

به نام خدا

# مجموعه سوال و جواب‌های جاواسکریپت

عیسی رضائی





Mariotek platform

[Mariotek.ir](http://Mariotek.ir)

# مجموعه سوال و جواب‌های جاواسکریپتی

اگه از کتاب خوشتون اومد به گیت‌هابمون مراجعه کنین و بهمون ★ بدین. اگر هم قصد مشارکت داشتید همونجا می‌تونین شروع کنین و ما هم خیلی خوشحال می‌شیم 😊

<https://github.com/mariotek>

لینک گیت‌هاب ما برای مشارکت برای تولید کتاب‌ها:

## نحوه دانلود کتاب به فرمتهای PDF/Epub

می‌توانید خیلی راحت نسخه آنلاین کتاب استفاده کنید یا اگه به فایل کتاب می‌خواهید دسترسی داشته باشید، از بخش ریلیزهای گیت‌هاب به فرمتهای مختلف آخرین نسخه کتاب رو می‌توانید دریافت کنید.

## فهرست

صفحه	سؤال	ردیف
	روش‌های ایجاد object‌ها توى جاوا‌سکریپت کدوما هستن؟	۱
	زنجیره prototype چیه؟	۲
	تفاوت‌های بین bind، apply و call کدوما هستن؟	۳
	فرمت JSON چیه و عملیات‌های معمول روی اون چیا هستن؟	۴
	هدف از متدهای slice روى آرایه‌ها چیه؟	۵
	هدف از متدهای splice روى آرایه‌ها چیه؟	۶
	تفاوت متدهای slice و splice چیا هستن؟	۷
	تفاوت‌های Object و Map چیا هستن؟	۸
	تفاوت‌های بین عملگرهای == و === چیا هستن؟	۹
	تابع arrow-function یا lambda چی هستن؟	۱۰
	یه تابع first-class چجور تابعیه؟	۱۱
	یه تابع first-order چجور تابعیه؟	۱۲
	یه تابع higher-order چجور تابعیه؟	۱۳
	یه تابع unary چجور تابعیه؟	۱۴
	تابع currying توابع یعنی چی؟	۱۵
	چه تابعی pure هستن؟	۱۶

صفحه	سوال	ردیف
	هدف از کلمه کلیدی let چیه؟	۱۷
	تفاوت‌های کلمات کلیدی let و var چیا هستن؟	۱۸
	دلیل انتخاب کلمه کلیدی let چیه؟	۱۹
	چطوری می‌تونیم توی بلوک مربوط به switch بدون دریافت خطا متغیر تعريف کنیم؟	۲۰
	Temporal-Dead-Zone چیه؟	۲۱
	(توابع بلا فاصله صدا زده شده) IIFE چی هستن؟	۲۲
	مزایای استفاده از module چیه؟	۲۳
	Memoization چیه؟	۲۴
	Hoisting چیه؟	۲۵
	Class ها توی ES6 چیکار می‌کنن؟	۲۶
	Closure ها چیا هستن؟	۲۷
	Module ها چیا هستن؟	۲۸
	چرا به module ها نیاز داریم؟	۲۹
	توی جاوا‌سکریپت scope چیه و چیکار می‌کننه؟	۳۰
	service-worker چیه؟	۳۱
	توی service-worker چطوری می‌شه DOM رو دستکاری کرد؟	۳۲
	چطوری می‌تونیم بین ریست شدن‌های service-worker داده‌های مورد نظرمون رو مجدد استفاده کنیم؟	۳۳
	IndexedDB چیه؟	۳۴
	Web-storage چیه؟	۳۵
	Post-message چیه؟	۳۶

صفحه	سوال	ردیف
	چیه؟ Cookie	۳۷
	چرا به cookie نیاز داریم؟	۳۸
	گزینه‌های قابل تنظیم توی cookie چیا هستن؟	۳۹
	چطوری می‌شه یه cookie رو حذف کرد؟	۴۰
	تفاوت‌های بین session-storage و local-storage و cookie چیا هستن؟	۴۱
	تفاوت‌های بین sessionStorage و localStorage چیا هستن؟	۴۲
	چطوری به web-storage دسترسی پیدا می‌کنی؟	۴۳
	چه متدهایی روی session-storage قابل استفاده هستن؟	۴۴
	رخداد storage چیه و چطوری ازش استفاده می‌کنیم؟	۴۵
	چرا به web-storage نیاز داریم؟	۴۶
	چطوری می‌تونیم پشتیبانی از web-storage توسط مرورگر رو بررسی کنیم؟	۴۷
	چطوری می‌تونیم پشتیبانی از web-worker توسط مرورگر رو بررسی کنیم؟	۴۸
	یه مثال از web-workerها می‌تونی بزنی؟	۴۹
	محدودیت‌های web-workerها روی DOM چیا هستن؟	۵۰
	چیه؟ Promise	۵۱
	چرا به Promise نیاز داریم؟	۵۲
	سه تا وضعیت ممکن برای یه Promise چیا هستن؟	۵۳
	توابع callback چی هستن؟	۵۴
	چرا به توابع callback نیاز داریم؟	۵۵
	یا جهنم توابع callback یا Callback-hell چیه؟	۵۶
	یا همون SSE یا Server-sent-events چیه؟	۵۷

صفحه	سوال	ردیف
	چطوری می‌تونیم پیام‌های server-sent-event را دریافت کنیم؟	۵۸
	چطوری می‌تونیم پشتیبانی مرورگر برای SSE را بررسی کنیم؟	۵۹
	کدام توابع روی SSE وجود دارند؟	۶۰
	اصلی‌ترین قوانین Promise‌ها چیا هستن؟	۶۱
	توی callback چطوری رخ میده؟	۶۲
	زنجیره Promise‌ها چیه؟	۶۳
	کاربرد متدهای promise.all چیه؟	۶۴
	هدف از متدهای race چیه؟	۶۵
	حالت strict توی جاواسکریپت چی کار میکنه؟	۶۶
	چرا به حالت strict نیاز داریم؟	۶۷
	چطوری می‌توانیم حالت strict را فعال کنیم؟	۶۸
	هدف از عملگر نقطی دوتابعی (!! چیه؟	۶۹
	هدف از عملگر delete چیه؟	۷۰
	عملگر typeof چیکار میکنه؟	۷۱
	عملگر undefined چیه و چه زمانی می‌گیریم؟	۷۲
	null چیه؟	۷۳
	تفاوت‌های بین null و undefined چیا هستن؟	۷۴
	متدهای eval چیه؟	۷۵
	تفاوت‌های بین window و document چیا هستن؟	۷۶
	توی جاواسکریپت چطوری می‌توانیم به history دسترسی داشته باشیم؟	۷۷
	انواع داده‌های جاواسکریپت کدوما هستن؟	۷۸

صفحه	سوال	ردیف
	isNaN چیه و چیکار میکنه؟	۷۹
	تفاوت‌های بین undefined و undeclared چیا هستن؟	۸۰
	کدوم متغیرها عمومی هستن؟	۸۱
	مشکلات متغیرهای عمومی چیا هستن؟	۸۲
	مقدار NaN چیه؟	۸۳
	هدف از تابع isFinite چیه؟	۸۴
	یه event-flow چیه؟	۸۵
	Event-bubbling چیه؟	۸۶
	Event-capturing چیه؟	۸۷
	چطوری می‌شه یه فرم رو با استفاده از جاواسکریپت ثبت کرد؟	۸۸
	چطوری می‌شه به اطلاعات مربوط به سیستم عامل کاربر دسترسی داشت؟	۸۹
	تفاوت‌های بین رخدادهای DOMContentLoaded و document-load چیا هستن؟	۹۰
	تفاوت‌های بین object‌های host ، native و user چیا هستن؟	۹۱
	کدوم ابزار و تکنیک‌ها برای دیباگ کردن برنامه جاواسکریپتی استفاده میشن؟	۹۲
	مزایا و معایب استفاده از Promise‌ها به جای callback چیا هستن؟	۹۳
	تفاوت‌های بین attribute و property روی DOM چیا هستن؟	۹۴
	سیاست same-origin چیه؟	۹۵
	هدف استفاده از void چیه؟	۹۶
	جاواسکریپت یه زبان تفسیری هست یا کامپایلری؟	۹۷

صفحه	سوال	ردیف
	آیا جاواسکریپت یه زبان حساس به بزرگی و کوچکی(case-sensitive) حروف است؟	۹۸
	ارتباطی بین JavaScript و Java وجود دارد؟	۹۹
	Event‌ها چی هستن؟	۱۰۰
	کی جاواسکریپت رو ساخته؟	۱۰۱
	هدف از متد preventDefault چیه؟	۱۰۲
	کاربرد متد stopPropagation چیه؟	۱۰۳
	مراحلی که موقع استفاده از event-handler توی یه return false میده چیا هستن؟	۱۰۴
	BOM چیه؟	۱۰۵
	موارد استفاده از setTimeout کدوما هستن؟	۱۰۶
	موارد استفاده از setInterval کدوما هستن؟	۱۰۷
	چرا جاواسکریپت رو به عنوان یه زبان تک thread می‌شناسن؟	۱۰۸
	Event-delegation چیه؟	۱۰۹
	ECMAScript چیه؟	۱۱۰
	JSON چیه؟	۱۱۱
	قوانين فرمات JSON کدوما هستن؟	۱۱۲
	هدف از متد JSON.stringify چیه؟	۱۱۳
	چطوری می‌تونیم یه رشته JSON رو تجزیه کنیم؟	۱۱۴
	چرا به JSON نیاز داریم؟	۱۱۵
	PWA‌ها چی هستن؟	۱۱۶
	هدف از متد clearTimeout چیه؟	۱۱۷

صفحه	سوال	ردیف
	هدف از متدهای clearInterval چیه؟	۱۱۸
	توی جاواسکریپت، چطوری می‌شه به یه صفحه جدید redirect انجام داد؟	۱۱۹
	چطوری بررسی می‌کنیم که یه string شامل یه substring هست یا نه؟	۱۲۰
	توی جاواسکریپت، چطوری مقدار یه آدرس email را اعتبارسنجی می‌کنیم؟	۱۲۱
	چطوری می‌تونیم مقدار آدرس url جاری رو بخوینیم؟	۱۲۲
	ویژگی‌های مختلف url روی object مربوط به history کدام هستند؟	۱۲۳
	توی جاواسکریپت چطوری می‌تونیم مقدار یه query-string رو بخوینیم؟	۱۲۴
	چطوری می‌تونیم بررسی کنیم که آیا یه پراپرتی روی آجکت وجود دارد یا نه؟	۱۲۵
	چطوری روی یه object حلقه میزیم؟	۱۲۶
	چطوری تست می‌کنی که یه object خالیه؟	۱۲۷
	چطوری arguments object چیه؟	۱۲۸
	چطوری حرف اول یه رشته رو به حرف بزرگ تبدیل می‌کنی؟	۱۲۹
	مزایا و معایب حلقه for چیا هستند؟	۱۳۰
	چطوری تاریخ جاری رو توی جاواسکریپت نشون میدی؟	۱۳۱
	چطوری دو تا date object رو با هم مقایسه می‌کنی؟	۱۳۲
	چطوری بررسی می‌کنی که یه رشته با یه رشته دیگه شروع می‌شه؟	۱۳۳
	چطوری یه رشته رو trim می‌کنی؟	۱۳۴
	توی جاواسکریپت چطوری می‌تونیم یه زوج مرتب از key یه value بسازیم؟	۱۳۵
	آیا عبارت '!-' عملگر خاصی هست؟	۱۳۶
	چطوری می‌تونیم به متغیرهای مقداری اولیه بدیم؟	۱۳۷
	چطوری می‌تونیم متن‌های چند خطی درست کنیم؟	۱۳۸

صفحه	سوال	ردیف
	مدل app-shell چیه؟	۱۳۹
	چطوری می‌تونیم روی یه تابع property اضافه کنیم؟	۱۴۰
	چطوری می‌تونیم تعداد پارامترهای ورودی یه تابع رو به دست بیاریم؟	۱۴۱
	عبارات Break و continue چی هستن؟	۱۴۲
	توی جاوا‌سکریپت labelها چیکار می‌کنن؟	۱۴۳
	مزایای declare کردن متغیرها در اوایل کد چیه؟	۱۴۴
	مزایای مقداردهی اولیه متغیرها چیه؟	۱۴۵
	روش توصیه شده برای ایجاد object چیه؟	۱۴۶
	چطوری می‌تونیم آرایه آرایه JSON تعریف کنیم؟	۱۴۷
	چطوری می‌تونیم اعداد تصادفی تولید کنیم؟	۱۴۸
	می‌تونی یه تابع تولید اعداد تصادفی توی یه بازه مشخص بنویسی؟	۱۴۹
	Tree-shaking چیه؟	۱۵۰
	دلایل نیاز به tree-shaking کدوما هستن؟	۱۵۱
	آیا استفاده از eval توصیه می‌شه؟	۱۵۲
	Regular-Expression چیه؟	۱۵۳
	متدهای رشته که روی Regular-expression مجاز هستن کداماست؟	۱۵۴
	توی Regex بخش modifiersها چیکار می‌کنه؟	۱۵۵
	پترن‌های regular-expression چیه؟	۱۵۶
	آبجکت RegExp چیه؟	۱۵۷
	چطوری روی یه رشته دنبال یه پترن RegExp می‌گردی؟	۱۵۸

صفحه	سوال	ردیف
	هدف از متدها exec چیه؟	۱۶۰
	چطوری استایل‌های یه المنت HTML رو تغییر میدی؟	۱۶۱
	نتیجه عبارت $3' + 2 + 1$ چی می‌شه؟	۱۶۲
	عبارت debugger چیکار میکنه؟	۱۶۳
	هدف از breakpoint چیه؟	۱۶۴
	آیا می‌تونیم از عبارت‌های رزرو شده در تعریف identifierها(اسم متغیر، کلاس و ...) استفاده کنیم؟	۱۶۵
	چطوری تشخیص بدیم که یه مرورگر mobile هست یا نه؟	۱۶۶
	چطوری بدون Regex تشخیص بدیم که یه مرورگر mobile هست یا نه؟	۱۶۷
	چطوری طول و عرض یه تصویر رو با جاواسکریپت به دست میاری؟	۱۶۸
	چطوری درخواست‌های synchronous HTTP بزنیم؟	۱۶۹
	چطوری درخواست‌های asynchronous HTTP بزنیم؟	۱۷۰
	چطوری یه تاریخ رو به یه تاریخ در timezone دیگه تبدیل کنیم؟	۱۷۱
	چه هایی برای اندازه‌گذاری سایز window به کار میره؟	۱۷۲
	عملگر شرطی سه گانه توی جاواسکریپت چیه؟	۱۷۳
	آیا می‌شه روی عملگر شرطی زنجیره شرط‌ها رو اعمال کرد؟	۱۷۴
	روش‌های اجرای جاواسکریپت بعد از لود شدن صفحه کدوما هستن؟	۱۷۵
	تفاوت‌های بین proto و prototype کدوما هستن؟	۱۷۶
	میتوانی یه مثال از زمانی که واقعا به سمیکولون(;) نیاز هست بزنی؟	۱۷۷
	متدها freeze چیکار میکنه؟	۱۷۸
	هدف از متدها freeze چیه؟	۱۷۹

صفحه	سوال	ردیف
	چرا به متدهای <code>freeze</code> نیاز داریم؟	۱۸۰
	چطوری می‌توانیم زبان ترجیحی یه مرورگر رو تشخیص بدیم؟	۱۸۱
	چطوری می‌توانیم حرف اول همه کلمات یه رشته رو به حرف بزرگ تبدیل کنیم؟	۱۸۲
	چطوری می‌شه تشخیص داد که جاواسکریپت یه صفحه وب غیرفعال شده؟	۱۸۳
	عملگرهای پشتیبانی شده توسط جاواسکریپت کدوما هستن؟	۱۸۴
	پارامتر <code>rest</code> چیکار میکنه؟	۱۸۵
	اگه پارامتر <code>rest</code> رو به عنوان آخرین پارامتر استفاده نکنیم چی می‌شه؟	۱۸۶
	عملگرهای منطقی بازیزی توی جاواسکریپت کدوما هستن؟	۱۸۷
	عملگر <code>spread</code> چیکار میکنه؟	۱۸۸
	چطوری تشخیص میدی که یه آبجکت <code>freeze</code> شده یا نه؟	۱۸۹
	چطوری بررسی کنیم که دو تا مقدار(شامل آبجکت) با هم برابرند یا نه؟	۱۹۰
	هدف از متدهای <code>is</code> روی <code>Object</code> چیه؟	۱۹۱
	چطوری <code>Object</code> ‌های یه <code>Object</code> روی <code>Object</code> می‌کنی؟	۱۹۲
	کاربردهای متدهای <code>assign</code> چیه؟	۱۹۳
	آبجکت <code>Proxy</code> چیه؟	۱۹۴
	هدف از متدهای <code>Seal</code> چیه؟	۱۹۵
	کاربردهای متدهای <code>Seal</code> چیه؟	۱۹۶
	تفاوت‌های بین متدهای <code>freeze</code> و <code>Seal</code> چیا هست؟	۱۹۷
	چطوری تشخیص میدی که یه آبجکت <code>Seal</code> شده یا نه؟	۱۹۸
	چطوری کلید و مقدارهای <code>enumerable</code> رو به دست میاری؟	۱۹۹

ردیف	سوال	صفحه
۲۰۰	تفاوت‌های بین متدهای Object.entries و Object.values چیا هست؟	
۲۰۱	چطوری لیست کلیدهای یه object رو بدست میاری؟	
۲۰۲	چطوری یه object با prototype درست می‌کنی؟	
۲۰۳	WeakSet چیه؟	
۲۰۴	تفاوت‌های بین WeakSet و Set کدوما هستن؟	
۲۰۵	لیست متدهایی که رو WeakSet قابل استفاده هستن رو می‌تونی بگی؟	
۲۰۶	WeakMap چیه؟	
۲۰۷	تفاوت‌های بین WeakMap و Map کدوما هستن؟	
۲۰۸	لیست متدهایی که رو WeakMap قابل استفاده هستن رو می‌تونی بگی؟	
۲۰۹	هدف از متدهای uneval چیه؟	
۲۱۰	چطوری یه URL رو encode می‌کنی؟	
۲۱۱	چطوری یه URL رو decode می‌کنی؟	
۲۱۲	چطوری محتوای یه صفحه رو پرینت می‌گیری؟	
۲۱۳	تفاوت‌های بین eval و uneval چیا هستن؟	
۲۱۴	تابع anonymous چیه؟	
۲۱۵	تفاوت تقدم بین متغیرهای local و global چطوریه؟	
۲۱۶	accessorهای جاوااسکریپت چیکار می‌کنن؟	
۲۱۷	چطوری روی constructor یه Object یه مقدار تعریف می‌کنی؟	
۲۱۸	تفاوت‌های بین get و defineProperty چیا هست؟	
۲۱۹	مزایای استفاده از Setter و Getter چیه؟	

صفحه	سؤال	ردیف
	می‌توانیم getter و setter را با استفاده از متدهای defineProperty تعریف کنیم؟	۲۲۰
	هدف استفاده از switch-case چیه؟	۲۲۱
	چه قواعدی برای استفاده از switch-case باید رعایت بشه؟	۲۲۲
	نوع داده‌های primitive کداما هستن؟	۲۲۳
	روش‌های مختلف دسترسی به property‌های object کداما هستن؟	۲۲۴
	قوانين پارامترهای توابع کداما هستن؟	۲۲۵
	آبجکت error چیه؟	۲۲۶
	چه موقعی خطای syntax دریافت می‌کنیم؟	۲۲۷
	عنوان خطاهای مختلف که روی error-object برミگردن کداما هستن؟	۲۲۸
	عبارات مختلف که در هنگام مدیریت error استفاده می‌شون کداما هستن؟	۲۲۹
	دو نوع مختلف حلقه‌ها تو جاواسکریپت کداما هستن؟	۲۳۰
	آبجکت nodejs چیه؟	۲۳۱
	آبجکت Intl چیه؟	۲۳۲
	چطوری تاریخ و زمان رو بر اساس زبان جاری سیستم کاربر نمایش بدیم؟	۲۳۳
	حلقه‌های synchronous(همزمان) چطوری کار می‌کنن؟	۲۳۵
	Event-loop چیه؟	۲۳۶
	Call-stack چیه؟	۲۳۷
	Event-queue چیه؟	۲۳۸
	Decorator چیه؟	۲۳۹

صفحه	سوال	ردیف
	مقادیر موجود روی آبجکت <code>Intl</code> کدوما هستن؟	۲۴۰
	عملگر <code>Unary</code> چیه؟	۲۴۱
	چطوری المنتهای موجود تو یه آرایه رو مرتب می‌کنی؟	۲۴۲
	هدف از تابع مرتب‌سازی موقع استفاده از متدهای <code>sort</code> چیه؟	۲۴۳
	چطوری آیتم‌های یه آرایه رو معکوس مرتب کنیم؟	۲۴۴
	چطوری حداقل و حداکثر مقدار یه آرایه رو بدست بیاریم؟	۲۴۵
	چطوری حداقل و حداکثر مقدار یه آرایه رو بدون استفاده از متدهای <code>Math</code> بدست بیاریم؟	۲۴۶
	عبارت خالی چیه و هدف از استفاده ازش چیه؟	۲۴۷
	چطوری <code>metadata</code> یه مازول رو بدست میاری؟	۲۴۸
	عملگر <code>comma</code> چیه و چیکار میکنه؟	۲۴۹
	مزایای استفاده از عملگر <code>comma</code> چیه؟	۲۵۰
	چیه؟ <code>TypeScript</code>	۲۵۱
	تفاوت‌های بین <code>TypeScript</code> و <code>javascript</code> کدوما هستن؟	۲۵۲
	مزایای <code>TypeScript</code> نسبت به <code>javascript</code> چیاست؟	۲۵۳
	چیه؟ <code>object-initializer</code>	۲۵۴
	متدهای <code>constructor</code> چیه؟	۲۵۵
	اگه متدهای <code>constructor</code> رو بیش از یه بار توى کلاس بنویسیم چی می‌شه؟	۲۵۶
	چطوری متدهای <code>constructor</code> کلاس والد رو صدا بزنیم؟	۲۵۷
	چطوری <code>Object.prototype</code> یه <code>Object</code> رو به دست میاری؟	۲۵۸
	اگه به متدهای <code>getPrototypeOf</code> رشته پاس بدیم چی می‌شه؟	۲۵۹

