممکن است به صورت استاتیک (مثلاً در زمان کامپایل) یا دینامیک (در زمان اجرا) انجام شود.

آزادسازی حافظه (Deallocation):

زمانی که یک برنامه دیگر نیاز به یک بخش از حافظه ندارد، بازیافت کننده حافظه مسئول آزادسازی (رها کردن) آن بخش از حافظه است. این مرحله مهم است تا حافظه نه‌تنها تخصیص دهی شود بلکه همچنین قابلیت آزادسازی داشته باشد.

در زبان‌هایی مثل C و C++، بازیافت کننده حافظه به صورت مستقیم قابل دسترسی است و برنامه‌نویس مسئولیت تخصیص و آزادسازی حافظه را دارد. اما در زبان‌هایی مثل جاوا یا C#, این فرآیند به صورت خودکار توسط مکانیزم‌های جمع‌آوری زباله (Garbage Collection) انجام می‌شود.

بله، در جاوا اسکریپت نیز مکانیزم جمع‌آوری زباله (Garbage Collection) به عنوان بازیافت‌کننده حافظه وجود دارد. این مکانیزم به صورت خودکار اشیاء غیرقابل دسترس را شناسایی و حذف می‌کند. در جاوا اسکریپت، مرورگرها مسئول اجرای کد هستند و مکانیزم جمع‌آوری زباله در مرورگرها پیاده‌سازی می‌شود.

پیاده‌سازی جمع‌آوری زباله در مرورگرهای معروف مانند Chrome، Firefox، و Safari از الگوریتم‌های مختلفی استفاده می‌کند. به عنوان مثال:

1. **V8 (Chrome):** مرورگر Chrome از موتور اجرای V8 استفاده می‌کند که یک مکانیزم جمع‌آوری زباله پیشرفته دارد. V8 از الگوریتم Generational Incremental Garbage Collection برای بهبود عملکرد استفاده می‌کند.
2. **Spider Monkey (Firefox):** مرورگر Firefox از موتور اجرای Spider Monkey استفاده می‌کند که نیز از الگوریتم‌های جمع‌آوری زباله Generational GC برای بهینه‌سازی مصرف حافظه استفاده می‌کند.
3. **JavaScript Core (Safari):** مرورگر Safari از موتور اجرای JavaScript Core یا Nitro) استفاده می‌کند که از الگوریتم‌های جمع‌آوری زباله به نام Write Barrier و GC و Thread-Assisted GC استفاده می‌کند.

پیاده‌سازی دقیق الگوریتم‌ها و رفتار جمع‌آوری زباله ممکن است بر اساس نسخه مرورگر و تغییرات در مدل‌های جدید مرورگرها متغیر باشد.

Top of Form