صفحه	سوال	ردیف
	چطوری object یه prototype روی یه object دیگه سمت کنیم؟	۲۶۰
	چطوری بررسی می‌کنی که یه object قابل extend هست یا نه؟	۲۶۱
	چطوری جلوی object یه extend را بگیریم؟	۲۶۲
	روش‌های مختلف برای تبدیل یه object به object غیرقابل extend چیه؟	۲۶۳
	چطوری property های متعددی روی object یه تعریف می‌کنی؟	۲۶۴
	منظور از MEAN توی جاواسکریپت چیه؟	۲۶۵
	توی جاواسکریپت Obfuscation چیه و چیکار می‌کنه؟	۲۶۶
	چه نیازی به Obfuscate کردن داریم؟	۲۶۷
	Minification چیه؟	۲۶۸
	مزایای minification یا کم حجم‌سازی چیه؟	۲۶۹
	تفاوت‌های بین Encryption و Obfuscation چیه؟	۲۷۰
	ابزارهای مختلف برای minification کدوما هستن؟	۲۷۱
	چطوری اعتبارسنجی فرم رو با javascript انجام میدی؟	۲۷۲
	چطوری اعتبارسنجی فرم رو بدون javascript انجام میدی؟	۲۷۳
	متدهای موجود روی DOM برای اعتبارسنجی کدوما هستن؟	۲۷۴
	مقادیر موجود روی DOM برای اعتبارسنجی کدوما هستن؟	۲۷۵
	مقادیر موجود روی input برای اعتبارسنجی کدوما هستن؟	۲۷۶
	یه مثال از استفاده ویژگی rangeOverflow می‌تونی بزنی؟	۲۷۷
	جاواسکریپت قابلیت استفاده از enum رو پیش‌فرض توی خودش داره؟	۲۷۸
	چیه enum؟	۲۷۹
	چطوری همه property های یه object رو به دست بیاریم؟	۲۸۰

ردیف	سوال	صفحه
۲۸۱	چطوری property-descriptor های آبجکت را بدست بیاریم؟	
۲۸۲	گزینه‌هایی که موقع تعریف ویژگی object descriptor با داریم کدام هستند؟	
۲۸۳	چطوری کلاس‌ها را extend می‌کنی؟	
۲۸۴	چطوری آدرس صفحه را بدون رفرش صفحه عوض کنیم؟	
۲۸۵	چطوری بررسی می‌کنی که یه آرایه یه مقدار مشخص را داره یا نه؟	
۲۸۶	چطوری آرایه‌های scalar را با هم مقایسه می‌کنی؟	
۲۸۷	چطوری می‌شه پارامترهای صفحه را از متدها GET گرفت؟	
۲۸۸	چطوری اعداد را می‌شه سه رقم سه رقم جدا کرد؟	
۲۸۹	تفاوت بین javascript و java چیه؟	
۲۹۰	آیا جاواسکریپت namespace را پشتیبانی می‌کنه؟	
۲۹۱	چطوری namespace تعریف می‌کنی؟	
۲۹۲	چطوری می‌تونیم تکه کد جاواسکریپت داخل یه iframe را از صفحه والد صدا بزنیم؟	
۲۹۳	چطوری می‌شه اختلاف timezone را از آبجکت date بگیریم؟	
۲۹۴	چطوری فایل‌های CSS و JS را به شکل داینامیک بارگذاری کنیم؟	
۲۹۵	روش‌های مختلف برای پیدا کردن element‌ها توی DOM کدام هستند؟	
۲۹۶	چیه؟ jQuery	
۲۹۷	مونتور V8 جاواسکریپت چیه؟	
۲۹۸	چرا ما جاواسکریپت را به عنوان یه زبان داینامیک می‌شناسیم؟	
۲۹۹	عملگر void چیکار می‌کنه؟	
۳۰۰	چطوری می‌شه نمایشگر موس صفحه را به درحال لود تغییر داد؟	

صفحه	سوال	ردیف
	چطوری می‌شه یه حلقه بی‌نهایت درست کرد؟	۳۰۱
	چرا باید در استفاده از عبارت with تجدیدنظر کرد؟	۳۰۲
	خروجی این حلقه‌ها چی می‌شه؟	۳۰۳
	می‌تونی یه سری از ویژگی‌های ES6 رو اسم ببری؟	۳۰۴
	آیا ES6 چیه؟	۳۰۵
	آیا می‌تونیم متغیرهای تعریف شده با let و const رو مجددا declare کنیم؟	۳۰۶
	آیا استفاده از const برای تعریف متغیر اونا رو immutable می‌کنه؟	۳۰۷
	چطوری رشته‌های پیش‌فرض چی هستن؟ parameter	۳۰۸
	چطوری template-literal ها چی هستن؟	۳۰۹
	چطوری template-literal رو توی hataها می‌نویسیم؟	۳۱۰
	چطوری template-literal های تودرتو چی هستن؟	۳۱۱
	چطوری tagged-template ها چی هستن؟	۳۱۲
	رشته‌های خام چی هستن؟	۳۱۳
	کردن با assign چیه و چطوری انجام می‌شه؟ destructuring	۳۱۴
	موقع assign کردن با destructuring چطوری می‌شه مقدار اولیه تعریف کرد؟	۳۱۵
	چطوری می‌تونیم مقدار یه آرایه رو با استفاده از destructuring assignment تعویض کنیم؟	۳۱۶
	چطوری Enhanced-object-literal ها چی هستن؟	۳۱۷
	چیهای داینامیک چی هستن؟ import	۳۱۸
	کاربرد import های داینامیک چیه؟	۳۱۹
	آرایه‌های نوع‌دار (typed-arrays) چیه؟	۳۲۰

ردیف	سوال	صفحه
۳۲۱	مزایای لودر مازول‌ها چیه؟	
۳۲۲	collation چیه؟	
۳۲۳	عبارت for...of چیه؟	
۳۲۴	خروجی عملگر spread روی آرایه زیر چیه؟	
۳۲۵	آیا PostMessage امنه؟	
۳۲۶	مشکلات استفاده از wildcard در postmessage با origin روی postMessage چیه؟	
۳۲۷	چطوری از دریافت postMessage های ناخواسته و نامن از طرف هکرهای جلوگیری کنیم؟	
۳۲۸	می‌توانیم کلا postMessage را غیرفعال کنیم؟	
۳۲۹	آیا postMessage ها به صورت synchronous و همزمان کار می‌کنند؟	
۳۳۰	پارادیم زبان جاواسکریپت چیه؟	
۳۳۱	تفاوت‌های بین جاواسکریپت داخلی و خارجی چیه؟	
۳۳۲	آیا جاواسکریپت سریعتر از اسکریپت‌های سمت سرور است؟	
۳۳۳	چطوری وضعیت چک بودن یه checkbox رو بدست بیاریم؟	
۳۳۴	هدف از عملگر double-tilde چیه؟	
۳۳۵	چطوری یه کاراکتر رو به کد ASCII تبدیل کنیم؟	
۳۳۶	ArrayBuffer چیه؟	
۳۳۷	خروجی کد زیر چی خواهد بود؟	
۳۳۸	هدف از Error-object چیه؟	
۳۳۹	هدف از EvalError-object چیه؟	
۳۴۰	خطاهایی که در حالت strict-mode رخ میدن ولی در غیر اون وجود ندارن کدام است؟	

صفحه	سوال	ردیف
	آیا همه object‌ها دارای prototype هستن؟	۳۴۱
	تفاوت‌های بین argument و parameter چیه؟	۳۴۲
	هدف از متدهای some روی آرایه‌ها چیه؟	۳۴۳
	چطوری دو یا تعداد بیشتری از آرایه‌ها را با هم ترکیب کنیم؟	۳۴۴
	تفاوت‌های بین Shallow و Deep کپی چیه؟	۳۴۵
	چطوری می‌توانیم به یه تعداد مشخص از یه رشته کپی کنیم؟	۳۴۶
	چطوری همه string‌ها را با regular-expression match کنیم؟	۳۴۷
	چطوری یه رشته را از اول یا از آخر trim کنیم؟	۳۴۸
	خروجی کنسول زیر با عملگر unary چی می‌شه؟	۳۴۹
	آیا جاواسکریپت از mixin‌ها استفاده می‌کنه؟	۳۵۰
	تابع thunk چیه و چیکار می‌کنه؟	۳۵۱
	چیکار می‌کنن؟ asynchronous‌های thunk چیکار می‌کنن؟	۳۵۲
	خروجی فراخوانی‌های توابع زیر چی می‌شه؟	۳۵۳
	چطوری همه خطوط جدید را از یه رشته حذف کرد؟	۳۵۴
	تفاوت بین repaint و reflow چیه؟	۳۵۵
	اگه قبل از یه آرایه عملگر نفی «!» بزاریم چی می‌شه؟	۳۵۶
	اگه دو تا آرایه را با هم جمع بندیم چی می‌شه؟	۳۵۷
	اگه عملگر جمع «+» روی قبل از مقادیر falsy قرار بدیم چی می‌شه؟	۳۵۸
	چطوری با استفاده از آرایه‌ها و عملگرهای منطقی می‌توانیم رشته self را تولید کنیم؟	۳۵۹
	چطوری می‌توانیم مقادیر falsy را از آرایه حذف کنیم؟	۳۶۰

ردیف	سوال	صفحه
۳۶۱	چطوری مقادیر تکراری رو از یه آرایه حذف کنیم؟	
۳۶۲	چطوری همهای همزمان با alias destructuring کار می‌کنن؟	
۳۶۳	چطوری آیتم‌های یه آرایه رو بدون استفاده از متدهای map پیمایش کنیم؟	
۳۶۴	چطوری یه آرایه رو خالی کنیم؟	
۳۶۵	چطوری اعداد رو با تعداد رقم اعشار مشخص رند می‌کنی؟	
۳۶۶	ساده‌ترین روش برای تبدیل آرایه به object چیه؟	
۳۶۷	چطوری یه آرایه با یه سری داده درست کنیم؟	
۳۶۸	متغیرهای موجود روی آبجکت console کدوماً هستن؟	
۳۶۹	می‌شه پیام‌های کنسول رو استایل‌دهی کرد؟	
۳۷۰	هدف از متدهای dir و console چیه؟	
۳۷۱	آیا می‌شه المنت‌های HTML رو توی console دیباگ کرد؟	
۳۷۲	چطوری می‌شه داده‌ها رو به شکل جدولی توی console نمایش بدیم؟	
۳۷۳	چطوری می‌شه بررسی کرد که یه پارامتر Number هست یا نه؟	
۳۷۴	چطوری یه متن رو می‌تونیم به clipboard کپی کنیم؟	
۳۷۵	چطوری می‌شه timestamp رو بدست آورد؟	
۳۷۶	چطوری یه آرایه چندسطحی رو تک سطحی کنیم؟	
۳۷۷	ساده‌ترین روش برای بررسی چندشرطی چیه؟	
۳۷۸	چطوری کلیک روی دکمه برگشت مرورگر رو متوجه بشیم؟	
۳۷۹	چطوری می‌تونیم کلیک راست رو غیرفعال کنیم؟	
۳۸۰	چی هستن؟ object-wrappers	
۳۸۱	AJAX چیه؟	

صفحه	سوال	ردیف
	روش‌های مختلف مدیریت یه کد Asynchronous چیه؟	۳۸۲
	چطوری یه درخواست fetch رو کنسل کنیم؟	۳۸۳
	Speech-API چیه؟	۳۸۴
	حداقل timeout توی throttling چقدره؟	۳۸۵
	چطوری می‌شه یه timeout صفر توی مرورگر اجرا کرد؟	۳۸۶
	چطوری event-loop چی هستن؟	۳۸۷
	microtask چی هستن؟	۳۸۸
	event-loop‌های مختلف کدوما هستن؟	۳۸۹
	هدف از queueMicrotask چیه؟	۳۹۰
	چطوری می‌شه از کتابخونه‌های جاواسکریپت توی فایل typescript استفاده کرد؟	۳۹۱
	تفاوت‌های بین observable‌ها و Promise‌ها کدوما هستن؟	۳۹۲
	heap چیه؟	۳۹۳
	event-table چیه؟	۳۹۴
	microTask چیه؟	۳۹۵
	تفاوت بین shim و polyfill چیه؟	۳۹۶
	چطوری متوجه primitive یا غیر primitive بودن یه نوع داده می‌شیم؟	۳۹۷
	babel چیه؟	۳۹۸
	آیا Node.js به شکل کامل تک thread کار می‌کنه؟	۳۹۹
	کاربردهای مرسوم observable‌ها کدوما هستن؟	۴۰۰
	RxJS چیه؟	۴۰۱

ردیف	سوال	صفحه
۴۰۲	تفاوت بین function-declaration و Function-constructor چیه؟	
۴۰۳	شرط Short-circuit یا اتصال کوتاه چیه؟	
۴۰۴	ساده‌ترین روش برای تغییر سایز یه آرایه چیه؟	
۴۰۵	observable چیه؟	
۴۰۶	تفاوت‌های بین توابع و کلاس‌ها چیه؟	
۴۰۷	تابع async چیه؟	
۴۰۸	چطوری خطاهای، ایجاد شده هنگام استفاده از Promise‌ها را کنترل کنیم؟	
۴۰۹	Deno چیه؟	
۴۱۰	توی جاوا‌سکریپت چطوری یه object قابل پیمایش درست کنیم؟	
۴۱۱	روش مناسب برای فراخوانی تابع بازگشتی چیه؟	
۴۱۲	چطوری بررسی کنیم که یه آبجکت Promise هست یا نه؟	
۴۱۳	چطوری متوجه بشیم که یا تابع با تابع constructor صدا زده شده یا نه؟	
۴۱۴	تفاوت‌های بین آبجکت argument و پارامتر rest چیه؟	
۴۱۵	تفاوت‌های بین عملگر spread و پارامتر rest چیه؟	
۴۱۶	نوع‌های مختلف generatorها کدوما هستن؟	
۴۱۷	built-in های iterable کدوما هستن؟	
۴۱۸	تفاوت‌های بین حلقه for...in و for...of چیه؟	
۴۱۹	چطوری property‌های instance و غیر instance‌ای تعریف می‌کنی؟	
۴۲۰	تفاوت‌های بین Number.isNaN و NaN یا isNaN کدوما هستن؟	



## پیشگفتار

در ابتداء، ممنونم از شما که با خرید این کتاب بهمون کمک کردین که بتونیم قدمی در راه کمک به افراد نیازمند برداریم و با درآمد حاصل از فروش این کتاب کمکی هر چند کوچیک در راه مسئولیت اجتماعی مون برداریم، به هم‌دیگه کمک کنیم، با هم مهربون‌تر باشیم و در کنار هم پیشرفت کنیم. تشکر گرم من رو، دورادور پذیرا باشین و امیدوارم این کتاب به جهت افزایش دانش‌تون و کمک به پیشرفت شغلی‌تون کمکی کرده باشه.

کتابی که پیش‌روی شماست، حاصل تلاش نه فقط من، بلکه چندین نفر از بهترین و حرفه‌ای‌ترین دوستان بندۀ هم هست که در اینجا به ترتیب میزان زحمتی که متقبل شدن اسمشونو قید می‌کنم و کمال تشکر رو ازشون دارم:

- جعفر رضائی
- مهسا مصباح

این عزیزان هر کدام با کمک‌هاشون برای ترجمه، ویراستاری‌هاشون و حتی دلگرمی‌هاشون باعث شدن این مجموعه به زبان فارسی آماده بشه و به شکل چاپی بتونه به دستان شما برسه.

## ماریوتک

برادر من جعفر رضائی، پلتفرم ماریوتک رو با هدف آموزش اصولی و رایگان، تاسیس کرد و من هم این کتاب رو از مجموعه ماریوتک منتشر میکنم. ما ماریوتک رو متعلق به همه می‌دونیم، پس اگه بعضی تایم‌های بیکاری داری که فکر می‌کنی می‌تونی باهایمون توی این مسیر همراه باشی حتما بهم ایمیل بزن. ایده‌های ماریوتک برای افزایش آگاهی و دانش تا حد امکان رایگان خواهد بود و تا به اینجا هم، تنها هزینه‌های چاپ برداشته شده و مابقی به موسسات خیریه داده شدن.

## مطلوب کتاب

مطلوب این کتاب می‌تونن تا حد بسیار خوبی دانش شما رو توی مسائل کلیدی مربوط جاواسکریپت و کتابخونه‌های پیرامون اون افزایش بدن. سوالات چالشی و کلیدی مطرح شده توی کتاب اکثرا سوالاتی هستند که توی مصاحبه‌های استخدامی پرسیده می‌شن و مسلط بودن به اونا می‌تونه شانس موفقیت شما برای موقعیت‌های شغلی که مد نظر دارین افزایش بده. مطالب این کتاب به دلیل ترجمه بودن تا حد زیادی قابل دستکاری نبودن و سعی شده تا حد امکان حق گردآورنده محفوظ باشه و با نسخه اصلی سورس که توسط Sudheer Jonna جمع‌آوری شده تفاوت معنایی نداشته باشه. بخشی از مطالب کتاب اصلی به خاطر قدیمی بودن منقضی شده بودن و به عنوان مترجم بخش‌های زیادی از نمونه کدها و مطالب قدیمی تصحیح شدند. در آخر، امیدوارم همیشه شاد و خندان و خوشحال باشین. مخلصیم



## ۱. روش‌های ایجاد object‌ها توی جاوا‌اسکریپت کدوما هستن؟

۱. سازنده آبجکت: ساده‌ترین راه برای ایجاد یه آبجکت خالی استفاده از کلاس Object. در حال حاضر این روش توصیه نمی‌شه.

```
const object = new Object();
```

۲. متد استاتیک **Object.create**: متد استاتیک **Object.create** روی Object با انتقال آبجکت به عنوان پارامتر، یه آبجکت جدید ایجاد می‌کنه.

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

۳. استفاده از **syntax آبجکت**: با مقداردهی یه متغیر توسط یه آبجکت ساده داخل syntax آبجکت.

```
const object = {
  name: "Ali Karimi",
  age: 30,
};
```

۴. **تابع new constructor**: هر تابعی که بخوایم رو ایجاد می‌کنیم و از طریق عملگر new یه نمونه آبجکت جدید می‌سازیم.

```
function Person(name) {
  const object = {};
  object.name = name;
  object.age = 30;

  return object;
}

const object = new Person("Ali Karimi");
```

۵. **تابع به همراه prototype** شبیه سازنده تابع هستش اما از prototype برای متدها و خصوصیاتشون استفاده می‌کنه.

```
function Person() {}
Person.prototype.name = "Ali Karimi";

const object = new Person();
```

این معادل نمونه‌ای هستش که با متدهای آبجکت با prototype تابع ایجاد شده و تابع رو با یه نمونه و پارامترهاش به عنوان آرگومان فراخوانی میکنه.

```
function func {};
new func(x, y, z);
```

(ب)

```
// Create a new instance using function prototype.
const newInstance = Object.create(func.prototype)

// Call the function
const result = func.call(newInstance, x, y, z),

// If the result is a non-null object then use it otherwise
// just use the new instance.
console.log(result && typeof result === 'object' ? result :
newInstance);
```

۶. استفاده از کلاس‌های ES6: توی ES6 کلمه کلیدی class رو برای ایجاد آبجکت‌ها معرفی کردن.

```
class Person {
    constructor(name) {
        this.name = name;
    }
}

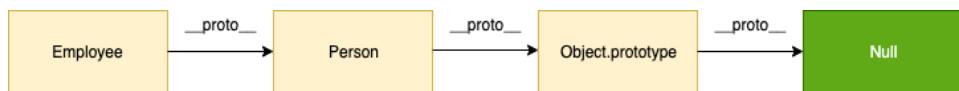
const object = new Person("Ali Karimi");
```

۷. الگو یا پترن Singleton: آبجکت‌ایه که فقط یه بار قابل نمونه‌گیری هستش و فراخوانی‌های بعدی روی سازندهش همون نمونه اولی رو برمی‌گردونه و اینطوری می‌شه مطمئن شد که به طور تصادفی نمونه‌های مختلف ایجاد نمی‌شه.

```
const object = new (function () {
    this.name = "Ali Karimi";
})();
```

## ۴. زنجیره prototype چیه؟

زنجیره **prototype** برای ساخت انواع جدیدی از آبجکت‌ها براساس موارد موجود استفاده می‌شود. این کار شبیه ارث بری توی یه زبان مبتنی بر کلاس هستش. روی نمونه آبجکت از طریق ویزگی **(Object.getPrototypeOf(object))** یا **\*\*proto\*\*** در دسترسه در حالی که توی **Object.prototype** عملکرد سازنده‌ها از طریق **object.prototype** در دسترسه.



## ۵. تفاوت‌های بین bind، apply و call چیا هستن؟

**متدهای call و apply:** متدهای **call** و **apply** یه تابع با یه مقدار **this** و آرگومان‌های ارائه شده رو دونه دونه فراخوانی میکنه.

```

const employee1 = {
  firstName: "John",
  lastName: "Rodson",
};

const employee2 = {
  firstName: "Jimmy",
  lastName: "Baily",
};

function invite(greeting1, greeting2) {
  console.log(
    greeting1 + " " + this.firstName + " " +
    this.lastName + ", " + greeting2
  );
}

invite.call(employee1, "Hello", "How are you?"); // Hello John Rodson, How are you?
invite.apply(employee2, ["Hello", "How are you?"]); // Hello Jimmy Baily, How are you?
  
```

**متدهای apply و call:** تابع رو فراخوانی میکنه و بهمون اجازه میده تا آرگومان‌ها رو به عنوان یه آرایه منتقل کنیم.

```

const employee1 = { firstName: "John", lastName: "Rodson" };
const employee2 = { firstName: "Jimmy", lastName: "Baily" };

function invite(greeting1, greeting2) {
    console.log(
        greeting1 + " " + this.firstName + " " +
        this.lastName + ", " + greeting2
    );
}

invite.apply(employee1, ["Hello", "How are you?"]); // Hello
John Rodson, How are you?
invite.apply(employee2, ["Hello", "How are you?"]); // Hello
Jimmy Baily, How are you?

```

**متد bind:** یه تابع جدید برمی‌گردونه، در حالی که بهمون اجازه میده هر تعداد آرگومانی که می‌خوایم رو توی یه آرایه منتقل کنیم.

```

const employee1 = { firstName: "John", lastName: "Rodson" };
const employee2 = { firstName: "Jimmy", lastName: "Baily" };

function invite(greeting1, greeting2) {
    console.log(
        greeting1 + " " + this.firstName + " " +
        this.lastName + ", " + greeting2
    );
}

const inviteEmployee1 = invite.bind(employee1);
const inviteEmployee2 = invite.bind(employee2);
inviteEmployee1("Hello", "How are you?"); // Hello John Rodson,
How are you?
inviteEmployee2("Hello", "How are you?"); // Hello Jimmy Baily,
How are you?

```

متدهای `call` و `apply` تقریباً قابل تعویض هستن. هر دو بلافاصله تابع فعلی رو اجرا می‌کنن. شما باید تصمیم بگیرین که ارسال آرایه در آرایه آسون‌تره یا فهرستی از آرگومان‌های جدا شده با کاما. باید یادمون باشه که متدهای `call` برای کاما (فهرست جدا شده) و `apply` برای حالت آرایه‌ایه. در حالی که `bind` یه تابع جدید ایجاد میکنه و می‌تونه با دریافت «`this`» روی اولین پارامتر ارسال شده کانتکست آبجکت جدید ساخته رو تنظیم کنه.

## ۴. فرمت JSON چیه و عملیات‌های معمول روی اون چیا هستن؟

JSON یه قالب داده مبتنی بر متن‌هه که از آبجکت جاواسکریپت (Javascript) syntax آبجکت جاواسکریپت (Object syntax) پیروی میکنه و توسط Douglas Crockford رایج شد. کاربردش زمانیه که بخواهیم داده‌ها رو از طریق شبکه انتقال بدیم و اساساً یه فایل متنی با پسوند json و نوع application/json داشته باشیم. اکثراً دو عملیات زیر روی JSON انجام میشه:

**تجزیه (Parsing):** تبدیل یه رشته به یه آبجکت جاواسکریپت (native Object).

```
JSON.parse(text);
```

**رشته‌سازی (stringify):** تبدیل یه آبجکت جاواسکریپت به یه رشته تا بتونه از طریق شبکه منتقل بشه یا یه جایی ذخیره بشه.

```
JSON.stringify(object);
```

## ۵. هدف از متده slice روی آرایه‌ها چیه؟

متده slice عناصر انتخاب شده توی یه آرایه رو به عنوان یه آرایه جدید برمی‌گردونه. این عناصر رو از اولین آرگومان داده شده انتخاب میکنه و با آرگومان پایانی و اختیاری داده شده بدون در نظر گرفتن آخرین عنصر به پایان می‌رسونه. اگه آرگومان دوم رو حذف کنیم تا آخر آرایه همه عناصر رو انتخاب میکنه. چند تا مثال در مورد نحوه کارکردش ببینیم:

```
const arrayIntegers = [1, 2, 3, 4, 5];
const arrayIntegers1 = arrayIntegers.slice(0, 2); // returns
[1, 2]
const arrayIntegers2 = arrayIntegers.slice(2, 3); // returns
[3]
const arrayIntegers3 = arrayIntegers.slice(4); // returns [5]
```

**نکته:** متده slice آرایه اصلی رو تغییر نمیده ولی یه زیرمجموعه به عنوان یه آرایه جدید برمی‌گردونه.

## ۶. هدف از متدهای splice و slice چیه؟

متدهای **splice** برای اضافه کردن یه عنصر به آرایه یا حذف از اون استفاده می‌شوند و مورد یا موارد حذف شده را برمی‌گردونه. آرگومان اول موقعیت آرایه را برای درج یا حذف مشخص می‌کنند و آرگومان اخباری دوم تعداد عناصر برای حذف را مشخص می‌کنند و مابقی هر آرگومان اضافه‌ای که به این متدها ارسال بشوند، به آرایه اضافه می‌شوند. چند تا مثال در مورد نحوه کارکردش ببینید:

```
const arrayIntegersOriginal1 = [1, 2, 3, 4, 5];
const arrayIntegersOriginal2 = [1, 2, 3, 4, 5];
const arrayIntegersOriginal3 = [1, 2, 3, 4, 5];

const arrayIntegers1 = arrayIntegersOriginal1.splice(0, 2); // returns [1, 2]; original array: [3, 4, 5]
const arrayIntegers2 = arrayIntegersOriginal2.splice(3); // returns [4, 5]; original array: [1, 2, 3]
const arrayIntegers3 = arrayIntegersOriginal3.splice(3, 1, "a", "b", "c"); //returns [4]; original array: [1, 2, 3, "a", "b", "c", 5]
```

**نکته:** متدهای **splice** و **slice** می‌کنند که مقداری از اصلی را اصلاح می‌کنند یعنی متغیر اصلی را تحت تاثیر قرار میدهند و آرایه حذف شده را برمی‌گردونند.

## ۷. تفاوت متدهای splice و slice چیا هستن؟

splice	slice
آرایه اصلی را تغییر میده (یا <b>mutable</b> ) تغییرپذیر	آرایه اصلی را تغییر نمیده (یا <b>immutable</b> ) تغییرنپذیر
عناصر حذف شده را به عنوان آرایه برمی‌گردونه	زیر مجموعه آرایه اصلی را برمی‌گردونه
برای درج عناصر به آرایه یا حذف از اون استفاده می‌شوند	برای انتخاب عناصر از آرایه استفاده می‌شوند

## ۸. تفاوتهای Map و Object چیا هستن؟

آبجکتها شبیه به **Map** ها هستن (البته خود Map هم آبجکته و اینجا منظور شاعر خود شخص Object را میگه) از این جهت که هردو بهمن این امکان رو میدن که مقادیر رو روی کلیدهای مشخصی تعریف کنیم، مقادیر رو بازیابی کنیم، کلیدها رو حذف کنیم و بینیم چیزی توی یه کلید ذخیره شده یا نه. به همین دلیل در طول تاریخ از آبجکتها به عنوان **HashMap** و **Map** استفاده‌های زیادی شده. اما تفاوتهای مهمی وجود داره که استفاده از **Map** رو توی موارد خاص ارجعیت میده.

۱. کلیدهای یه آبجکت رشته‌ها و **Symbol** ها هستن، در حالی که برای Map مقادیر مختلفی میتوانه وجود داشته باشه که شامل توابع، آبجکتها و هر نوع اولیه دیگهای میشه.

۲. کلیدهای Map مرتب میشن در حالی که کلیدهای اضافه شده به آبجکت اینپروری نیستن. بنابراین موقع تکرار روی اون، آبجکت map کلیدها رو به ترتیب اضافه شدنشون برمی‌گردونه.

۳. اندازه Map رو میتونیم به راحتی با ویژگی سایز بدست بیاریم، در حالی که تعداد خصوصیات یه آبجکت باید به صورت دستی و با ساخت یه آرایه از روی کلیدها و یا مقادیرش حساب بشه.

۴. Map قابل تکراره و میتوانه مستقیما تکرار بشه، در حالی که تکرار روی یه آبجکت مستلزم بدست آوردن کلیدهای اون به روشی خاص و تکرار روی اونهاست.

۵. آبجکت یه **prototype** داره، بنابراین کلیدهای پیش‌فرض توی Object وجود داره که که اگه دقت نکنیم ممکنه با کلیدهای اون برخورد کنه. از زمان ES5 میتوانیم با استفاده از **Map = Object.create(null)**، این قضیه رو دور بزنیم ولی بهندرت این کار انجام میشه.

۶. Map ممکنه توی سناریوهای شامل جمع و حذف مکرر جفت کلیدها عملکرد بهتری داشته باشه.

## ۹. تفاوتهای بین عملگرهای == و === چیا هستن؟

جاواسکریپت مقایسه برابری سخت (`==`، `!=`) و تبدیل نوع (`==`، `!=`، `==!`) رو فراهم میکنه. عملگرهای سختگیرانه نوع متغیر رو در نظر می‌گیرند، در حالی که عملگرهای غیر دقیق، اصلاح/تبدیل نوع رو بر اساس مقادیر متغیرها انجام میدن. اپراتورهای سختگیر از شرایط زیر برای انواع مختلف پیروی می‌کنن.

۱. دو رشته زمانی کاملاً برابر هستن که توالی کاراکترهای یکسان، طول یکسان و کاراکترهای مشابه در موقعیت‌های منتظر داشته باشن.
  ۲. دو عدد زمانی که از نظر عددی مساوی باشن کاملاً برابر هستن. یعنی داشتن مقدار عددی یکسان.
- دو مورد خاص در این مورد وجود داره:
۱. NaN با هیچ چیز از جمله NaN برابر نیست.
  ۲. صفرهای مثبت و منفی با هم برابرند.
۳. اگه هر دو درست یا نادرست باشن، دو عملوند بولین کاملاً برابر هستن.
  ۴. اگه دو شیء به یه شیء اشاره کنن کاملاً برابر هستن.
۵. انواع Null و Undefined با === برابر نیستن، بلکه با == برابر هستن. یعنی null == undefined --> true اما null === undefined --> false
- یه چندتا مثال که موارد بالا رو پوشش میدن:

```

0 == false    // true
0 === false   // false
1 == "1"      // true
1 === "1"     // false
null == undefined // true
null === undefined // false
'0' == false // true
'0' === false // false
[]==[] or []==={} //false, refer different objects in memory
{}=={} or {}==={} //false, refer different objects in memory

```

## ۱۰. توابع arrow-function یا lambda چی هستن؟

توابع arrow function ها به صورت ساده‌تر و کوتاه‌تر تعریف می‌شن و شئ this ، متغیر جادویی new.target یا super ، متده arguments استفاده نمی‌شن. constructor یا سازنده ندارن. این توابع بدون متده هستن و به عنوان

## ۱۱. یه تابع first-class چجور تابعیه؟

توى جاواسکریپت، توابع آبجکت‌های کلاس اول یا first class هستن. توابع کلاس اول توى هر زبان برنامه‌نویسی، زمانی معنی میدن که توابع توى اون زبان باهاشون مثل بقیه

متغیرها رفتار بشه.  
برای مثال، توی جاوااسکریپت، یه تابع می‌تونه به عنوان آرگومان به یه تابع دیگه پاس داده بشه، می‌تونه به عنوان مقدار نهایی یه تابع دیگه برگشت داده بشه و می‌تونه به یه متغیر دیگه به عنوان مقدار اختصاص داده بشه. برای مثال توی کد زیر، تابع handler به عنوان listener پاس داده شده.

```
const handler = () => console.log("This is a click handler function");

/**
 * Usage of `handler` method
 */
document.addEventListener("click", handler);
```

## ۱۲. یه تابع first-order چجور تابعیه؟

تابع مرتبه اول یا first-order تابعیه که هیچ تابع دیگه‌ای رو به عنوان آرگومان قبول نمیکنه و هیچ تابعی رو هم به عنوان مقدار برگشتی یا return value برنمی‌گردونه. مثل:

```
const firstOrder = () => console.log("I am a first order function!");
```

## ۱۳. یه تابع higher-order چجور تابعیه؟

تابع مرتبه بالا توابعی هستن که یه تابع رو به عنوان پارامتر ورودی دریافت و یا به عنوان خروجی ارسال میکنن. مثل:

```
const firstOrderFunc = () =>
  console.log("Hello I am a First order function");
const higherOrder = (ReturnFirstOrderFunc) =>
  ReturnFirstOrderFunc();

higherOrder(firstOrderFunc);
```

## ۱۴. یه تابع unary چجور تابعیه؟

تابع unary تابعیه که فقط یه آرگومان ورودی دریافت میکنه. مثل:

```
// Add 10 to the given argument and display the value
const unaryFunction = (a) => {
  console.log(a + 10);
};
```

## ۱۵. توابع یعنی چی؟

به فرایندی که در اون یه تابع با چندین آرگومان رو به مجموعه‌ای از توابع که فقط یه آرگومان دریافت میکنن، تبدیل کنیم currying می‌گیم. Curryning از نام یه ریاضیدان به اسم unary Haskell Curry گرفته شده. با استفاده از Curryning در واقع یه تابع رو به یه تابع currying می‌کنیم. بیاین یه مثال از یه تابع با چندین آرگومان و تبدیلش به تابع بزرگیم.

```
const multiArgFunction = (a, b, c) => a + b + c;
const curryUnaryFunction = (a) => (b) => (c) => a + b + c;
curryUnaryFunction(1); // returns a function: b => c => 1 + b
+ c
curryUnaryFunction(1)(2); // returns a function: c => 3 + c
curryUnaryFunction(1)(2)(3); // returns the number 6
```

تابع Curried برای بهبود قابلیت استفاده مجدد کد و ترکیب عملکردی عالی هستن.

## ۱۶. چه توابعی pure هستن؟

تابع pure تابعیه که مقدار برگشتیش فقط توسط آرگومان‌هاش تعیین می‌شه بدون هیچ side effect یا عوارض جانبی. برای مثال اگه ما یه تابع رو n بار در n جای مختلف برنامه فراخوانی کنیم همیشه یه مقدار مشخص برگشت داده می‌شه. بیاین یه مثال از تفاوت بین توابع pure و توابع غیرpure بزرگیم.

```
//Impure
let numberArray = [];
const impureAddNumber = (number) => numberArray.push(number);

//Pure
const pureAddNumber = (number) => (argNumberArray) =>
  argNumberArray.concat([number]);

//Display the results
console.log(impureAddNumber(6)); // returns 1
console.log(numberArray); // returns [6]
console.log(pureAddNumber(7)(numberArray)); // returns [6, 7]
console.log(numberArray); // returns [6]
```

بر اساس تکه کدهای بالا، تابع push با تغییر روی آرایه و برگرداندن شماره ایندکس push که مستقل از مقدار پارامتر هستش، یه تابع ناخالص یا Impure به حساب می‌آید. در حالی که از یه طرف متده concat آرایه رو می‌گیره و اونو با یه آرایه دیگه ترکیب می‌کنه و یه آرایه کاملاً جدید و بدون هیچ side-effect تولید می‌کنه. همچنین مقدار برگشتی با آرایه قبلی ترکیب شده هستش.

یادتون باشه که توابع خالص یا pure مهمان چون اوها unite-test رو بدون هیچ side-effect و بدون نیاز به dependency-injection ساده می‌کنن. اوها همچنین از اتصال محکم بین بخش‌های مختلف برنامه جلوگیری می‌کنن و با نداشتن side-effect، احتمال بروز خطا تو برنامه رو کمکتر می‌کنن. این اصول کنار هم جمع می‌شن و کنار **Immutability**

و باعث می‌شن که از const به جای let استفاده بشه.

**نکته:** دقت کنیم که چاپ یه مقدار روی کنسول و یا درخواست api-call هم side-effect محسوب میشه.

## ۱۷. هدف از کلمه کلیدی let چیه؟

دستور let یه متغیر محلی block scope تعریف می‌کنه. از این رو متغیرهایی که با کلمه کلیدی let تعریف می‌شن محدود به همون اسکوپی که تو ش تعریف شدن، می‌شن و فقط دستورها و عبارتهای توی همون اسکوپ بهش دسترسی دارن. درحالی که متغیرهای تعریف شده با کلمه کلیدی var برای تعریف یه متغیر توی سطح global و یا به شکل محلی و برای استفاده در کل تابع بدون در نظر گرفتن اسکوپی که تو ش تعریف شده، استفاده می‌شه. بیاین برای نشون دادن کاربردش یه مثال بزنیم.

```

let counter = 30;
if (counter === 30) {
  let counter = 31;
  console.log(counter); // 31
}
console.log(counter); // 30 (because if block variable won't
exist here)

```

## ۱۸. تفاوت‌های کلمات کلیدی `var` و `let` چیا هستن؟

: var

۱. دامنه تابع داره
۲. متغیرها Hoist میشن
۳. از ابتدای جاواسکریپت در دسترس هستش

:let

۱. به عنوان بخشی از ES6 معرفی شده
۲. محدود به scope یا دامنه هستش
۳. Hoist شده ولی قابل استفاده نیست

بیاین با یه مثال تفاوتش رو بهتر ببینیم:

```

function userDetails(username) {
  if (username) {
    console.log(salary); // undefined(due to hoisting)
    console.log(age); // error: age is not defined
    (ReferenceError)
    let age = 30;
    var salary = 10000;
  }
  console.log(salary); //10000 (accessible to due function
  scope)
  console.log(age); //error: age is not defined(due to block
  scope)
}

```

## ۱۹. دلیل انتخاب کلمه کلیدی let چیه؟

یه عنوان ریاضی هستش که توسط زبان‌های برنامه‌نویسی اولیه مثل Let و Scheme پذیرفته شده. این زبان از دهها زبان دیگه گرفته شده که از let به عنوان یه کلمه کلیدی سنتی تا حد ممکن نزدیک به var استفاده میکنه.

## ۲۰. چطوری می‌تونیم توی بلوک مربوط به switch بدون دریافت خطا متغیر تعریف کنیم؟

اگه بخوایم متغیرها رو مجدداً توی یه بلوک switch تعریف کنیم، این کار باعث خطا می‌شه چون در واقع فقط یه بلوک وجود داره. برای مثال توی کد زیر یه خطای نحوی ایجاد می‌شه:

```
let counter = 1;
switch (x) {
  case 0:
    let name;
    break;

  case 1:
    let name; // SyntaxError for redeclaration.
    break;
}
```

برای جلوگیری از این خطأ، می‌تونیم یه بلوک تودرتو داخل case ایجاد کنیم و یه محیط واژگانی دارای محدوده بلوک جدید ایجاد کنیم.

```
let counter = 1;
switch (x) {
  case 0: {
    let name;
    break;
  }
  case 1: {
    let name; // No SyntaxError for redeclaration.
    break;
  }
}
```

## ۲۱. چیه؟ Temporal-Dead-Zone

رفتاری توی جاواسکریپت که موقع تعریف متغیر با کلمات کلیدی let و const رخ میده، نه با کلمه کلیدی var. توی اکماسکریپت ۶، دستیابی به متغیر let و قبل از تعریفش (توی scope خودش) باعث خطای reference می‌شه. Temporal Dead Zone فاصله زمانی ایجاد اون، بین ایجاد اتصال متغیر و تعریف اون، منطقه هستش. بیاین با یه مثال ببینیم:

```
function somemethod() {
    console.log(counter1); // undefined
    console.log(counter2); // ReferenceError
    var counter1 = 1;
    let counter2 = 2;
}
```

## ۲۲. (توابع بلافصله صدا زده شده) چی هستن؟ IIFE

(فراخوانی عملکرد بلافصله) یه تابع جاواسکریپت که به محض تعریف اجرا می‌شه. تعریف اون به این صورته:

```
(function () {
    // logic here
})();
```

دلیل اصلی استفاده از IIFE بدست آوردن حریم خصوصی داده‌هاست، چون محیط خارجی به متغیرهایی که توی IIFE تعریف شده دسترسی نداره. برای مثال، اگه سعی کنیم با IIFE به متغیرها دسترسی پیدا کنیم این خطا رو می‌گیریم:

```
(function () {
    var message = "IIFE";
    console.log(message);
})();

console.log(message); //Error: message is not defined
```

## ۲۳. مزایای استفاده از module‌ها چیه؟

استفاده از مازول‌ها مزایای زیادی داره، مثلا:

۱. قابلیت نگهداری بالایی دارن
۲. قابلیت استفاده مجدد دارن
۳. نامگذاری بخش‌های مختلف کد رو راحت‌تر می‌کنن

## ۲۴. Memoization چیه؟

روش برنامه‌نویسی Memoization یه عملکرد(p‌رفورمنس performance) اون تابع رو افزایش بده. هر بار که یه تابع Memoize شده فراخوانی می‌شه، پارامترهای اون به همراه نتجه بدست اومنده cache می‌شه(یعنی توی حافظه ذخیره می‌شه). موقع اجرای بعدی قبل از انجام محاسبه بررسی می‌شه که داده قبل پردازش شده یا نه و اگه داده وجود داشته باشه، بدون اجرای کل تابع و از روی مقدار cache شده نتیجه برگشت داده می‌شه، در غیر این صورت تابع به شکل عادی اجرا می‌شه و بعدش نتیجه بدست اومنده توی حافظه ذخیره می‌شه.

بیاین یه مثال از نوشتن یه تابع با Memoization بزنیم،

```

const memoizAddition = () => {
  let cache = {};
  return (value) => {
    if (value in cache) {
      console.log("Fetching from cache");
      return cache[value];
    } else {
      console.log("Calculating result");
      let result = value + 20;
      cache[value] = result;
      return result;
    }
  };
};

// returned function from memoizAddition
const addition = memoizAddition();
console.log(addition(20)); //output: 40 calculated
console.log(addition(20)); //output: 40 cached

```

## چیه؟ Hoisting .۲۵

یه مکانیسم جاواسکریپتیه که متغیرها و تعاریف توابع رو به بالای scope یا دامنه خودشون انتقال میده. یادمون باشه که جاواسکریپت فقط تعریف متغیرها و توابع رو Hoist میکنه، نه مقدار دهی اولیه اونا رو.

بیاین یه مثال ساده از hoist کردن متغیرها ببینیم:

```

console.log(message); //output : undefined
var message = "The variable Has been hoisted";

```

ترجمه کد بالا اینطوری می‌شه:

```
var message;
console.log(message);
message = "The variable Has been hoisted";
```

## ۲۶. ES6 Class‌ها توی چیکار می‌کنن؟

در ES6، کلاس‌های جاواسکریپت اصطلاحاً فقط یه sugar-syntax روی وراثت مبتنی بر prototype جاواسکریپت هستن. برای مثال، وراثت مبتنی بر prototype که به شکل تابع به صورت زیر نوشته شده:

```
function Bike(model, color) {
  this.model = model;
  this.color = color;
}

Bike.prototype.getDetails = function () {
  return this.model + " bike has" + this.color + " color";
};
```

حالا همین کد با کلاس‌های ES6 و به شکل ساده‌تری به صورت زیر قابل نوشتن‌هه:

```
class Bike {
  constructor(color, model) {
    this.color = color;
    this.model = model;
  }

  getDetails() {
    return this.model + " bike has" + this.color + " color";
  }
}
```

## ۲۷. Closure‌ها چیا هستن؟

کلائزور ترکیبی از یه تابع و یه scope هستش که تابع توی اون تعریف شده. برای مثال این یه تابع داخلی‌هه که به متغیرهای تابع خارجی دسترسی داره. کلائزور دارای سه زنجیره scope هستش:

۱. scope داخلی و جایی که متغیرهای محلی بین برآکت‌های اون تعریف شده باشن
  ۲. متغیرهای تابع بیرونی
  ۳. متغیرهای عمومی و general
- بیاین یه مثال راجع به مفهوم کلژور بزنیم

```
function welcome(name) {
  var greetingInfo = function (message) {
    console.log(message + " " + name);
  };
  return greetingInfo;
}

var myFunction = welcome("John");
myFunction("Welcome "); //Output: Welcome John
myFunction("Hello Mr."); //output: Hello Mr.John
```

مطابق کد بالا، تابع داخلی (greetingInfo) حتی بعد از بازگشت تابع خارجی به متغیرهای scope تابع خارجی (welcome) دسترسی داره.

## ۲۸. **ماژول‌ها چیا هستن؟**

ماژول‌ها به واحدهای کوچیکی از کد مستقل و قابل استفاده مجدد گفته میشن و به عنوان پایه بسیاری از دیزاین پترن‌های جاواسکریپت عمل میکنن. خروجی ماژول‌های جاواسکریپت یه شی، یه تابع یا constructor هستش.

## ۲۹. **چرا به module‌ها نیاز داریم؟**

دلایل زیادی برای استفاده کردن از ماژول‌ها وجود داره که یه تعداد از اونا رو این زیر میاریم:

۱. قابلیت نگهداری
۲. قابلیت استفاده مجدد
۳. نام‌گذاری
۴. جدا بودن بخش‌های مختلف برنامه

## ۳۰. توی جاوااسکریپت scope چیه و چیکار میکنه؟

scope یا محدوده، به نحوه دسترسی متغیرها، توابع و اشیاء توی بخش‌های مختلف کدمون در زمان اجرا گفته میشه. به عبارت دیگه، دامنه قابلیت دیده شدن متغیرها و بقیه منابع رو تو قسمت‌هایی از کدمون تعیین میکنه.

## ۳۱. توی service-worker چیه؟

اساساً یه اسکریپت هستش که جدا از یه صفحه وب توی پس‌زمینه اجرا می‌شه و ویژگی‌هایی رو فراهم میکنه که نیازی به صفحه وب یا تعامل کاربر نداره. بعضی از ویژگی‌های عمدۀ service worker عبارتند از: تجربه کار با برنامه بدون نیاز به اینترنت (آفلاین وب)، به روزرسانی داده‌ها به شکل متنابع در پس‌زمینه، push notification، رهگیری و رسیدگی به درخواست‌های شبکه و مدیریت درخواست‌های cache شده.

## ۳۲. توی DOM چطوری می‌شه service-worker رو دستکاری کرد؟

مستقیماً نمیتونه به DOM دسترسی پیدا کنه، اما میتوانه با پاسخ به پیام‌های ارسالی از طریق رابط `postMessage` با صفحاتی که کنترل میکنه ارتباط برقرار کنه و این صفحات میتوانن DOM رو دستکاری کنن.

## ۳۳. چطوری می‌تونیم بین ریست شدن‌های service-worker داده‌های مورد نظرمون رو مجدد استفاده کنیم؟

مشکلی که توی service worker وجود داره اینه که در صورت عدم استفاده برای یه مدت، قطع میشه و در صورت نیاز بعدی دوباره راهاندازی می‌شه، به همین دلیل نمی‌تونیم به service worker state سراسری توی `handler`های `onmessage` و `onfetch` یه `onmessage` استفاده کنیم. تو این حالت service worker برای تداوم کار و استفاده مجدد موقع شروع مجدد، به indexedDB دسترسی خواهد داشت و میشه از اون استفاده کرد.

## ۳۴.IndexedDB چیه؟

IndexedDB یه API سطح پایین برای ذخیره client-side یا سمت کاربر و برای مقدار زیادی از داده‌های ساخت یافته (structured) شامل فایل‌ها و blob‌ها هستش. این API از index‌ها برای فراهم‌سازی جستجو با کارایی بالا توی این داده‌ها استفاده میکنه.

## ۳۵. Web-storage چیه؟

web storage یه API هستش که مکانیزمی رو فراهم میکنه که مرورگرها می‌تونن یه مقدار رو به همراه یه کلید و بصورت محلی توی مرورگر کاربر ذخیره کنن، که روشی بسیار مدرن‌تر و عملی‌تر نسبت به کوکی‌ها محسوب میشه. فضای ذخیره‌سازی وب دو مکانیزم برای ذخیره اطلاعات روی client فراهم میکنه.

۱. **Local storage**: داده‌ها رو برای مسیر (origin) فعلی و بدون تاریخ انقضا ذخیره میکنه.

۲. **Session storage**: داده‌ها رو برای یه session ذخیره میکنه و با بسته شدن تب مرورگر داده‌ها از بین میرن.

## ۳۶. Post-message چیه؟

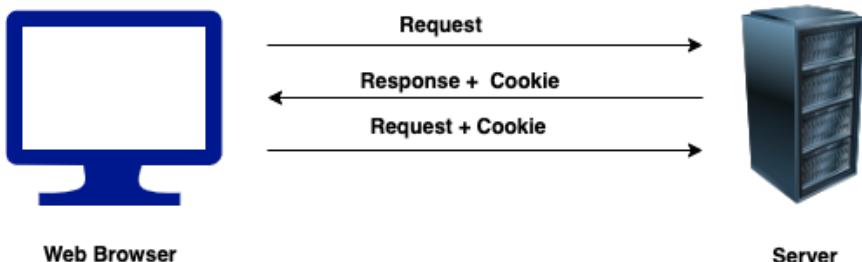
روشی هست که امکان ایجاد ارتباط متقابل بین آبجکت‌های window رو فراهم میکنه (برای مثال، بین یه صفحه و یه پنجره بازشو که باعث ایجاد اون شده، یا بین یه صفحه و یه iframe قرارداده شده توی اون) به طور کلی اسکریپت‌های موجود در صفحات مختلف اگه که صفحاتشون از خط و مشی یکسانی تبعیت کنن مجاز به دسترسی به همدیگه هستن. (یعنی صفحات از پروتکل، شماره پورت و host یکسانی برخوردار باشن).

## ۳۷. Cookie چیه؟

کوکی قطعه‌ای از داده هستش که توی کامپیوتمنون ذخیره می‌شه تا مرورگر به اون دسترسی داشته باشه. کوکی‌ها به عنوان جفت‌های کلید و مقدار ذخیره میشن.

برای مثال می‌توانیم یه کوکی با عنوان نام کاربری مثل زیر ایجاد کنیم:

```
document.cookie = "username=John";
```



## ۳۸. چرا به cookie نیاز داریم؟

از کوکی‌ها برای به خاطر سپردن اطلاعات مربوط به مشخصات کاربر (مانند نام کاربری) استفاده می‌شود. در اصل شامل دو مرحله هستند:

۱. وقتی که کاربر از یه صفحه وب بازدید میکنه، مشخصات کاربر میتونه توی یه کوکی ذخیره بشه.
۲. دفعه بعد که کاربر از صفحه بازدید کرد، کوکی مشخصات کاربر رو به خاطر میاره.

## ۳۹. گزینه‌های قابل تنظیم توی cookie چیا هستن؟

گزینه‌های زیر برای کوکی موجوده:

۱. به طور پیش فرض، کوکی موقع بسته شدن مرورگر حذف می‌شود اما با تنظیم تاریخ انقضا (به وقت UTC) می‌توانیم این رفتار را تغییر بدیم.

```
document.cookie = "username=John; expires=Sat, 8 Jun 2019  
12:00:00 UTC";
```

۲. به طور پیش فرض، کوکی به صفحه فعلی تعلق دارد. اما با استفاده از پارامتر path می‌توانیم به مرورگر بگیم که کوکی متعلق به چه مسیریه.

```
document.cookie = "username=John; path=/services";
```

## ۴۰. چطوری می‌شه یه cookie رو حذف کرد؟

با تنظیم تاریخ انقضا به عنوان تاریخ گذشته می‌تونیم کوکی رو حذف کنیم. تو این حالت نیازی به تعیین مقدار کوکی نیست.

برای مثال، می‌تونیم کوکی نام کاربری رو توی صفحه فعلی به صورت زیر حذف کنیم.

```
document.cookie =
"username=; expires=Fri, 07 Jun 2019 00:00:00 UTC; path=/";
```

**نکته** برای اطمینان از نحوه درست پاک کردن کوکی باید گزینه مسیر کوکی رو تعیین کنیم. بعضی از مرورگرها تا زمانی که پارامتر مسیر رو تعیین نکنیم اجازه حذف کوکی رو نمیدن.

## ۴۱. تفاوت‌های بین cookie و local-storage و session-storage چیا هستن؟

Session storage	Local storage	Cookie	ویژگی
client فقط	فقط client	client و server هردو	قابل دسترسی از server یا client طرف
تا وقتی که تب بسته نشده	تا وقتی که حذف بشه	پیکربندی شده با استفاده از گزینه اکسپایر	طول عمر
پشتیبانی نمی‌شه	پشتیبانی نمی‌شه	پشتیبانی می‌شه	پشتیبانی از SSL
5MB	5MB	4KB	حداکثر اندازه داده

## ۴۲. تفاوت‌های بین sessionStorage و localStorage چیا هستن؟

لوکال استوریج همون سشن استوریج هستش اما داده‌ها با بستن و دوباره باز کردن مرورگر همچنان حفظ می‌شه (تاریخ انقضا نداره) در حالی که سشن استوریج داده‌ها رو با بستن

پنجره مرورگر پاک میکنه.

## ۴۳. چطوری به web-storage دسترسی پیدا می‌کنی؟

آبجکت window ویژگی‌های WindowSessionStorage و WindowLocalStorage و ke دارای ویژگی‌های sessionStorage و localStorage هستن رو پشتیبانی میکنه. این خصوصیات نمونه ای از شئ Storage رو ایجاد میکنه که از طریق اون می‌شه موارد داده رو برای یه دامنه خاص و نوع ذخیره سازی (session یا محلی) تنظیم، بازیابی و حذف کرد.

برای مثال، می‌تونیم روی اشیای ذخیره سازی محلی مثل زیر بخونیم و بنویسیم:

```
localStorage.setItem("logo",
document.getElementById("logo").value);
localStorage.getItem("logo");
```

## ۴۴. چه متدهایی روی session-storage قابل استفاده هستن؟

متدهایی روی خواندن، نوشتن و پاکسازی داده‌های session storage ارائه میده. مثلا:

```
// Save data to sessionStorage
sessionStorage.setItem("key", "value");

// Get saved data from sessionStorage
let data = sessionStorage.getItem("key");

// Remove saved data from sessionStorage
sessionStorage.removeItem("key");

// Remove all saved data from sessionStorage
sessionStorage.clear();
```

## ۴۵. رخداد storage چیه و چطوری ازش استفاده می‌کنیم؟

رویدادی هستش که با همزمان با تغییر یه محل ذخیره‌سازی تو یه context صفحه دیگه‌ای فراخوانی می‌شه. این امکان بهمون قابلیت پردازش تغییرات مقادیر ذخیره‌سازی شده توسط یه EventHandler رو میده. ساختار کد اون یه چیزی مث کد زیر میشه:

```
window.onstorage = functionRef;
```

برای مثال استفاده از رویداد onstorage رو ببینیم که کلید ذخیره و مقادیر اونو ثبت میکنه:

```
window.onstorage = function (e) {
    console.log(
        "The " +
        e.key +
        " key has been changed from " +
        e.oldValue +
        " to " +
        e.newValue +
        ".");
};
```

## ۴۶. چرا به web-storage نیاز داریم؟

با استفاده از این قابلیت میشه گفت که فضای ذخیره‌سازی وب از امنیت بیشتری برخورداره و مقدار زیادی داده می‌تونن به صورت محلی ذخیره بشن، بدون اینکه روی عملکرد وب سایت تأثیر بذارن. همچنین، اطلاعات هرگز به سرور منتقل نمیشون و به همین دلیل این روش نسبت به کوکی‌ها بیشتر توصیه می‌شه.

## ۴۷. چطوری می‌تونیم پشتیبانی از web-storage توسط مرورگر رو بررسی کنیم؟

قبل از استفاده از فضای ذخیره‌سازی وب، می‌تونیم پشتیبانی مرورگر رو برای localStorage و sessionStorage بررسی کنیم. البته مرورگرهای مدرن امروزی کاملاً پشتیبانی-

رو ارائه میدن storage.

```
if (typeof Storage !== "undefined") {
    // Code for localStorage/sessionStorage.
} else {
    // Sorry! No Web Storage support..
}
```

## ۱۴۸. چطوری می‌تونیم پشتیبانی از web-worker توسط مرورگر رو برسی کنیم؟

می‌تونیم برای داشتن یه کد cross-functional قبل از استفاده از وب ورکرهای پشتیبانی مرورگر رو برسی کنیم، انجام این کار هم ساده اس و با یه شرط ساده محقق میشه:

```
if (typeof Worker !== "undefined") {
    // code for Web worker support.
} else {
    // Sorry! No Web Worker support..
}
```

## ۱۴۹. یه مثال از web-workerها می‌تونی بزنی؟

یه مثال ساده برای شروع استفاده از web worker می‌تونه مثال شمارنده باشه که برای اجرای اون باید مراحل زیر رو دنبال کنیم:

۱. ساخت یه فایل **Web Worker**: برای افزایش مقدار شمارشی، باید یه اسکریپت بنویسیم که این کار رو انجام میده. بیاین اسم مشو counter.js بذاریم

```
let i = 0;

function timedCount() {
    i = i + 1;
    postMessage(i);
    setTimeout("timedCount()", 500);
}

timedCount();
```

اینجا از روش `postMessage` برای ارسال پیام به صفحه HTML استفاده می‌شود.

۲. ایجاد شی **Web Worker**: با بررسی پشتیبانی مرورگر می‌توانیم یه شی Web Worker ایجاد کنیم. بیاین اسم این فایل رو `web_worker_example.js` بذاریم.

```
if (typeof w == "undefined") {
    w = new Worker("counter.js");
}
```

روی همین شی می‌توانیم پیام‌ها را از web worker دریافت کنیم:

```
w.onmessage = function (event) {
    document.getElementById("message").innerHTML = event.data;
};
```

۳. پایان دادن به **Web Worker**: می‌دونیم که **web Worker**ها تا زمان خاتمه یافتن پیام‌ها (حتی بعد از اتمام یه اسکریپت خارجی) به گوش دادن ادامه میدن. برای قطع کردن گوش دادن به پیام‌ها می‌توانیم از دستور `terminate` استفاده کنیم.

```
w.terminate();
```

۴. استفاده مجدد از **web worker**: اگه متغیر `worker` رو `undefined` مقداردهی کنیم، می‌توانیم از کد نوشته شده مجدد استفاده کنیم.

```
w = undefined;
```

## ۵. محدودیت‌های **DOM** روی **web-worker** چیا هستن؟

WebWorker‌ها به اشیا جاواسکریپت دسترسی ندارن چون توی یه فایل خارجی تعریف شدن. پس به این آبجکت‌ها دسترسی نداریم:

۱. آبجکت `Window`
۲. آبجکت `Document`
۳. آبجکت `Parent`

## ۵. چیه؟ Promise

یه آبجکت‌ایه که یه ممکنه یه callback برای موفق بودن یا رو با یه موفق نبودن(مثلاً خطای شبکه) فرآیند تولید کنه. پس کلا حالت‌های ممکن برای یه Promise یکی از این سه حالت خواهد بود: انجام شده، رد شده، یا در انتظار. برای ساختن Promise‌ها از سینتکس زیر استفاده می‌شه

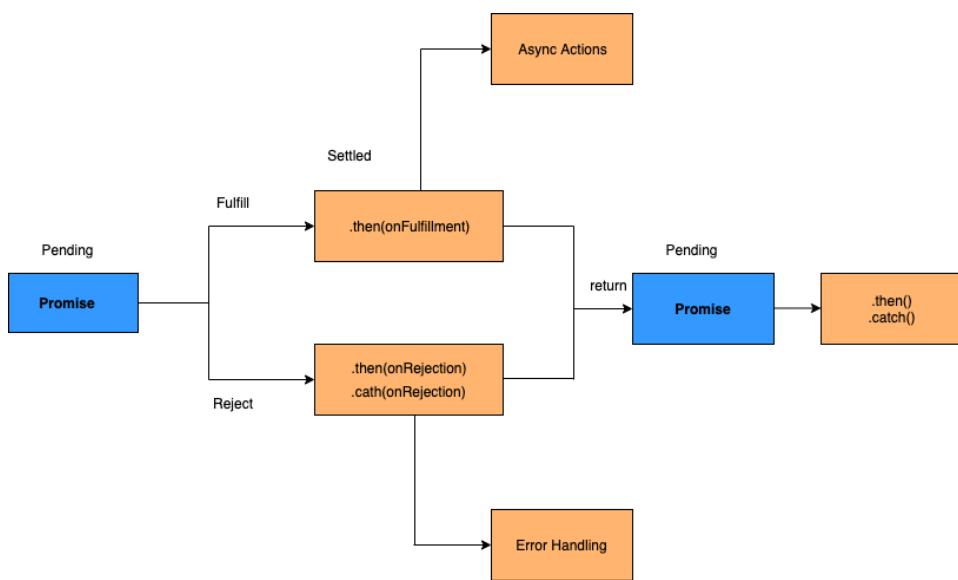
```
const promise = new Promise(function (resolve, reject) {
  // promise description
});
```

استفاده از Promise‌ها رو هم باهم ببینیم

```
const promise = new Promise(
  (resolve) => {
    setTimeout(() => {
      resolve("I'm a Promise!");
    }, 5000);
  },
  (reject) => {
    // if we want to reject, we can call `reject()` cb method
  }
);

/**
 * We can call promise and add some success callback
 functionality to run after resolve
 */
promise.then((value) => console.log(value)); // here after 5s
we will have a log in our console
```

چرخه فرآیند یه Promise به این شکل انجام می‌شه:



## ۵۲. چرا به Promise نیاز داریم؟

Promise‌ها برای رسیدگی به عملیات ناهمزمان (async) استفاده می‌شون. اونا با جلوگیری از وقوع callback hell و نوشتن کد clean و تر و تمیزتر، یه رویکرد جایگزین برای callback‌ها را ارائه می‌کنند.

## ۵۳. سه تا وضعیت ممکن برای یه Promise چیا هستن؟

های سه حالت دارن: Promise

۱. این حالت اولیه Promise قبیل از شروع عملیاته **Pending**.
۲. این حالت نشون میده که عملیات مشخص شده تکمیل شده **Fulfilled**.
۳. این حالت نشون میده که عملیات کامل نشده. تو این حالت یه مقدار خطای داده خواهد شد **Rejected**.

## ۵۴. توابع callback چی هستن؟

تابع تابعیه که به عنوان آرگومان به تابع دیگری منتقل می‌شه. این تابع در داخل تابع خارجی برای تکمیل یه عملیات فراخوانی می‌شه.  
بیاین یه مثال ساده از نحوه استفاده از تابع callback ببینیم:

```
function callbackFunction(name) {
  console.log("Hello " + name);
}

function outerFunction(callback) {
  let name = prompt("Please enter your name.");
  callback(name);
}

outerFunction(callbackFunction);
```

## ۵۵. چرا به توابع callback نیاز داریم؟

چون جاواسکریپت یه زبان ایونت محوره به callback ها نیاز داریم. این به این معنیه که جاواسکریپت به جای منظر موندن برای جواب یه عملیات فراخوانی شده، در حین گوش دادن به ایونت‌های دیگه به اجرا شدن ادامه میده.

بیاین یه مثال از دو تابع که پشت سرهم اجرا میشن ولی یکی‌شون callback میزنه به یه API (شبیه‌سازی شده با setTimeout) و اون یکی یه لاغ کنسول ساده می‌ندازه رو ببینیم.:

```

function firstFunction() {
    // Simulate a code delay
    setTimeout(function () {
        console.log("First function called");
    }, 1000);
}

function secondFunction() {
    console.log("Second function called");
}

firstFunction();
secondFunction();

// Output :
// Second function called
// First function called

```

همونطور که از خروجی می‌شه فهمید، جاواسکریپت منتظر جواب تابع اول نمونده و بلوک کد باقی مونده رو اجرا کرده. بنابراین از callbackها می‌تونیم به شکلی استفاده کنیم که مطمئن شیم یه کد تا زمانی که اجرای کد دیگه تمام نشده، اجرا نمی‌شه.

## ۵۶.Callback hell یا جهنم توابع Callback چیه؟

اصطلاحا یه آنتی‌پترن‌ه که با چندین callback تودرتو ایجاد میشه، همیشه هم خوندن کد و رفع خطاهای رو سختتر میکنه، مخصوصا زمان‌هایی که با async کار می‌کنیم. یه جهنم callback با یه کد مثل کد پایین ایجاد میشه:

```

async1(function () {
    async2(function () {
        async3(function () {
            async4(function () {
                // ....
            });
        });
    });
});

```

## ۵۷. چیه؟ SSE یا همون Server-sent-events

امکان رو میده که به روزرسانی‌های خودکار رو از طریق درخواست HTTP و بدون استفاده از polling(درخواست‌های پیوسته و منظم) دریافت کنه. این روش، یه کانال ارتباطی به طرفهست و event‌ها فقط از سرور به client ارسال میشون. این روزا از قابلیت‌های SSE روی روزرسانی‌های فیسبوک/توییتر، به روزرسانی قیمت سهام، فیدهای خبری و غیره استفاده میشه.

## ۵۸. چطوری می‌تونیم پیام‌های server-sent-event رو دریافت کنیم؟

کلاس EventSource برای دریافت پیام‌های event ارسال شده از سرور استفاده می‌شه. برای مثال، می‌تونیم پیام‌هایی که میخواهیم رو از سرور مثل مثال زیر بگیریم.

```
if (typeof EventSource !== "undefined") {
  var source = new EventSource("sse_generator.js");
  source.onmessage = function (event) {
    document.getElementById("output").innerHTML += event.data +
    "<br>";
  };
}
```

## ۵۹. چطوری می‌تونیم پشتیبانی مرورگر برای SSE رو بررسی کنیم؟

می‌تونیم قبل از استفاده از SSE مانند زیر، پشتیبانی مرورگر رو با یه شرط شبیه کد زیر بررسی کنیم:

```
if (typeof EventSource !== "undefined") {
  // Server-sent events supported. Let's have some code here!
} else {
  // No server-sent events supported
}
```

## ۶۰. کدوم توابع روی SSE وجود دارن؟

این پایین یه لیست از event‌های موجود برای SSE رو ذکر می‌کنیم:

توضیحات	ایونت
موقعی که اتصال به سرور باز می‌شه استفاده می‌شه	onopen
این event زمانی استفاده می‌شه که پیامی دریافت شه	onmessage
زمانی اتفاق می‌وقته که خطای رخ بدھ	onerror

## ۶۱. اصلی‌ترین قوانین Promise‌ها چیا هستن؟

میشه گفت اصلی‌ترین قانون‌های Promise‌ها ایناست:

۱. آبجکت‌ایه که متده «then» رو برای منتظر پاسخ درخواست موندن ارائه میکنه.
۲. یه Promise معلق ممکنه به حالت تحقق یافته یا رد شده تبدیل بشه.
۳. Promise تمام شده یا رد شده حل و فصل می‌شه و نباید به حالت دیگری تبدیل شه.
۴. پس از اتمام Promise مقدار اون نباید تغییر کنه.

## ۶۲. توى callback چطوری رخ میده؟

می‌تونیم یه پاسخ api-call رو داخل یه api-call دیگر قرار بدین تا عملیات‌ها رو به شکل متوالی و یکی یکی انجام بدین. این به عنوان callbacks در callback شناخته می‌شه.

```

loadScript('/script1.js', function(script) {

    console.log('first script is loaded');

    loadScript('/script2.js', function(script) {

        console.log('second script is loaded');

        loadScript("/script2.js", function (script) {
            console.log("second script is loaded");

            loadScript("/script3.js", function (script) {
                console.log("third script is loaded");
                // after all scripts are loaded
            });
        });
    });
});

```

## ۶۳. زنجیره‌ها چیه؟

فرآیند اجرای دنباله‌ای از عملیات‌های ناهمزمان (async) به طوریکه هر کدام بعد از تموم شدن قبلی اجرا بشن با استفاده از Promise‌ها به عنوان Promise chaining شناخته می‌شوند. بیان برای محاسبه نتیجه نهایی یه مقدار، یه مثال از زنجیره‌ها رو بینیم:

```

new Promise(function (resolve, reject) {
    setTimeout(() => resolve(1), 1000);
})
.then(function (result) {
    console.log(result); // 1
    return result * 2;
})
.then(function (result) {
    console.log(result); // 2
    return result * 3;
})
.then(function (result) {
    console.log(result); // 6
    return result * 4;
});

```

- توی بلوک‌های کد بالا، نتیجه هر Promise به زنجیره‌های then بعدی منتقل می‌شه، فرآیندشون به شکل زیر انجام می‌شه:
۱. پس از اول توی ۱ ثانیه حل می‌شه، Promise با چاپ مقدار ۱ توی کنسول، فراخوانی می‌شه و یه با مقدار result ضریب ۲ برمی‌گردونه.
  ۲. پس از اون then با چاپ مقدار ۲ برمی‌گردونه.
  ۳. پس از اون result به callback بعدی منتقل شده و با چاپ مقدار ۳ برمی‌گردوند.
  ۴. در نهایت مقدار به آخرین callback رسیده و با چاپ مقدار ۶ توی کنسول، یه result با Promise با نتیجه ۴ برمی‌گردونه.
  ۵. پس از اون «then» با ثبت نتیجه (۱) فراخوانی می‌شه و سپس یه Promise با مقدار نتیجه \* ۲ برمی‌گردونه.
  ۶. پس از اون مقدار به بعدی منتقل شد. سپس با ثبت نتیجه (۲) و برگرداندن یه Promise با نتیجه \* ۳ با نتیجه \* ۲ برمی‌گردونه.
  ۷. در نهایت مقدار به آخرین . سپس با ثبت نتیجه (۶) و یه Promise با نتیجه \* ۴ .`handler` می‌شه.

## ۶۴. کاربرد متدهای promise.all چیه؟

یه Promise.all که آرایه‌ای از Promises را به عنوان ورودی می‌گیره (یک تکرار) و زمانی می‌شه که همه Promises resolve شون، و اگه یکی از اونها رد شه reject می‌شه. برای مثال، تیکه کد زیر رو ببینیم:

```
Promise.all([Promise1, Promise2, Promise3])
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.log(`Error in promises ${error}`);
  });
}
```

**نکته:** ترتیب خروجی Promises طبق همون ترتیب آرایه ورودی ایجاد می‌شه.

## ۶۵. هدف از متد race روی Promise چیه؟

متد race از Promise یه آرایه ازPromise‌ها را می‌گیره و نتیجه اولین resolve‌ی ایPromise را return می‌کند. بیاین مثالی از متد race را در نظر بگیریم که تو اون reject شده را برمی‌گردونه. بیاین مثالی از متد race را در نظر بگیریم که تو اون resolve اول Promise2 می‌شود:

```
const promise1 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 500, "one");
});

const promise2 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 100, "two");
});

Promise.race([promise1, promise2]).then(function (value) {
  console.log(value); // "two" // Both promises will resolve,
but promise2 is faster
});
```

## ۶۶. حالت توی strict چی کار میکنه؟

یکی از قابلیت‌های ارائه شده توی ES5 هست که به ما این امکان را میده که یه برنامه یا یه تابع رو تو یه حالت اجرایی "سخت‌تر" قرار بدیم. به این ترتیب از انجام بعضی اقدامات جلوگیری میکنه و استثنای Exception‌های بیشتری را ایجاد میکنه. عبارت تحت اللفظی "usestrict" به مرورگر دستور میده تا از کد جاواسکریپت در حالت Strict استفاده کنه.

## ۶۷. چرا به حالت strict نیاز داریم؟

حالت سخت گیرانه (strict mode) برای نوشتن جاواسکریپت "امن" و حصول اطمینان از اطلاع از "بد" برای جلوگیری از خطاهای واقعی استفاده میشه. برای مثال، ایجاد تصادفی یه متغیر گلوبال رو با ایجاد یه Exception حذف میکنه و یا یه خط برای انتساب به یه ویژگی غیرقابل نوشتن، یه ویژگی فقط گیرنده، یه ویژگی غیرموجود، یه متغیر غیرموجود یا یه ویژگی غیر قابل نوشتن ایجاد میکنه.

## ۶۸. چطوری می‌تونیم حالت strict را فعال کنیم؟

حالت سخت با اضافه کردن رشته `use strict` به ابتدای scope اعلام می‌شود. پس می‌توانیم اونو به ابتدای یه اسکریپت یا یه تابع اضافه کنیم. اگه در ابتدای یه اسکریپت اعلام شد، دامنه عمومی داره:

```
"use strict";
x = 3.14; // This will cause an error because x is not declared
```

و اگه در داخل یه تابع اعلام کنیم محدوده محلی داره:

```
x = 3.14; // This will not cause an error.
myFunction();

function myFunction() {
    "use strict";
    y = 3.14; // This will cause an error
}
```

## ۶۹. هدف از عملگر نقیض دوتایی (!! ) چیه؟

علامت نقیض دوتایی یا نفی (!!) اینه که تضمین میکنه که نوع حاصل از عملیات یه مقدار یا `false` و تایپش بولینه. اگه بود (برای مثال `0`, `null`, `undefined` و غیره)، میشه و در غیر این صورت، درسته و نتیجه `true` خواهد بود. برای مثال، می‌توانیم نسخه IE رو با استفاده از عبارت زیر آزمایش کنیم.

```
let isIE8 = false;
isIE8 = !!navigator.userAgent.match(/MSIE 8.0/);
console.log(isIE8); // returns true or false
```

اگه از این عبارت استفاده نکنیم مقدار اصلی رو برمی‌گردونه.

```
console.log(navigator.userAgent.match(/MSIE 8.0/)); // returns
either an Array or null
```

**نکته:** `!!` یه اپراتور جدید نیست و فقط دوتا اپراتور `!` هست.

**نکته:** استفاده از `!!` معادل استفاده از `Boolean(var)` هست ولی `!!` سریعتره.

## ۷۰. هدف از عملگر **delete** چیه؟

کلمه کلیدی **delete** برای حذف ویژگی و همچنین مقدار اون استفاده می‌شه.

```
var user = { name: "John", age: 20 };
delete user.age;

console.log(user); // {name: "John"}
```

## ۷۱. عملگر **typeof** چیکار میکنه؟

برای بدست آوردن نوع متغیر جاوااسکریپت می‌تونیم از عملگر **typeof** استفاده کنیم و نوع یه متغیر یا یه عبارت رو به صورت یه رشته برمی‌گردونه.

```
typeof "John Abraham"; // Returns "string"
typeof (1 + 2); // Returns "number"
```

## ۷۲. چیه و چه زمانی **undefined** می‌گیریم؟

ویژگی **undefined** می‌گیریم که به یه متغیر مقداری اختصاص داده نشده یا اصلاً تعریف نشده‌ست. هم نوع مقدار تعریف نشده هم تعریف نشده.

```
var user; // Value is undefined, type is undefined
console.log(typeof user); //undefined
```

هر متغیری رو می‌شه با تنظیم مقدار روی **undefined** خالی کرد.

```
user = undefined;
```

## ۷۳. چیه **null**؟

مقدار **null** عدم وجود عمدى هر مقدار شى رو نشون میده. **null** یکی از مقادیر اولیه جاوااسکریپتیه و حواسمنون باید باشه که نوع مقدار **null** آبجکته.

با قرار دادن مقدار null هم می‌توانیم متغیر را خالی کنیم.

```
var user = null;
console.log(typeof user); //object
```

## ۷۴. تفاوت‌های بین null و undefined چیا هستن؟

تفاوت‌های اصلی بین null و undefined :

Undefined	Null
یه مقدار انتساب نیست که تو اون متغیری اعلام شده باشه اما هنوز مقداری به اون اختصاص داده نشده.	یه مقدار انتسابه که نشون میده متغیر به هیچ شیئی اشاره نمیکنه.
تایپ undefined همون تعریف نشده و undefined هستش	تایپ null آبجکته
مقدار undefined یه مقدار اولیه اس و زمانی استفاده میشه که به یه متغیر مقداری اختصاص داده نشده باشه.	مقدار null یه مقدار اولیه اس که نشون دهنده مرجع تهی، خالی یا غیر موجوده.
عدم وجود خود متغیر رو نشون میده	عدم وجود مقدار برای یه متغیر رو نشون میده
در حین انجام عملیات اولیه به NaN تبدیل میشه	در حین انجام عملیات اولیه به صفر (۰) تبدیل میشه

## ۷۵. متد eval چیه؟

تابع eval کد جاواسکریپت‌ای رو که به صورت رشته بهش پاس داده شده رو اجرا میکنه. رشته میتوانه یه عبارت جاواسکریپت، متغیر، دستور یا دنباله‌ای از عبارات باشه.

```
console.log(eval("1 + 2")); // 3
```

## ۷۶. تفاوت‌های بین **document** و **window** چیا هستن؟

Document	Window
فرزنده مستقیم شی window هستش و همچنین به عنوان مدل شیء document ((DOM))	عنصر ریشه در هر صفحه وب
می‌توانیم از طریق window.document یا window. <b>document</b> به اون دسترسی داشته باشیم.	به طور پیش فرض شی window به طور ضمنی در هر صفحه قرار داره
متدهایی مانند، getElementById، getElementByTagName، createElement و غیره رو فراهم می‌کنه	دارای متدهایی مانند، alert، confirm و ویژگی‌هایی مانند document، location

## ۷۷. توی جاوااسکریپت چطوری می‌توانیم به **history** دسترسی داشته باشیم؟

شی window.history حاوی تاریخچه مرورگره. با استفاده از متدهای `next` و `back` می‌توانیم URL‌های قبلی و بعدی رو در تاریخچه بارگذاری کنیم. مثلا:

```
function goBack() {
    window.history.back();
}
function goForward() {
    window.history.forward();
}
```

نکته: همچنین می‌توانیم بدون پیشوند window به آبجکت history دسترسی داشته باشیم.

## ۷۸. انواع داده‌های جاواسکریپت کدوما هستن؟

- ۱. Number
- ۲. String
- ۳. Boolean
- ۴. Object
- ۵. Undefined

## ۷۹. چیه و چیکار میکنه isNaN؟

تابع `isNaN` (Not-a-Number) برای تعیین اینه که آیا یه مقدار یه عدد واقعیه یا نه هست یا نه استفاده می‌شه. یعنی اگه مقدار برابر با `Nan` باشه، این تابع `true` برمی‌گردونه. در غیر این صورت `false` برمیگردد.

```
isNaN("Hello"); //true
isNaN("100"); //false
```

## ۸۰. تفاوت‌های بین undefined و undeclared چیا هستن؟

<code>undefined</code>	<code>undeclared</code>
این متغیرها در برنامه هست، اما هیچ مقداری نداره	این متغیرها تو یه برنامه وجود ندارن و تعریف نشدن
اگه سعی کنیم مقدار یه متغیر تعریف نشده رو بخوnim، یه مقدار تعریف نشده برگردونده می‌شه.	اگه سعی کنیم مقدار یه متغیر undeclared رو بخوnim، با خطای زمان اجرا مواجه می‌شیم

## ۱۱. کدوم متغیرها عمومی هستن؟

متغیرهای عمومی اونایی ان که در طول کد بدون هیچ محدوده ای در دسترسن. کلمه کلیدی var برای اعلام یه متغیر محلی استفاده می‌شه اما اگه اونو حذف کنیم تبدیل به متغیر عمومی می‌شه.

```
msg = "Hello"; // var is missing, it becomes global variable
```

## ۱۲. مشکلات متغیرهای عمومی چیا هستن؟

مشکل متغیرهای سراسری تضاد نام متغیرها با دامنه محلی و گلوباله. دیباگ و آزمایش کدی که به متغیرهای سراسری متکیه سخته.

## ۱۳. مقدار NaN چیه؟

ویژگی NaN یه ویژگی گلوباله که مقدار "Not-a-Number" رو نشون میده. یعنی نشون میده که یه مقدار یه متغیر واقعاً عددی نیست. استفاده از NaN تو برنامه‌ها خیلی نداره، اما می‌شه از اون به عنوان مقدار بازگشتی یه سری توابع و برای یه سری موارد کم استفاده کرد:

```
Math.sqrt(-1);
parseInt("Hello");
```

## ۱۴. هدف از تابع isFinite چیه؟

تابع isFinite برای تعیین اینکه آیا یه عدد یه عدد محدود و قانونیه استفاده می‌شه. اگه مقدار infinity، -infinity+ یا NaN (Not-a-Number) باشه false برمی‌گردونه، در غیر این صورت true رو برمی‌گردونه.

```
isFinite(Infinity); // false
isFinite(NaN); // false
isFinite(-Infinity); // false

isFinite(100); // true
```

## ۸۵. یه event-flow چیه؟

ترتبیه که `event` در صفحه وب دریافت می‌شه. وقتی روی یه المنت در صفحه وب کلیک می‌کنیم و این المنت به شکل تودرتو داخل المنتهای مختلف استفاده شده، قبل از اینکه کلیک‌مون واقعاً به مقصد یا المنت هدف برسه، باید `event` کلیک رو برای هر کدوم از عنصرهای والد خودش بفرسته و از بالا با شی پنجره گلوبال شروع شه. به شکل کلی **دو راه** برای جریان `event` وجود داره:

۱. از بالا به پایین(Event Capturing)

۲. از پایین به بالا(Event Bubbling)

۳. از بالا به پایین(Event Capturing)

۴. از پایین به بالا(Event Bubbling)

## ۸۶. چیه؟ Event-bubbling

نوعی انتشار `event` که تو اون `event` ابتدا روی درونی‌ترین عنصر هدف فراخوانی می‌شه و سپس به طور متوالی روی اجداد (والد) عنصر هدف تو همون سلسله مراتب تودرتو راه‌اندازی می‌شه تا زمانی که به بیرونی‌ترین عنصر DOM برسه.

## ۸۷. چیه؟ Event-capturing

نوعی انتشار `event` که تو اون `event` اول با بیرونی‌ترین عنصر ثبت می‌شه و سپس به طور متوالی بر روی `children` (children) عنصر هدف تو همون سلسله مراتب تودرتو راه‌اندازی می‌شه تا زمانی که به درونی‌ترین عنصر DOM برسه.

## ۸۸. چطوری می‌شه یه فرم رو با استفاده از جاواسکریپت ثبت کرد؟

می‌تونیم با استفاده از جاواسکریپت فرم مورد نظرمون رو با استفاده از کد `document.form[0].submit` ارسال کنیم. تمام اطلاعات ورودی فرم با استفاده از `onsubmit` event handler ارسال می‌شه

```
function submit() {
  document.form[0].submit();
}
```

## ۸۹. چطوری می‌شه به اطلاعات مربوط به سیستم عامل کاربر دسترسی داشت؟

شی window.navigator حاوی اطلاعاتی درباره جزئیات سیستم عامل مرورگر بازدیدکنده‌ست. بعضی از ویژگی‌های سیستم عامل روی ویژگی پلتفرم در دسترس هستند:

```
console.log(navigator.platform);
```

## ۹۰. تفاوت‌های بین رخدادهای `document-load` و `DOMContentLoaded` چیا هستن؟

رویداد DOMContentLoaded زمانی فراخوانی می‌شه که سند اولیه HTML به‌طور کامل بارگیری و تجزیه شده باشد، بدون اینکه منتظر بمونیم تا بارگیری assets (استایل‌ها، تصاویر و فریم‌های فرعی) تموم شه. در حالی که رویداد load روی داکیومنت زمانی فراخوانی می‌شه که کل صفحه بارگیری شه، شامل همه استایل‌ها، تصاویر و ... .

## ۹۱. تفاوت‌های بین `user object` ، `host object` و `native object` چیا هستن؟

آبجکت‌هایی هستن که به عنوان بخشی از زبان جاواسکریپت تعریف شدن و به عنوان بخشی از مشخصات ECMAScript هستن. برای مثال، اشیاء اصلی رشته، ریاضی، ECMAScript و غیره که در مشخصات RegExp، Object، Function تعریف شدن.

آبجکت‌هایی هستن که توسط مرورگر یا محیط زمان اجرا (Node) ارائه می‌شن. برای مثال، پنجره، XMLHttpRequest نودهای DOM و غیره به عنوان اشیاء میزبان در نظر گرفته می‌شن.

**User objects** آبجکت‌هایی هستند که تو کد جاواسکریپت تعریف شدن. برای مثال، آبجکت‌های ایجاد شده توسط برای اطلاعات پروفایل.

## ۹۲. کدام ابزار و تکنیک‌ها برای دیباگ کردن برنامه جاواسکریپتی استفاده می‌شون؟

۱. Chrome Devtools
۲. عبارت debugger
۳. متدهای console.log

## ۹۳. مزایا و معایب استفاده از **Promise**‌ها به جای **callback** چیا هستن؟

**مزایا:**

**مزایا:**

۱. از جهنم callback که قابل خواندن نیست جلوگیری می‌کنه.
۲. نوشتن کدهای ناهمزمان متوالی با then آسون‌تره.
۳. نوشتن کد ناهمزمان موازی با all آسون‌تره.
۴. بعضی از مشکلات رایج callback‌های بازگشتی رو حل می‌کنه (مشکل فراخوانی بسیار دیر، خیلی زود، یا چندبار فراخوانی callback و استثنایها رو مدیریتیش رو راحت‌تر می‌کنه).

**معایب:**

۱. کد يه کمی پیچیده میشه.
۲. اگه ES6 پشتیبانی نشد باید يه polyfill بارگذاری بشه.
۳. کد کمی پیچیده می‌سازد
۴. اگه ES6 پشتیبانی نمی‌شه، باید يه polyfill بارگذاری کنیم

## ۹۴. تفاوت‌های بین **DOM property** و **attribute** روی چیا هستن؟

DOM property ها برای نشونه گذاری HTML تعریف می‌شن در حالی که Attribute تعریف می‌شون برای مثال، عنصر HTML زیر دارای 2 ویژگی نوع و مقدار هستش

```
<input type="text" value="Name:>
```

می‌توانیم مقدار یه ویژگی رو به صورت زیر بدست بیاریم:

```
const input = document.querySelector("input");
console.log(input.getAttribute("value")); // Good morning
console.log(input.value); // Good morning
```

و بعد از اینکه مقدار فیلد متن به "Good evening" تغییر داده شد، نتیجه مثل زیر می‌شه:

```
console.log(input.getAttribute("value")); // Good morning
console.log(input.value); // Good evening
```

## ۹۵. سیاست same-origin چیه؟

سیاست یا خط مشی same-origin خط مشی که از درخواست جاوااسکریپت در روی کل domain جلوگیری می‌کنند. مبدأ به عنوان ترکیبی از شمای URI، نام میزبان(hostname) و شماره پورت(port) تعریف می‌شود. اگه این خطمشی رو فعال کنیم، مرورگر از دسترسی به اسکریپت مخرب تو یه صفحه به داده‌های حساس توی صفحه وب دیگه با استفاده از DOM(Document Object Model) جلوگیری می‌کند.

## ۹۶. هدف استفاده از 0 void چیه؟

Void(0) برای جلوگیری از به روزرسانی صفحه استفاده می‌شود. این متده برای از بین بردن ساید افکت‌های ناخواسته مفیده، چون مقدار اولیه تعریف نشده رو برمی‌گردونه. معمولاً برای اسناد HTML استفاده می‌شود که از (href="JavaScript:Void(0)") روی تو یه عنصر استفاده می‌کنند. یعنی وقتی روی یه لینک کلیک می‌کنیم مرورگر یه صفحه جدید رو بارگیری می‌کند یا همون صفحه رو تازه‌سازی(reload) می‌کند. ولی با استفاده از این عبارت می‌توانیم از این رفتار جلوگیری کنیم.

برای مثال، لینک زیر پیام رو بدون بارگیری مجدد صفحه مطلع می‌کند:

```
<a href="JavaScript:void(0);" onclick="alert('Well done!')">Click Me!</a>
```

## ۹۷. جاواسکریپت یه زبان تفسیری هست یا کامپایلری؟

جاواسکریپت یه زبان تفسیریه و نه یه زبان کامپایلری. یه مفسر توی مرورگر کد جاواسکریپت رو می خونه، هر خط رو تفسیر میکنه و اونو اجرا میکنه. امروزه مرورگرهای مدرن از فناوری موسوم به کامپایل JIT (Just-In-Time) استفاده میکنند که جاواسکریپت رو موقعی که در شرف اجراست به بایت کد اجرایی کامپایل میکنه.

## ۹۸. آیا جاواسکریپت یه زبان حساس به بزرگی و کوچکی (case-sensitive) است؟

بله، جاواسکریپت یه زبان حساس به حروف کوچک و بزرگه. کلمات کلیدی استفاده شده توی زبان، متغیرها، توابع و اشیا، و هر شناسه دیگر باید همیشه با حروف بزرگ تایپ شن.

## ۹۹. ارتباطی بین JavaScript و Java وجود داره؟

نه، این دو زبان برنامه نویسی کاملاً متفاوت هستن و هیچ ارتباطی با همدیگه ندارن. اما هر دوی اونا زبان‌های برنامه نویسی شی گرا هستن و مثل خیلی از زبان‌های دیگه، از syntax مشابهی برای ویژگی‌های اساسی (if، else، for، switch، break، continue) پیروی میکنند.

## ۱۰۰. Event‌ها چی هستن؟

رویدادها «چیزهایی» هستن که روی عناصر HTML و برای اونا اتفاق میافتن. موقعی که جاواسکریپت توی صفحات HTML استفاده میشه، میتونه به این رویدادها واکنش نشون بده و ما با استفاده از این رخدادها میتوانیم رفتار خاصی رو موقع رخداد خاص تعريف کنیم. بعضی از نمونه‌های رویدادها HTML عبارتند از:

۱. بارگذاری صفحه وب تموم شه
  ۲. فیلد ورودی تغییر کنه
  ۳. روی یه دکمه کلیک شه
- بیاین رفتار رویداد کلیک رو برای یه button بینیم:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function greeting() {
        alert("Hello! Good morning");
      }
    </script>
  </head>
  <body>
    <button type="button" onclick="greeting()">Click me</button>
  </body>
</html>
```

## ۱۰۱. کی جاواسکریپت رو ساخته؟

جاواسکریپت توسط Brendan Eich تو سال ۱۹۹۵ و موقع فعالیت ایشون توی نت اسکیپ (Netscape Communication) ایجاد شد و با نام Mocha توسعه پیدا کرد، اما بعدها زمانی که برای اولین بار در نسخه‌های بتا نت اسکیپ عرضه شد، این زبان به طور رسمی نامیده شد. LiveScript

## ۱۰۲. هدف از متد preventDefault چیه؟

متد preventDefault اگه رویداد قابل لغو باشه، اونو لغو میکنه، به این معنی که عمل یا رفتار پیش‌فرض متعلق به رویداد اتفاق نمی‌افته. برای مثال، جلوگیری از ارسال فرم موقع کلیک بر روی دکمه ارسال و جلوگیری از باز شدن URL صفحه موقع کلیک کردن روی لینک از موارد رایج استفاده‌شده.

```
document
  .getElementById("link")
  .addEventListener("click", function (event) {
    event.preventDefault();
});
```

**نکته:** همه رویدادها قابل لغو نیستن.

## ۱۰۳. کاربرد متدها stopPropagation چیه؟

روش `stopPropagation` برای جلوگیری از `event bubbling` توی یه زنجیره از رویدادها استفاده می‌شه. برای مثال، برای `div`‌های تودرتو زیر با متدها `stopPropagation` از انتشار پیش فرض رویداد موقع کلیک روی `DIV1` جلوگیری میکنه.

```
<p>Click DIV1 Element</p>
<div onclick="secondFunc()">
  DIV 2
  <div onclick="firstFunc(event)">DIV 1</div>
</div>

<script>
  function firstFunc(event) {
    alert("DIV 1");
    event.stopPropagation();
  }

  function secondFunc() {
    alert("DIV 2");
  }
</script>
```

## ۱۰۴. مراحلی که موقع استفاده از `return false` توی یه event-handler رخ میده چیا هستن؟

عبارت `return false` توی `event-handler` مراحل زیر رو انجام میده:

۱. ابتدا عملکرد یا رفتار پیش فرض مرورگر رو متوقف میکنه.

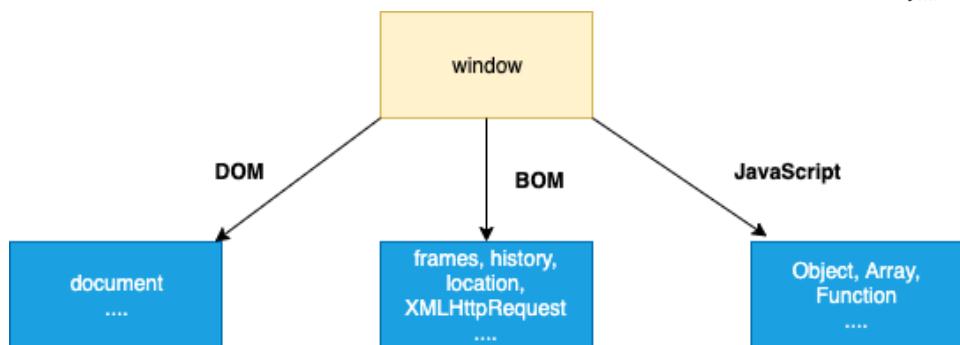
۲. رویداد از انتشار `DOM` جلوگیری میکنه.

۳. اجرای `callback` رو متوقف میکنه و بلاfacilte پس از فراخونی بر می‌گردد.

## ۱۰۵. BOM چیه؟

مدل آبجکتی مرورگر (BOM) به جاواسکریپت اجازه میده تا با مرورگر صحبت کنه. این مدل شامل `window`, `document`, `navigation`, `history`, `page`, `location`

هستن. BOM مدل استاندارد همه مرورگرها نیست و می‌توانه بر اساس مرورگرهای مختلف تغییر کنه.



## ۱۰۶. موارد استفاده از setTimeout کدوما هستن؟

متد `setTimeout` برای فراخونی یه تابع یا ارزیابی یه عبارت پس از میزان مشخصی از زمان(میلی ثانیه) استفاده می‌شه. برای مثال، بیاین یه پیام رو پس از 2 ثانیه با استفاده از متد `setTimeout` چاپ کنیم:

```
setTimeout(function () {
  console.log("Good morning");
}, 2000);
```

## ۱۰۷. موارد استفاده از setInterval کدوما هستن؟

متد `setInterval` برای فراخوانی یه تابع یا ارزیابی یه عبارت در بازه‌های زمانی مشخص (بر حسب میلی ثانیه) استفاده می‌شه. برای مثال، بیاین یه پیام رو هر 2 ثانیه با استفاده از متد `setInterval` چاپ کنیم:

```
setInterval(function () {
  console.log("Good morning");
}, 2000);
```

## ۱۰۸. چرا جاواسکریپت رو به عنوان یه زبان تک thread می‌شناسن؟

چون مشخصات زبان به برنامه نویس اجازه نمیده تا کدی بنویسه که مفسر ب-tone بخشهایی از اونو به صورت موازی در چندین Thread یا پردازش اجرا کنه. در حالی که زبان‌هایی مانند C، ++java، go، میتوان برنامه‌های چند رشته‌ای و چند فرآیندی بسازن.

## ۱۰۹. چیه؟ Event-delegation ؟

تکنیکی برای گوش دادن به رویدادهاس که تو اون یه عنصر والد رو به عنوان شنونده برای همه رویدادهایی که در داخلش اتفاق می‌افتن، تفویض می‌کنیم. برای مثال، اگه می‌خواین تغییرات فیلد رو تو یه فرم خاص تشخیص بدین، می‌تونیم از تکنیک Event-delegation استفاده کنیم.

```
var form = document.querySelector("#registration-form");

// Listen for changes to fields inside the form
form.addEventListener(
  "input",
  function (event) {

    // Log the field that was changed
    console.log(event.target);

  },
  false
);
```

## ۱۱۰. چیه؟ ECMAScript ؟

زبان برنامه نویسیه که اساس جاواسکریپت رو تشکیل میده. ECMAScript توسط سازمان استانداره بین المللی ECMA در مشخصات ECMA-262 و ECMA-402 استانداره شده است. اولین نسخه ECMAScript در سال ۱۹۹۷ منتشر شد.

## JSON چیه؟ .۱۱۱

(JSON) یه فرمت سبک هستش که برای تبادل داده‌ها استفاده می‌شه. اصول اولیه جیسون بر اساس زیرمجموعه‌ای از زبان جاواسکریپت و مشابه آبجکت اشیاییه که توی جاواسکریپت ساخته میشن.

## ۱۱۲. قوانین فرمت JSON کدوما هستن؟

۱. داده‌ها به صورت جفت نام/مقدار هستن
۲. داده‌ها با کاما از هم جدا میشن
۳. برآکتها اجسام رو نگه می‌دارن
۴. کروشه‌ها آرایه‌ها رو نگه می‌دارن

## ۱۱۳. هدف از متدهای JSON.stringify چیه؟

موقع ارسال داده‌ها به وب سرور، داده‌ها باید در قالب رشته‌ای باشن. می‌تونیم با استفاده از متدهای JSON.stringify روشی پنهان کنیم. مثل:

```
var userJSON = { name: "John", age: 31 };
var userString = JSON.stringify(user);
console.log(userString); //>{"name":"John", "age":31}"
```

## ۱۱۴. چطوری می‌تونیم یه رشته JSON رو تجزیه کنیم؟

موقع دریافت داده‌ها از یه وب سرور، داده‌ها همیشه در قالب رشته متنی هستن. اما می‌تونیم این مقدار رشته رو با استفاده از متدهای parse به یه آبجکت جاواسکریپت تبدیل کنیم.

```
var userString = '{"name":"John", "age":31}';
var userJSON = JSON.parse(userString);
console.log(userJSON); // {name: "John", age: 31}
```

## ۱۱۵. چرا به JSON نیاز داریم؟

موقع تبادل داده بین مرورگر و سرور، داده‌ها به دلیل تبادل شدن با استفاده از پروتکل Http فقط می‌توان متنی باشند. از اونجایی که JSON فقط متنی‌هه می‌شود اونو به راحتی به سرور ارسال کرد و از اون به عنوان قالب داده برای هر زبان برنامه‌نویسی استفاده کرد.

## ۱۱۶. PWA‌ها چی هستن؟

نوعی از برنامه‌های تلفن همراه هستن (Progressive web applications) که از طریق وب ارائه می‌شون، و با استفاده از فناوری‌های رایج وب از جمله HTML، CSS و جاواسکریپت ساخته می‌شون. PWA‌ها در سرورها قرار می‌گیرند و از طریق آدرس صفحه وب قابل دسترسی و نصب هستن.

## ۱۱۷. هدف از متده clearTimeout چیه؟

تابع clearTimeout در جاواسکریپت برای پاک کردن بازه زمانی استفاده می‌شود که قبل از اون توسط تابع setTimeout تنظیم شده. یعنی مقدار بازگشتی تابع setTimeout تو یه متغیر ذخیره می‌شود و برای پاک کردن تایمر به تابع clearTimeout پاس داده می‌شود. برای مثال، از تابع setTimeout موجود توی کد پایین برای نمایش پیام بعد از ۳ ثانیه استفاده می‌شود. این مهلت زمانی رو می‌شود با تابع clearTimeout پاک کرد.

```
<script>
var msg;

function greeting() {
    alert("Good morning");
}

function start() {
    msg = setTimeout(greeting, 3000);
}

function stop() {
    clearTimeout(msg);
}
</script>
```

## ۱۱۸. هدف از متدهای clearInterval چیه؟

تابع `clearInterval` تو جاواسکریپت برای پاک کردن `interval` که توسط تابع `setInterval` تنظیم شده استفاده می‌شود. برای مثال، مقدار بازگشتی که توسط تابع `setInterval` برمی‌گردد تو یه متغیر ذخیره می‌شود و برای پاک کردن `interval` ارسال می‌شود. برای مثال، از تابع `setInterval` توی کد پایینی برای نمایش پیام در هر ۳ ثانیه استفاده می‌شود. این بازه رو می‌شه با تابع `clearInterval` پاک کرد.

```
<script>
var msg;

function greeting() {
    alert("Good morning");
}

function start() {
    msg = setInterval(greeting, 3000);
}

function stop() {
    clearInterval(msg);
}
</script>
```

## ۱۱۹. توی جاواسکریپت، چطوری می‌شه به یه صفحه جدید redirect انجام داد؟

در `vanila` جاواسکریپت(جاواسکریپت خام یا خالص هم می‌گن)، می‌تونیم با استفاده از ویژگی `location` آبجکت گلوبال `window` به صفحه جدیدی هدایت بشین.

```
function redirect() {
    window.location.href = "newPage.html";
}
```

## ۱۲۰. چطوری بررسی می‌کنیم که یه `string` شامل یه `substring` هست یا نه؟

سه روش برای بررسی اینکه یه رشته دارای یه رشته فرعیه یا نه، وجود دارد.

۱. استفاده از متدهای `includes` و `String.prototype.includes`: ES6 روش `includes` برای آزمایش یه رشته حاوی یه رشته فرعی ارائه کرد.

```
var mainString = "hello",
    subString = "hell";
mainString.includes(subString);
```

۲. استفاده از متدهای `indexof` و `String.prototype.indexOf`: توى محیط ES5 یا قدیمی‌تر، می‌توانیم از استفاده کنیم که `index` یه رشته فرعی رو برمی‌گردونه. اگه مقدار برابر با ۱ نباشه، یعنی رشته فرعی توى رشته اصلی وجود دارد.

```
var mainString = "hello",
    subString = "hell";
mainString.indexOf(subString) !== -1;
```

۳. استفاده از `test`: **Regex** راه حل پیشرفته از روش `test` عبارت استفاده می‌کنیم، که امکان آزمایش در برابر عبارات منظم رو فراهم می‌کند.

```
var mainString = "hello",
    regex = /hell/;
regex.test(mainString);
```

## ۱۲۱. توى جاوااسکریپت، چطوری مقدار یه آدرس `email` رو اعتبارسنجی می‌کنیم؟

می‌توانیم با استفاده از `Regex` ایمیل رو توى جاوااسکریپت تأیید کنیم. توصیه می‌شود به جای سمت کلاینت، اعتبارسنجی سمت سرور انجام شود. چون جاوااسکریپت رو می‌شه سمت کلاینت غیرفعال کرد.

```

function validateEmail(email) {
  var re =
    /^(([^<>()\\.\,\;\:\s@"]+(\.\[^<>()\\.\,\;\:\s@"]+)*|(.+"))@((\[[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\])|(([a-zA-Z\-\-0-9]+\.)+[a-zA-Z]{2,}))$/;
  return re.test(String(email).toLowerCase());
}

```

بالا کاراکترهای یونیک رو می‌پذیرد.

## ۱۲۲. چطوری می‌توnim مقدار آدرس url جاری رو بخونیم؟

می‌توnim از عبارت `window.location.href` برای دریافت مسیر آدرس فعلی استفاده کنیم و می‌توnim از همون عبارت برای بهروزرسانی URL هم استفاده کنیم. همچنین می‌توnim از `document.URL` برای اهداف فقط خواندنی استفاده کنیم اما این راه حل مشکلاتی در FF دارد.

```
console.log("location.href", window.location.href); // Returns full URL
```

## ۱۲۳. ویژگی‌های مختلف url روی object history مربوط به کدوما هستن؟

برای دسترسی به اجزای URL صفحه می‌توان از ویژگی‌های شی `location` زیر استفاده کرد.

۱. URL - href
۲. URL - protocol
۳. URL - host
۴. URL - hostname
۵. URL - port
۶. URL - pathname
۷. URL - search
۸. URL - hash

## ۱۲۴. توی جاوااسکریپت چطوری می‌تونیم مقدار یه query-string رو بخونیم؟

می‌تونیم از `URLSearchParams` برای دریافت مقادیر رشته پرس و جو توی جاوااسکریپت استفاده کنیم. بیاین مثالی برای دریافت مقدار کد مشتری از رشته پرس و جو `URL` ببینیم:

```
const urlParams = new URLSearchParams(window.location.search);
const clientCode = urlParams.get("clientCode");
```

## ۱۲۵. چطوری می‌تونیم بررسی کنیم که آیا یه پرآپرتی روی آبجکت وجود داره یا نه؟

۱. استفاده از عملگرها: می‌تونیم از عملگر `in` استفاده کنیم که آیا کلیدی توی آبجکت وجود داره یا نه

```
"key" in obj;
```

برای بررسی وجود یا عدم وجود کلید، باید از پرانتز استفاده کنیم

```
!("key" in obj);
```

۲. استفاده از متدهای `hasOwnProperty`: می‌تونیم از `hasOwnProperty` برای آزمایش ویژگی‌های نمونه آبجکت (و نه ویژگی‌های ارشی) استفاده کنیم.

```
obj.hasOwnProperty("key"); // true
```

۳. استفاده از مقایسه `undifiend`: اگه بخوایم از یه آبجکت به یه ویژگی غیر موجود دسترسی پیدا کنیم، بهمون `undifiend` برمیگردونه. بیاین ویژگی‌ها را با مقایسه کنیم تا وجود ویژگی را مشخص کنیم.

```
const user = {
  name: "John",
};

console.log(user.name !== undefined); // true
console.log(user.nickName !== undefined); // false
```

## ۱۲۶. چطوری روی یه object حلقه میزني؟

می‌تونیم از حلقه `for-in` برای حلقه زدن آبجکت جاواسکریپت استفاده کنیم. همچنین می‌تونیم مطمئن شیم که کلیدی که دریافت می‌کنیم ویژگی واقعی یه آبجکته و با استفاده از روش `hasOwnProperty` نیست.

```
var object = {
  k1: "value1",
  k2: "value2",
  k3: "value3",
};

for (let key in object) {
  if (object.hasOwnProperty(key)) {
    console.log(key + " → " + object[key]); // k1 → value1
  ...
}
}
```

## ۱۲۷. چطوری تست می‌کنی که یه object خالیه؟

راه حل‌های مختلفی بر اساس نسخه‌های ECMAScript وجود داره  
۱. استفاده `+Object.entries(ECMA ۷)`:

می‌توانیم از `object.entries` استفاده کنیم و `length` او را چک کنیم

```
Object.entries(obj).length === 0 && obj.constructor === Object;
// Since date object length is 0, you need to check constructor
check as well
```

۲. استفاده از `+Object.keys(ECMA ۵)`:  
می‌توانیم از `object.keys` استفاده کنیم و `length` او را چک کنیم

```
Object.keys(obj).length === 0 && obj.constructor === Object; //
Since date object length is 0, you need to check constructor
check as well
```

۳. استفاده از `:hasOwnProperty (Pre-ECMA ۵)` با متدهای `for-in` استفاده کنیم و هر پارامتر رو با `hasOwnProperty` از حلقه `for-in` استفاده کنیم و چک کنیم

```

function isEmpty(obj) {
  for (var prop in obj) {
    if (obj.hasOwnProperty(prop)) {
      return false;
    }
  }

  return JSON.stringify(obj) === JSON.stringify({});
}

```

## ۱۲۸. arguments object چیه؟

آبجکت arguments یه آبجکت آرایه ماننده که داخل توابع قابل دسترسیه و حاوی مقادیر آرگومان‌های ارسال شده به اون تابعه. برای مثال، بیاین ببینیم چطوری از آبجکت arguments تابع sum استفاده کنیم:

```

function sum() {
  var total = 0;
  for (var i = 0, len = arguments.length; i < len; ++i) {
    total += arguments[i];
  }
  return total;
}

sum(1, 2, 3); // returns 6

```

**نکته:** ما نمی‌تونیم از متدهای ارایه برای آبجکت arguments استفاده کنیم اما می‌توانیم به ارایه تبدیلش کنیم

```
let argsArray = [...arguments];
```

## ۱۲۹. چطوری حرف اول یه رشته رو به حرف بزرگ تبدیل می‌کنی؟

می‌توانیم با درست کردن یه تابع و با استفاده از زنجیره ای از متدهای string مثل `charAt` و `slice` یه string با حرف اول بزرگ ایجاد کنیم و `toUpperCase`

```
function capitalizeFirstLetter(string) {
  return string.charAt(0).toUpperCase() + string.slice(1);
}
```

## ۱۳۰. مزایا و معایب حلقه for چیا هستن؟

حلقه `for` یه syntax تکرار رایجه که از مزایا و معایبیش میشه به موارد زیر اشاره کرد:

### مزایا

۱. توی همه‌ی محیط‌ها `env` کار میکنه
۲. می‌تونیم از `continue` و `break` برای کنترل جریان داده استفاده کنیم

### معایب

۱. از لحاظ نوشتاری کد بیشتری باید نوشته بشه
۲. Imperative هست
۳. ممکنه با خطاهای one-by-off رو برو بشیم

## ۱۳۱. چطوری تاریخ جاری رو توی جاواسکریپت نشون میدی؟

می‌تونیم از کلاس `new Date()` استفاده کنیم و با اجرای متده `toLocaleString()` تاریخ و زمان جدا شده توسط کاما رو بدست بیاریم :

```
const today = new Date().toLocaleString();
console.log(today.split(",")[0]); // 5/19/2022
```

## ۱۳۲. چطوری دو تا `date object` رو با هم مقایسه می‌کنی؟

برای این مقایسه نباید از اپراتورها استفاده کنیم. می‌تونیم مثل کد زیر از متده `getTime()` بر روی آجکت `Date` قرار داره استفاده کنیم

```
var d1 = new Date();
var d2 = new Date(d1);
console.log(d1.getTime() === d2.getTime()); //True
console.log(d1 === d2); // False
```

## ۱۳۳. چطوری بررسی می‌کنی که یه رشته با یه رشته دیگه شروع می‌شه؟

می‌تونیم از متدهای `startsWith` که بر روی پروتوتایپ `String` وجود داره استفاده کنیم که یه رشته رو می‌گیره و چک می‌کنه که رشته مورد نظر با اون رشته شروع می‌شه یا نه، برای مثال کد زیر رو براش مینویسیم:

```
"Good morning".startsWith("Good"); // true
"Good morning".startsWith("morning"); // false
```

## ۱۳۴. چطوری یه رشته رو `trim` می‌کنی؟

جاواسکریپت یه متدهایی می‌ده به اسم `trim` که روی رشته‌ها قرار داره با استفاده از این متدهای فضاهای خالی بین اون رشته برداشته می‌شه

```
"Hello World      ".trim(); //Hello World
```

## ۱۳۵. توی جاواسکریپت چطوری می‌تونیم یه زوج مرتب از `key` و `value` بسازیم؟

برای اضافه کردن `key` جدید به آبجکتها دو روش وجود داره

```
var object = {
  key1: value1,
  key2: value2,
};
```

۱. استفاده از `dot`: این روش زمانی موثر هستش که اسم پرآپرتی رو میدونیم

```
object.key3 = "value3";
```

۱. استفاده از کروشه[]: این روش زمانی موثر هستش که اسم پراپرتی داینامیک باشد

```
obj ["key3"] = "value3";
```

## ۱۳۶. آیا عبارت '--!' عملگر خاصی هست؟

نه! اپراتور خاص نیست اما ترکیب دو تا اپراتور استاندار هستش یکی بعد اون یکی

۱. اپراتور نقیض (!)

۲. کاهش کننده (-)

اول یه شماره از مقدار متغیر به مثال کم می‌شه بعد تست می‌شه که مساوی صفر هستش یا نه، که مشخص کننده درست یا غلط بودن شرط هست

## ۱۳۷. چطوری می‌تونیم به متغیرهای مقادیر اولیه بدم؟

می‌تونیم از عملگر یا اپراتور || برای تعریف یه مقدار پیش‌فرض استفاده کرد:

```
var a = b || c;
```

مثال تعریف شده بالا مقدار متغیر a زمانی برابر مقدار متغیر c خواهد شد که b خالی یا undefined باشد.

## ۱۳۸. چطوری می‌تونیم متن‌های چند خطی درست کنیم؟

می‌تونیم از / برای تعریف کردن رشته‌های چند خطی استفاده کنیم برای مثال:

```
const str =
  "This is a \
very lengthy \
sentence!";
```

اما اگه یه فاصله بعد / داشته باشیم، کد دقیقاً به همون حالتی که هست نشون داده می‌شه اما یه ارور خطای نوشتاری کد قراره داشته باشیم

روش بعدی استفاده کردن از `backtick` هست که وقتی موقع تعریف رشته به جای کوتیشن مارک ازش استفاده بشه می‌توانیم راحت‌تر بشه چند خطی تعریف کنیم. برای مثال می‌شه کد زیر رو براش نوشت:

```
const str = `This is a
very lengthy
sentence!`;
```

## ۱۳۹. مدل app-shell چیه؟

(shell) یکی از راههای ساخت PWA هستش که به طور قابل اعتماد و فوری بر روی صفحه نمایش کاربران شما بارگیری می‌شه، مشابه اونی‌که توی برنامه‌های کاربردی native به کاربر نشون داده می‌شه. برای رسوندن سریع HTML اولیه به صفحه بدون نیاز به شبکه مفیده.

## ۱۴۰. چطوری می‌تونیم روی یه تابع اضافه کنیم؟

می‌تونیم برای توابع پرآپرتی تعیین کنیم چون توابع اصولاً آبجکت هستن.

```
const fn = function (x) {
    //Function code goes here
};

fn.userName = "John";

fn.profile = function (y) {
    //Profile code goes here
};
```

## ۱۴۱. چطوری می‌تونیم تعداد پارامترهای ورودی یه تابع رو به دست بیاریم؟

با استفاده کردن از `function.length` می‌تونیم به تعداد پارامترهایی که یه تابع انتظار داره بگیره دسترسی داشته باشیم.  
بریم یه مثال درموردش ببینیم:

```
function sum(num1, num2, num3, num4) {
    return num1 + num2 + num3 + num4;
}
sum.length; // 4 is the number of parameters expected.
```

## ۱۴۲. چیه؟ Polyfill.

یه قسمت از کد جاواسکریپتی که با استفاده از اون ما می‌تونیم توابع پیشترفته رو روی مروگرهایی که به طور طبیعی پشتیبانی نمیکنن استفاده کنیم. پلاگین `IE7` که برای تقلید کردن توابع بر روی `canvas` یا مروگر `Silverlight` استفاده کرد

## ۱۴۳. عبارات `continue` و `Break` چی هستن؟

دستور `break` برای "پرش به بیرون" از یه حلقه استفاده می‌شه. یعنی حلقه رو می‌شکنه و اجرای کد رو بعد از حلقه ادامه میده.

```
for (i = 0; i < 10; i++) {
    if (i === 5) {
        break;
    }
    text += "Number: " + i + "<br>";
}
```

دستور `continue` برای "پرش از روی" یه تکرار در حلقه استفاده می‌شه. یعنی یه تکرار (در حلقه) رو می‌شکنه، اگه شرایط مشخصی رخ بده، و با تکرار بعدی در حلقه ادامه میده.

```

for (i = 0; i < 10; i++) {
  if (i === 5) {
    continue;
  }
  text += "Number: " + i + "<br>";
}

```

## ۱۴۴. توی جاواسکریپت labelها چیکار می‌کنن؟

دستور `label` به ما اجازه میده تا حلقه‌ها و بلوک‌ها رو توی جاواسکریپت نام‌گذاری کنیم. بعد می‌تونیم از این برچسب‌ها برای مراجعه به کد استفاده کنیم. برای مثال، کد زیر با استفاده از برچسب‌ها از چاپ اعداد وقتی که یکسان هستن، جلوگیری می‌کنه.

```

loop1: for (let i = 0; i < 3; i++) {
  loop2: for (let j = 0; j < 3; j++) {
    if (i === j) {
      continue loop1;
    }
    console.log("i = " + i + ", j = " + j);
  }
}

// Output is:
//   "i = 1, j = 0"
//   "i = 2, j = 0"
//   "i = 2, j = 1"

```

## ۱۴۵. مزایای declare کردن متغیرها در اوایل کد چیه؟

توصیه می‌شه که تمام تعریف متغیرها رو بالای هر اسکریپت یا تابع انجام بدیم. مزیت این کار:

۱. کد ما تمیز تر می‌شه
۲. یه مکان واحد برای جستجوی متغیرهای محلی فراهم می‌کنه
۳. می‌شه راحت از استفاده متغیرهای ناخواسته جلوگیری کرد
۴. این کار محاسبات ناخواسته رو کمتر می‌کند

## ۱۴۶. مزایای مقداردهی اولیه متغیرها چیه؟

- توضیه می‌شه که حتماً یه مقدار اولیه برای متغیرها تعیین بشه که دلایلشو چک می‌کنیم
۱. خروجی‌مون کد تمیز تری می‌شه
  ۲. این کار باعث می‌شه یه جا برای این متغیر رزرو بشه
  ۳. از برگشتن خطای `undefined` جلوگیری می‌شه

## ۱۴۷. روش توصیه شده برای ایجاد `object` چیه؟

برای ساخت یه `object` با مقادیر پیش‌فرض می‌توانیم به صورت زیر عمل کنیم

۱. استفاده از {} به جای `new Object`
۲. استفاده از "" به جای `new String`
۳. استفاده از 0 به جای `new Number`
۴. استفاده از false به جای `new Boolean`
۵. استفاده از [] به جای `new Array`
۶. استفاده از /() / به جای `new RegExp`
۷. استفاده از () {} به جای `new Function`

بریم چند تا مثال ببینیم:

```
const v1 = {};
const v2 = "";
const v3 = 0;
const v4 = false;
const v5 = [];
const v6 = /()/;
const v7 = function () {};
```

## ۱۴۸. چطوری می‌تونیم آرایه JSON تعریف کنیم؟

برای تعریف آرایه‌های JSON از برآکت استفاده می‌کنیم و هر تعداد که آبجکت خواستیم داخلش تعریف می‌کنیم.

بیاین یه مثال در موردش ببینیم

```
"users": [
    {"firstName":"John", "lastName":"Abrahm"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Shane", "lastName":"Warn"}
]
```

## ۱۴۹. چطوری می‌تونیم اعداد تصادفی تولید کنیم؟

می‌تونیم از متدهای `Math.random()` برای ساخت یه عدد رندوم بین ۰ تا ۱ و از متدهای `Math.floor()` برای رند کردن اون عدد استفاده کنیم حالا اگه حاصل عدد به دست اومده رو ضربدر ده کنیم عددی بین یک تا ده خواهیم داشت.

```
Math.floor(Math.random() * 10) + 1; // returns a random integer from 1 to 10
Math.floor(Math.random() * 100) + 1; // returns a random integer from 1 to 100
```

**نکته:** `Math.random()` یه عدد تصادفی بین ۰ تا ۱ ایجاد می‌کنه، یادمون باشه که استفاده از `Math.floor()` قبل ضرب کردن عدد رندوم به دست اومده در ده به عنوان کمترین مقدار، باعث می‌شه که خروجیمون همیشه به صفر رند بشه.

## ۱۵۰. می‌تونی یه تابع تولید اعداد تصادفی توی یه بازه مشخص بنویسی؟

بله ما می‌تونیم کد زیر رو برای این تابع داشته باشیم که مقادیر حداقل و حداقل رو بگیره و برای ما عدد رندوم ایجاد کنه:

```
function randomInteger(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
randomInteger(1, 100); // returns a random integer from 1 to 100
randomInteger(1, 1000); // returns a random integer from 1 to 1000
```

## ۱۵۱. Tree-shaking چیه؟

نوعی حذف کد مرده هستش و به این معنیه که مازول‌های استفاده نشده در طول فرآیند ساخت در بسته گنجونده نمی‌شن و برای اون بر ساختار استاتیک ES2015 rollup (یعنی import و export) توشی باشد. مازول ES2015 از این عملکرد استفاده شده.

## ۱۵۲. دلایل نیاز به tree-shaking کدوما هستن؟

می‌تونه اندازه کد رو در هر برنامه ای به میزان قابل توجهی کاهش بده. یعنی هرچی کد کمتری از طریق سیم بفرستیم برنامه کاربردی تره، به عنوان مثال، اگه فقط بخواهیم یه برنامه Hello World با استفاده از چارچوب‌های SPA ایجاد کنیم، حدود چند مگابایت حافظه رو اشغال می‌کنه، اما می‌تونه اندازه رو به چند صد کیلوبایت کاهش بده. تو باندلرهای Rollup و Webpack پیاده سازی شده.

## ۱۵۳. آیا استفاده از eval توصیه می‌شه؟

نه، eval اجازه اجرای کد دلخواه رو میده که باعث ایجاد مشکل امنیتی می‌شه. همونطور که میدونیم از تابع eval برای اجرای متن به عنوان کد استفاده می‌شه. در بیشتر موارد استفاده از اون ضروری نیست.

## ۱۵۴. Regular-Expression چیه؟

یه همون Regex یا regular expression به توالیه که یه ساختار جستجو ایجاد می‌کنه با استفاده از این ساختار ما می‌تونیم دیتامون رو جستجو کنیم و به قولی دیتامون رو اعتبارسنجی کنیم.

```
/pattern/modifiers;
```

برای مثال Regex حساس به حروف کوچک و بزرگ زبان انگلیسی به صورت ریر نوشته می‌شه:

```
/John/i
```

## ۱۵۵. متدهای رشته که روی Regular-expression مجاز هستن کدام است؟

متدهای `replace` و `search` برای رشته‌ها دارند. `replace` دو تا متد برای این رشته‌ها است: `i` و `g`. `search` یه عبارت رو می‌گیره اونو جستجو می‌کنه و محل اون عبارت رو برمی‌گردونه:

```
var msg = "Hello John";
var n = msg.search(/John/i); // 6
```

متدهای `replace` برای برگرداندن رشته اصلاح شده تو جایی که الگو جایگزین می‌شود، استفاده می‌شود:

```
var msg = "Hello John";
var n = msg.replace(/John/i, "Buttler"); // Hello Buttler
```

## ۱۵۶. توی Regex بخش modifiers چیکار می‌کنه؟

متدهای `replace` و `search` زمانی استفاده بشون که به جستجوهای بدون حروف کوچک و بزرگ سراسری نیاز داریم بیاین یه مثال درموردشون ببینیم:

توضیح	اصلاح کننده
تطبیق حساس به حروف	i
تطبیق کلی به جای توقف در اولین تشابه	g
تطبیق چندخطی	m

برای یه مثال از `modifier` گلوبال ببینیم:

```
const text = "Learn JS one by one";
const pattern = /one/g;
const result = text.match(pattern); // one,one
```

## ۱۵۷. پترن‌های regular-expression چیه؟

Regex یه گروهی از ساختارها برامون اماده کرده که با اونا کاراکترها رو چک کنیم اونا تو سه مدل طبفه بندی میشن.

۱. **براکتها:** برای پیدا کردن رنجی از کاراکتر استفاده میشن

برای مثال پایین چن تا مورد استفاده لیست شدن

۱. [abc]: برای پیدا کردن هر کاراکتری بین این سه کاراکتر استفاده میشه

۲. [0-9]: برای پیدا کردن ارقام بین این دو عدد استفاده میشه

۳. (a|b): برای پیدا کردن هر یه از گزینه‌های جدا شده با | استفاده میشه

۲. **کاراکتر برابر با:** این عبارت‌ها کاراکترهایی با معنی خاص هستن

برای مثال پایین سه تا مورد که استفاده میشه ازشون رو ببینیم

۱. \d: برای پیدا کردن اعداد استفاده میشه

۲. \\$\wedge: برای پیدا کردن فاصله‌ها استفاده میشه

۳. \b: برای پیدا کردن کاراکترهای همخوانی داشته با شروع شدن یا

پایانشون استفاده میشه

۳. **کمیت کننده‌ها:** این‌ها برای تعریف کمیت‌ها موثر هستن

برای مثال پایین دو تا مورد استفاده برآشون اوردیم

۱. +\n: برای پیدا کردن رشته همخوانی داشته با حداقل یه کاراکتر

۲. \n\*: برای پیدا کردن همخوانی هر رشته شامل صفر یا بیشتر

۳. ?: برای پیدا کردن هر رشته که شامل صفر یا یه کاراکتر میشه

## ۱۵۸. آبجکت RegExp چیه؟

Regex های object یه عبارت معمولی با پراپرتی‌ها و متدهای تعریف شده از قبل هس. بریم یه مثال از نحوه استفادشون ببینیم.

```
var regexp = new RegExp('^\w+');
console.log(regexp);
// expected output: /\w+/
```

## ۱۵۹. چطوری روی یه رشته دنبال یه پترن RegExp می‌گردی؟

می‌تونیم از متده استفاده برای جستجوی یه رشته برای الگو استفاده کنیم که بسته به نتیجه، `false` یا `true` را برمی‌گردانه.

```
var pattern = /you/;
console.log(pattern.test("How are you?")); //true
```

## ۱۶۰. هدف از متده exec چیه؟

هدف متده `exec` شبیه به روش `test` است که جستجوی یه تطابق تو یه رشته مشخص رو انجام میده و یه آرایه نتیجه یا `null` رو به جای برگرداندن `true/false` برمی‌گردانه.

```
var pattern = /you/;
console.log(pattern.exec("How are you?")); //["you", index: 8,
input: "How are you?", groups: undefined]
```

## ۱۶۱. چطوری استایلهای یه المنت HTML رو تغییر میدی؟

می‌تونیم سبک درون خطی یا اسم کلاس یه عنصر HTML رو با استفاده از جاوااسکریپت تغییر بدین

۱. استفاده از پرپرتبی `style`: با استفاده از ویژگی `style` می‌تونیم استایل درون خطی رو تغییر بدین

```
document.getElementById("title").style.fontSize = "30px";
```

۲. استفاده از پرپرتبی `className`: تغییر کلاس عنصر با استفاده از ویژگی `className` آسان است

```
document.getElementById("title").style.className = "custom-
title";
```

## ۱۶۲. نتیجه عبارت $1+2+3' چی می‌شه؟$

خروجی '33' می‌شه. از اونجایی که «1» و «2» مقادیر عددی هستن، نتیجه دو رقم اول به مقدار عددی «3» خواهد بود. رقم بعدی یه مقدار نوع رشته اس چون افزودن مقدار عددی «3» و مقدار رشته «3» فقط یه مقدار الحقی «33» می‌شه.

## ۱۶۳. عبارت **debugger** چیکار میکنه؟

دستور **debugger** هر گونه عملکرد اشکال زدایی موجود رو فراخوانی میکنه، مانند تعیین **breakpoint**. اگه هیچ عملکرد اشکال زدایی در دسترس نباشه، این عبارت تاثیری نداره. برای مثال، در تابع زیر یه دستور **debugger** درج شده. بنابراین اجرا تو دستور **debugger** مثل یه **breakpoint** در منبع اسکریپت متوقف می‌شه.

```
function getProfile() {
  // code goes here
debugger;
  // code goes here
}
```

## ۱۶۴. هدف از **breakpoint** توى **debugging** چیه؟

پس از اجرای دستور **debugger** و باز شدن پنجره دیباگر، می‌تونیم **breakpoint**ها رو در کد جاوا‌سکریپت تنظیم کنیم. در هر **breakpoint**، جاوا‌سکریپت اجرا نمی‌شه و به ما اجازه میده مقادیر جاوا‌سکریپت رو بررسی کنیم. پس از بررسی مقادیر، می‌تونیم با استفاده از دکمه پخش، اجرای کد رو ادامه بدیم.

## ۱۶۵. آیا می‌تونیم از عبارت‌های رزرو شده در تعریف **identifier**ها(اسم متغیر، کلاس و ...) استفاده کنیم؟

نه، ما نمی‌تونیم از کلمات رزرو شده به عنوان متغیر، برچسب، اسم آبجکت یا تابع استفاده کنیم. بیاین یه مثال ساده رو ببینیم:

```
let else = "hello"; // Uncaught SyntaxError: Unexpected token  
else
```

## ۱۶۶. چطوری تشخیص بدیم که یه مرورگر mobile هست یا نه؟

ما می‌تونیم با استفاده از Regex که یه boolean به ما برمی‌گردونه بفهمیم که مرورگری که کاربر داره ازش استفاده می‌کنه موبایل هست یا نه، کد تشخیص به این صورت نوشته می‌شه:

```

window.mobilecheck = function() {
    var mobileCheck = false;
    (function(a)
    {if(/(android|bb\d+|meego).+mobile|avantgo|bada\/*|blackberry|blaz
     |maemo|midp|mmp|mobile.+firefox|netfront|opera m(ob|in)i|palm(
os)?
|phone|p(ixi|re)\/*|plucker|pocket|psp|series(4|6)0|symbian|treo|u|
(browser|link)|vodafone|wap|windows
ce|xda|xiino/i.test(a) |||1207|6310|6590|3gso|4thp|50[1-
6]i|770s|802s|a
wa|abac|ac(er|oo|s\-)|ai(ko|rn)|al(av|ca|co)|amo|an(ex|ny|yw)|ap
m|r|s
)|avan|be(ck|ll|nq)|bi(lb|rd)|bl(ac|az)|br(e|v)w|bumb|bw\-
(n|u)|c55\/*|capi|ccwa|cdm\-*|cell|ctm|cldc|cmd\-
|co(mp|nd)|craw|da(it|ll|ng)|dbte|dc\-
s|devi|dica|dmob|do(c|p)o|ds(12|\-
d)|el(49|ai)|em(l2|ul)|er(ic|k0)|esl8|ez([4-
7]0|os|wa|ze)|fetc|fly(\-|_)|g1 u|g560|gene|gf\-*|g\-
mo|go(\.w|od)|gr(ad|un)|haie|hcit|hd\-(m|p|t)|hei\-
|hi(pt|ta)|hp( i|ip)|hs\-*c|ht(c(\-|
_|a|g|p|s|t)|tp)|hu(aw|tc)|i\-(20|go|ma)|i230|iac( |\-|
|\/*)|ibro|idea|ig01|ikom|im1k|inno|ipaq|iris|ja(t|v)a|jbro|jemu|j
| |\/*)|klon|kpt |kwc\-*|kyo(c|k)|le(no|xi)|lg(
g|\/*(k|l|u)|50|54|\-*[a-w])|libw|lynx|m1\-
w|m3ga|m50\/*|ma(te|ui|xo)|mc(01|21|ca)|m\-
cr|me(rc|ri)|mi(o8|oa|ts)|mmef|mo(01|02|bi|de|do|t(\-|
|o|v)|zz)|mt(50|p1|v )|mwbp|mywa|n10[0-2]|n20[2-
3]|n30(0|2)|n50(0|2|5)|n7(0(0|1)|10)|ne((c|m)\-
|on|tf|wf|wg|wt)|nok(6|i)|nzph|o2im|op(ti|wv)|oran|owg1|p800|pan([
|[1-
8]|c))|phil|pire|pl(ay|uc)|pn\-*|po(ck|rt|se)|prox|psio|pt\-
g|qa\-*a|qc(07|12|21|32|60|\-*[2-
7]|i\-*|qtek|r380|r600|raks|rim9|ro(ve|zo)|s55\/*|sa(ge|ma|mm|ms|n
|oo|p\-*|sdk\/*|se(c(\-|0|1)|47|mc|nd|ri)|sgh\-*|shar|sie(\-|
|m)|sk\-*|sl(45|id)|sm(al|ar|b3|it|t5)|so(ft|ny)|sp(01|h\-*|v\-
|v )|sy(01|mb)|t2(18|50)|t6(00|10|18)|ta(gt|lk)|tcl\-*|tdg\-
|tel(i|m)|tim\-*|t\-*mo|to(pl|sh)|ts(70|m\-
|m3|m5)|tx\-*|up(\.b|g1|si)|utst|v400|v750|veri|vi(rg|te)|vk(40|5
3)|\-*v)|vm40|voda|vulc|vx(52|53|60|61|70|80|81|83|85|98)|w3c(\-|
| )|webc|whit|wi(g |nc|nw)|wmlb|wonu|x700|yas\-
|your|zeto|zte\-*|i.test(a.substr(0,4))) mobileCheck = true;})
(navigator.userAgent||navigator.vendor||window.opera);
    return mobileCheck;
};

```

## ۱۶۷. چطوری بدون **Regex** تشخیص بدیم که یه مرورگر mobile هست یا نه؟

می‌تونیم مرورگرهای تلفن همراه رو با اجرای فهرستی از دستگاهها و بررسی اینکه آیا `navigator.userAgent` با چیزی مطابقت داره یا نه، شناسایی کنیم. این یه حل جایگزین برای استفاده از `RegExp` هستش

```
function detectmob() {
  if( navigator.userAgent.match(/Android/i)
    || navigator.userAgent.match(/webOS/i)
    || navigator.userAgent.match(/iPhone/i)
    || navigator.userAgent.match(/iPad/i)
    || navigator.userAgent.match(/iPod/i)
    || navigator.userAgent.match(/BlackBerry/i)
    || navigator.userAgent.match(/Windows Phone/i)
  ){
    return true;
  }
  else {
    return false;
  }
}
```

## ۱۶۸. چطوری طول و عرض یه تصویر رو با جاواسکریپت به دست میاری؟

می‌تونیم با استفاده از جاواسکریپت به صورت برنامه‌ریزی شده تصویر رو بدست بیاریم و ابعاد (عرض و ارتفاع) رو بررسی کنیم، برای مثال `syntax` بررسی می‌تونه طبق مثال زیر انجام بشه:

```
const img = new Image();

img.onload = function() {
  console.log(this.width + 'x' + this.height);
}

img.src = 'http://www.google.com/intl/en_ALL/images/logo.gif';
```

## ۱۶۹. چطوری درخواست‌های synchronous HTTP بزنیم؟

مرورگرها یه کلاس XMLHttpRequest ارائه می‌دان که می‌تونه برای ایجاد درخواست‌های HTTP هم‌زمان از جاواسکریپت استفاده شه.

```
function httpGet(theUrl) {
    const xhttpReq = new XMLHttpRequest();
    xhttpReq.open("GET", theUrl, false); // false for
    synchronous request
    xhttpReq.send(null);
    return xhttpReq.responseText;
}
```

## ۱۷۰. چطوری درخواست‌های asynchronous HTTP بزنیم؟

مرورگرها یه کلاس XMLHttpRequest رو ارائه می‌دان که می‌تونن برای درخواست‌های HTTP ناهم‌زمان از جاواسکریپت با ارسال پارامتر سوم به عنوان true استفاده کنن.

```
function httpGetAsync(theUrl, callback)
{
    var xhttpReq = new XMLHttpRequest();
    xhttpReq.onreadystatechange = function() {
        if (xhttpReq.readyState == 4 && xhttpReq.status ==
200)
            callback(xhttpReq.responseText);
    }
    xhttpReq.open("GET", theUrl, true); // true for asynchronous
    xhttpReq.send(null);
}
```

## ۱۷۱. چطوری یه تاریخ رو به یه تاریخ در timezone دیگه تبدیل کنیم؟

می‌تونیم از متده toLocaleString برای تبدیل تاریخ‌ها تو یه منطقه زمانی به منطقه زمانی دیگه استفاده کنیم. بریم یه مثال درموردش ببینیم.

```
console.log(event.toLocaleString('en-GB', { timeZone: 'UTC' })
}); //29/06/2019, 09:56:00
```

## ۱۷۲. چه property‌هایی برای اندازه‌گذاری سایز window به کار میره؟

می‌تونیم از ویژگی‌های `innerWidth`, `innerHeight`, `clientWidth`, `clientHeight` ویندوز، عنصر `document` و آبجکت `body` document برای پیدا کردن اندازه یه پنجره استفاده کنیم. بیانی از ترکیب اونا برای محاسبه اندازه یه `window` یا `document` استفاده کنیم.

```
const width = window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

const height = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;
```

## ۱۷۳. عملگر شرطی سه گانه توی جاواسکریپت چیه؟

عملگر شرطی `ternary` تنها عملگر جاواسکریپت هستش که سه عملوند رو می‌گیره که به عنوان میانبر برای دستور `if` عمل میکنه.

```
var isAuthenticated = false;
console.log(isAuthenticated ? 'Hello, welcome' : 'Sorry, you
are not authenticated'); //Sorry, you are not authenticated
```

## ۱۷۴. آیا می‌شه روی عملگر شرطی زنجیره شرط‌ها رو اعمال کرد؟

بله، می‌تونیم زنجیره‌سازی روی عملگرهای مشابه شرطی `if ... else if ... else if... other chain` اعمال کنیم.

```

function traceValue(someParam) {
    return condition1 ? value1
      : condition2 ? value2
      : condition3 ? value3
      : value4;
}

// The above conditional operator is equivalent to:

function traceValue(someParam) {
    if (condition1) { return value1; }
    else if (condition2) { return value2; }
    else if (condition3) { return value3; }
    else { return value4; }
}

```

## ۱۷۵. روش‌های اجرای جاواسکریپت بعد از لود شدن صفحه کدوما هستن؟

:window.onload .۱

```
window.onload = function ...
```

:document.onload .۲

```
document.onload = function ...
```

:body onload .۳

```
<body onload="script()">
```

## ۱۷۶. تفاوت‌های بین prototype و proto کدوما هستن؟

آبجکت واقعیه که در زنجیره جستجو برای حل متدها و غیره استفاده می‌شه. proto در حالی که آبجکت‌ایه که برای ساخت prototype استفاده می‌شه زمانی که یه آبجکت با new Object() ایجاد می‌کنیم.

```
( new Employee ).__proto__ === Employee.prototype;
( new Employee ).prototype === undefined;
```

## ۱۷۷. میتوانی یه مثال از زمانی که واقعاً به سمیکولون ( ; ) نیاز هست بزنی؟

توصیه می‌شده که بعد از هر عبارت در جاواسکریپت از سیمیکالن استفاده کنیم. برای مثال، توى مثال زیر نداشتن سیمیکالن، خطای `is not a function ..` را در زمان اجرا ایجاد میکنه:

```
// define a function
const fn = function () {
    //...
} // semicolon missing at this line

// then execute some code inside a closure
(function () {
    //...
})();
```

از مثال بالا جاواسکریپت اینطور برداشت میکنه

```
const fn = function () {
    //...
}(function () {
    //...
})();
```

در این حالت، تابع دوم رو به عنوان آرگومان به تابع اول ارسال می‌کنیم و سعی می‌کنیم نتیجه فراخوانی تابع اول رو به عنوان تابع فراخوانی کنیم. با خاطر همین برای تابع دوم خطای `is not a function ..` رو موقع اجرا می‌گیریم.

## ۱۷۸. متد `freeze` چیکار میکنه؟

متد `freeze` برای فریز کردن یه آبجکت استفاده می‌شه. ثابت کردن یه آبجکت اجازه افزودن ویژگی‌های جدید به یه آبجکت رو نمی‌ده. از حذفش جلوگیری میکنه و از تغییر قابلیت

شمارش پذیری، پیکربندی یا قابلیت نوشتن ویژگی‌های موجود جلوگیری میکنه. یعنی آبجکت‌ء ارسال شده رو برمی‌گردونه و کپی ثابتی ایجاد نمیکنه.

```
const obj = {
  prop: 100
};

Object.freeze(obj);
obj.prop = 200; // Throws an error in strict mode

console.log(obj.prop); //100
```

**نکته:** یه تایپ ارور بهمون می‌ده که ارگومان داده شده `object` نیست

## ۱۷۹. هدف از متده `freeze` چیه؟

۱. برای فربیز کردن آبجکت‌ها و آرایه‌ها
۲. برای کردن آبجکت‌ها `immutable`

## ۱۸۰. چرا به متده `freeze` نیاز داریم؟

در پارادایم شی گرا، یه API موجود حاوی عناصر خاصیه که قصد توسعه، اصلاح یا استفاده مجدد رو خارج از زمینه فعلی خودشون ندارن. در زبان‌های مختلف کلمه کلیدیه `final` برای داشتن همچین خاصیتی استفاده می‌شه.

## ۱۸۱. چطوری می‌تونیم زبان ترجیحی یه مرورگر رو تشخیص بدیم؟

ما می‌تونیم از آبجکت `navigator` که بر روی مرورگر وجود داره این کارو انجام بدیم

```

var language = navigator.languages && navigator.languages[0] ||
// Chrome / Firefox
    navigator.language || // All browsers
    navigator.userLanguage; // IE <= 10

console.log(language);

```

## ۱۸۲. چطوری می‌تونیم حرف اول همه کلمات یه رشته رو به حرف بزرگ تبدیل کنیم؟

ما می‌تونیم با تابع زیر این کارو انجام بدیم:

```

function toTitleCase(str) {
    return str.replace(
        /\w\S*/g,
        function(txt) {
            return txt.charAt(0).toUpperCase() +
txt.substr(1).toLowerCase();
        }
    );
}

toTitleCase("good morning john"); // Good Morning John

```

## ۱۸۳. چطوری می‌شه تشخیص داد که جاواسکریپت یه صفحه وب غیرفعال شده؟

برای تشخیص غیرفعال بودن یا نبودن جاواسکریپت می‌تونیم از تگ `<noscript>` استفاده کنیم. بلوک کد داخل `<noscript>` زمانی اجرا می‌شه که جاواسکریپت غیرفعاله و معمولاً برای نمایش محتوای جایگزین زمانی که صفحه در جاواسکریپت تولید می‌شه، استفاده می‌شه.

```
<script type="javascript">
    // JS related code goes here
</script>
<noscript>
    <a href="next_page.html?noJS=true">JavaScript is disabled
    in the page. Please click Next Page</a>
</noscript>
```

## ۱۸۴. عملگرهای پشتیبانی شده توسط جاواسکریپت کدوما هستن؟

یه عملگر قادر به دستکاری (محاسبات ریاضی و منطقی) مقدار یا عملوند معینه. اپراتورهای مختلفی توسط جاواسکریپت پشتیبانی میشن این اپراتورها هستن

۱. **عملگرهای حسابی**: شامل + (اضافه), - (منها), \* (ضرب), / (تقسیم), % (درصد), + (اضافه کردن) و - (کم کردن)

۲. **عملگرهای مقایسه ای**: شامل = (برابر), != (غیر برابر), == (برابر و تایپ برابر), < (بزرگتر), > (بزرگتر مساوی), <= (کوچکتر مساوی)

۳. **عملگرهای منطقی**: شامل && ("و" منطقی), || ("یا" منطقی), !( منطقی "نه")

۴. **عملگرهای تعیین مقدار**: شامل = (اپراتور تعیین مقدار), += (اضافه کردن و تعیین مقدار), -= (منها کردن و تعیین مقدار), \*= (ضرب و تعیین مقدار), /= (تقسیم و تعیین مقدار), %= (باقي مانده و تعیین مقدار)

۵. **اپراتور سه تایی**: شامل اپراتورهای شرطی سه تایی

۶. **اپراتور تایپ**: از اون برای پیدا کردن تایپ متغیرها استفاده میشه به صورت

`typeof variable`

## ۱۸۵. پارامتر rest چیکار میکنه؟

پارامتر **Rest** یه روش بهبود یافته برای مدیریت پارامترهای تابع هستش که به ما امکان میده تعداد نامحدودی از آرگومان‌ها رو به عنوان یه آرایه دریافت کنیم:

```
function f(a, b, ...theArgs) {
    // ...
}
```

برای مثال، بیاین یه مثال مجموع برای محاسبه تعداد پویا پارامترها در نظر بگیریم،

```
function total(...args){
    let sum = 0;
    for(let i of args){
        sum+=i;
    }
    return sum;
}
console.log(fun(1,2)); //3
console.log(fun(1,2,3)); //6
console.log(fun(1,2,3,4)); //13
console.log(fun(1,2,3,4,5)); //15
```

**نکته:** پارامتر `Rest` در ES6 به استاندارد جاوااسکریپت اضافه شد

## ۱۸۶. اگه پارامتر `rest` رو به عنوان آخرین پارامتر استفاده نکنیم چی میشه؟

پارامتر `Rest` چون وظیفه اش جمع آوری تمام آرگومان‌های باقی مونده تو یه آرایه اس پس باید همیشه آخرین پارامتر باشه. برای مثال، اگه تابعیو مثل کد زیر تعریف کنیم معنی نداره و یه خطأ ایجاد میکنه:

```
function someFunc(a,...b,c){
    //Your code goes here
    return;
}
```

## ۱۸۷. عملگرهای منطقی باینری توی جاوااسکریپت کدوما هستن؟

۱. به صورت بیتی `( & ) AND`
۲. به صورت بیتی `( | ) OR`
۳. به صورت بیتی `( ^ ) XOR`
۴. به صورت بیتی `( ~ ) NOT`
۵. تغییر مکان به چپ `( >> )`
۶. علامت در حال انتشار به سمت راست `( << )`
۷. صفر پر کردن Shift راست `( <<< )`

## ۱۸۸. عملگر spread چیکار میکنه؟

عملگر Spread به تکرارپذیرها (آرایه‌ها / اشیاء / رشته‌ها) اجازه میده تا به آرگومان‌ها / عناصر منفرد گسترش پیدا کنن. برای مشاهده این رفتار مثالی بزنیم:

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

const numbers = [1, 2, 3];

console.log(calculateSum(...numbers)); // 6
```

## ۱۸۹. چطوری تشخیص میدی که یه آبجکت freeze شده یا نه؟

متده `Object.isFrozen` برای تعیین اینکه آیا یه آبجکت منجمد هس یا نه استفاده می‌شه. اگه همه شرایط زیر درست باشه، یه آبجکت منجمد می‌شه.

۱. اگه قابل توسعه نباشه.
۲. اگه تمام خصوصیاتش غیر قابل تنظیم باشن.
۳. اگه تمام خصوصیات داده اون غیر قابل نوشتن باشه.

```
const object = {
  property: 'Welcome JS world'
};
Object.freeze(object);
console.log(Object.isFrozen(object));
```

## ۱۹۰. چطوری بررسی کنیم که دو تا مقدار(شامل آبجکت) با هم برابرن یا نه؟

متده `Object.is` تعیین میکنه که آیا دو مقدار یه مقدار هستن یا نه. برای مثال، استفاده با انواع مختلف مقادیر،

```
Object.is('hello', 'hello');      // true
Object.is(window, window);      // true
Object.is([], []); // false
```

اگه یکی از موارد زیر برقرار باشه، دو مقدار یکسان در نظر گرفته میشه:

۱. هردو undefined

۲. هردو null

۳. هردو true یا هر دو false

۴. هر دو رشته با طول یکسان با کاراکترهای مشابه به ترتیب یکسان

۵. هر دو یه آبجکت (یعنی هر دو شی رفرنس یکسان دارن)

۶. هر دو عدد و

۰+ هر دو

۰- هر دو

NaN هر دو

هر دو غیر صفر و هر دو NaN نیستن و هر دو دارای یه مقدار هستن.

## ۱۹۱. هدف از متده روی object چیه؟

برای مقایسه دو رشته یا عدد و یا آبجکت با همدیگه و یا پیدا کردن قطبیت دو عدد استفاده میشه:

## ۱۹۲. چطوری object‌های یه object رو به یه property دیگه کپی می‌کنی؟

می‌تونیم از متده `Object.assign` استفاده کنیم که برای کپی کردن مقادیر و ویژگی‌ها از یه یا چند آبجکت منبع به یه آبجکت هدف استفاده می‌شه. آبجکت مورد نظر رو که دارای خواص و مقادیر کپی شده از آبجکت اولیه اس رو برمی‌گردونه.

```
Object.assign(target, ...sources)
```

بیایین با یه منبع و یه شی هدف مثال بزنیم،

```
const target = { a: 1, b: 2 };
const source = { b: 3, c: 4 };

const returnedTarget = Object.assign(target, source);

console.log(target); // { a: 1, b: 3, c: 4 }

console.log(returnedTarget); // { a: 1, b: 3, c: 4 }
```

همونطور که در کد بالا مشاهده شد، یه ویژگی مشترک (b) از منبع به مقصد وجود داره، بنابراین مقداش بازنویسی شده.

### ۱۹۳. کاربردهای متدهای assign چیه؟

۱. برای شبیه‌سازی یه آبجکت.
۲. برای ادغام آبجکتها با ویژگی‌های یکسان.

### ۱۹۴. آبجکت proxy چیه؟

آبجکت Proxy برای تعریف رفتار سفارشی برای عملیات‌های اساسی مثل جستجوی ویژگی، تخصیص، شمارش، فراخوانی تابع و غیره استفاده می‌شه.

```
var p = new Proxy(target, handler);
```

بیاین مثالی از شیء پروکسی بزنیم،

```

const handler = {
  get: function(obj, prop) {
    return prop in obj ?
      obj[prop] :
      100;
  }
};

const p = new Proxy({}, handler);
p.a = 10;
p.b = null;

console.log(p.a, p.b); // 10, null
console.log('c' in p, p.c); // false, 100

```

در کد بالا، از کنترل‌کننده «get» استفاده می‌کنیم که رفتار پراکسی رو موقع انجام عملیات روی اون تعریف می‌کنیم.

## ۱۹۵. هدف از متد seal چیه؟

روش **Object.seal** برای مهر و موم کردن یه آبجکت با جلوگیری از اضافه شدن ویژگی‌های جدید بهش و علامت گذاری تمام ویژگی‌های موجود به عنوان غیر قابل تنظیم، استفاده می‌شه. اما مقادیر پراپرتی‌های فعلی تا زمانی که قابل نوشتمن باشن، قابل تغییر هستن. بیان مثال زیرو برای درک بیشتر در مورد روش seal ببینیم

```

const object = {
  property: 'Welcome JS world'
};
Object.seal(object);
object.property = 'Welcome to object world';
console.log(Object.isSealed(object)); // true
delete object.property; // You cannot delete when sealed
console.log(object.property); //Welcome to object world

```

## ۱۹۶. کاربردهای متد seal چیه؟

۱. برای آب بندی آبجکت‌ها و آرایه‌ها استفاده می‌شه.

۲. برای غیرقابل تغییر کردن یه آبجکت استفاده می‌شه.

## ۱۹۷. تفاوت‌های بین متدهای `freeze` و `seal` چیا هست؟

اگه یه آبجکت با استفاده از متدهای `Object.freeze` منجمد شه، ویژگی‌هاش تغییرناپذیر می‌شن و هیچ تغییری در اونا نمی‌توnim ایجاد کنیم در حالی که اگه یه آبجکت با استفاده از متدهای `Object.seal` مهر و موم شده باشه، می‌شه تغییرات روی ویژگی‌های موجود ایجاد کرد.

## ۱۹۸. چطوری تشخیص میدی که یه آبجکت `seal` شده یا نه؟

- ۱. متدهای `Object.isSealed` برای تعیین مهر و موم بودن یا نبودن یه آبجکت استفاده می‌شه. اگه همه شرایط زیر درست باشه یه شی مهر و موم می‌شه
  - ۱. اگه قابل توسعه نباشه.
  - ۲. اگه تمام خصوصیات اون غیر قابل تنظیم باشن.
  - ۳. اگه قابل جابجایی نباشه (اما لزوماً غیرقابل نوشتن نیست).
- ۲. بیاین اونو در عمل ببینیم:

```
const object = {
  property: 'Hello, Good morning'
};

Object.seal(object); // Using seal() method to seal the object

console.log(Object.isSealed(object)); // checking whether
// the object is sealed or not
```

## ۱۹۹. چطوری کلید و مقدارهای `enumerable` رو به دست می‌اري؟

متدهای `Object.entries` برای برگرداندن آرایه‌ای از جفت‌های `[key, value]` دارای کلید رشته‌ای شمارش‌پذیر یه شی معین، به همون ترتیبی که توسط یه حلقه `for...in` ارائه می‌شه، استفاده می‌شه. بیاین عملکرد متدهای `Object.entries` رو تو یه مثال ببینیم،

```

const object = {
    a: 'Good morning',
    b: 100
};

for (let [key, value] of Object.entries(object)) {
    console.log(` ${key}: ${value}`);
    // a: 'Good morning'
    // b: 100
}

```

نکته: ترتیب به عنوان آبجکت تعریف شده تضمین نمی‌شود.

## ۲۰۰. تفاوت‌های بین متدهای Object.entries و Object.values چیا هست؟

رفتار متد `Object.entries` مشابه روش `Object.values` هست اما به جای جفت آرایه‌ای از مقادیر را بر می‌گردوند.

```

const object = {
    a: 'Good morning',
    b: 100
};

for (let value of Object.values(object)) {
    console.log(` ${value}`); // 'Good morning' 100
}

```

## ۲۰۱. چطوری لیست کلیدهای یه `object` رو بدست می‌اري؟

مي‌تونيم از متد `Object.keys` استفاده کنيم که برای برگردوندن آرایه‌اي از اسم ويزگی‌های یه آبجکت معين استفاده مي‌شه، به همون ترتيبی که با يه حلقه معمولی دریافت مي‌کنيم. برای مثال:

```
const user = {
  name: 'John',
  gender: 'male',
  age: 40
};

console.log(Object.keys(user)); //['name', 'gender', 'age']
```

## ۲۰۲. چطوری به object prototype می‌کنی؟

متدهای `Object.create` برای ایجاد یک object جدید با `object prototype` و ویژگی‌های مشخص شده استفاده می‌شوند. برای مثال، از یک object موجود به عنوان `object prototype` جدید ایجاد شده استفاده می‌کنند. یه object جدید را با `object prototype` ویژگی‌های مشخص شده برمی‌گردونند.

```
const user = {
  name: 'John',
  printInfo: function () {
    console.log(`My name is ${this.name}.`);
  }
};

const admin = Object.create(user);

admin.name = "Nick"; // Remember that "name" is a property set
on "admin" but not on "user" object

admin.printInfo(); // My name is Nick
```

## ۲۰۳. چیه WeakSet؟

برای ذخیره مجموعه‌ای از اشیاء ضعیف (مرجع ضعیف) استفاده می‌شوند.

```
new WeakSet([iterable]);
```

بیان مثال زیر و برای توضیح رفتارش ببینیم،

```

var ws = new WeakSet();
var user = {};
ws.add(user);
ws.has(user);    // true
ws.delete(user); // removes user from the set
ws.has(user);    // false, user has been removed

```

## ۲۰۴. تفاوت‌های بین Set و WeakSet کدوما هستن؟

تفاوت اصلی اینه که ارجاع به اشیاء تو Set قویه در حالی که ارجاع به اشیا تو WeakSet ضعیفه. برای مثال، یه شی تو WeakSet میتونه زباله جمع آوری شه اگه مرجع دیگری به اون وجود نداشته باشه.

تفاوت‌های دیگر عبارتند از

۱. مجموعه‌ها میتونن هر مقداری رو ذخیره کنن در حالی که WeakSets میتونه تنها مجموعه‌ای از اشیاء رو ذخیره کنه
۲. WeakSet برخلاف Set دارای ویژگی اندازه نیست
۳. WeakSet متدهایی مانند پاک کردن، کلیدها، مقادیر، ورودی‌ها، forEach و رو نداره.
۴. WeakSet قابل تکرار نیست.

## ۲۰۵. لیست متدهایی که رو WeakSet قابل استفاده هستن رو می‌تونی بگی؟

۱. (add)(value): یه شی جدید با مقدار داده شده به مجموعه ضعیف اضافه می‌شه
  ۲. (delete)(value): مقدار رو از مجموعه WeakSet حذف میکنه.
  ۳. (has)(value): اگه مقدار در مجموعه WeakSet وجود داشته باشه true رو برمی‌گردونه در غیر این صورت false رو برمی‌گردونه.
  ۴. length: طول ضعیف SetObject رو برمی‌گردونه
- بیاین عملکرد تمام روش‌های بالا رو توی یه مثال ببینیم،

```
var weakSetObject = new WeakSet();
var firstObject = {};
var secondObject = {};
// add(value)
weakSetObject.add(firstObject);
weakSetObject.add(secondObject);
console.log(weakSetObject.has(firstObject)); //true
console.log(weakSetObject.length()); //2
weakSetObject.delete(secondObject);
```

## ۲۰۶. چیه WeakMap؟

آبجکت WeakMap مجموعه‌ای از جفت‌های کلید/مقداره که تو اون کلیدها به صورت ضعیف ارجاع داده شدن. در این حالت، کلیدها باید اشیا باشن و مقادیر میتونن مقادیر دلخواه باشن. برای مثال:

```
new WeakMap([iterable])
```

بیاین مثال زیرو برای توضیح رفتار اون ببینیم،

```
var ws = new WeakMap();
var user = {};
ws.set(user);
ws.has(user);    // true
ws.delete(user); // removes user from the map
ws.has(user);    // false, user has been removed
```

## ۲۰۷. تفاوت‌های بین Map و WeakMap کدوما هستن؟

تفاوت اصلی اینه که ارجاعات به آبجکت‌ها کلیدی در نقشه قوی هستن در حالی که ارجاعات به اشیاء کلیدی در WeakMap ضعیف هستن. برای مثال، یه شی کلیدی در WeakMap در صورتی که هیچ مرجع دیگری بهش وجود نداشته باشه، میتونه زباله جمع آوری شه.

تفاوت‌های دیگر عبارتند از

۱. WeakMaps ها میتوانن هر نوع کلیدی رو ذخیره کن، در حالی که فقط میتوانه مجموعه ای از اشیاء کلیدی رو ذخیره کن.
۲. برخلاف Map دارای ویژگی size WeakMap نیست.
۳. متدهایی مثل clear, keys, values, entries forEach WeakMap رو نداره.
۴. قابل تکرار WeakMap نیست.

## ۲۰۸. لیست متدهایی که رو WeakMap قابل استفاده هستن رو می‌تونی بگی؟

۱. مقدار کلید رو در آبجکت WeakMap تنظیم میکنه. آبجکت WeakMap رو برمی‌گردونه.
۲. هر مقدار مربوط به کلید رو حذف میکنه.
۳. یه Boolean رو برمی‌گردونه که نشون میده آیا مقداری به کلید در WeakMap مرتبط شده اس یا نه.
۴. مقدار مربوط به کلید رو برمی‌گردونه، یا اگه کلیدی وجود نداشته باشه، تعريف نشده.

بیایین عملکرد تمام روش‌های بالا رو تو یه مثال ببینیم،

```
const weakMapObject = new WeakMap();
const firstObject = {};
const secondObject = {};
// set(key, value)
weakMapObject.set(firstObject, 'John');
weakMapObject.set(secondObject, 100);
console.log(weakMapObject.has(firstObject)); //true
console.log(weakMapObject.get(firstObject)); // John
weakMapObject.delete(secondObject);
```

## ۲۰۹. هدف از متدهای uneval چیه؟

uneval یه تابع داخلیه که برای ایجاد نمایش رشته‌ای از کد منبع یه شی استفاده می‌شه. این یه تابع سطح بالاس و با هیچ آبجکت‌ای مرتبط نیست. بیاین مثال زیر رو درموردش ببینیم:

```
var a = 1;
uneval(a); // returns a String containing 1
uneval(function user() {}); // returns "(function user(){})"
```

## ۲۱۰. چطوری یه URL رو encode می‌کنی؟

تابع `encodeURI` برای رمزگذاری URI کامل استفاده می‌شود که دارای کاراکترهای خاص به جز (، /، #، \$، +، =، &، @، :، ?) هست.

```
var uri = 'https://mozilla.org/?x=шеллы';
var encoded = encodeURI(uri);
console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
```

## ۲۱۱. چطوری یه URL رو decode می‌کنی؟

تابع `decodeURI` برای رمزگشایی یه شناسه منبع یکنواخت (URI) که قبلاً توسط `encodeURI` ایجاد شده اس، استفاده می‌شود.

```
var uri = 'https://mozilla.org/?x=шеллы';
var encoded = encodeURI(uri);
console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
try {
    console.log(decodeURI(encoded)); // "https://mozilla.org/?x=шеллы"
} catch(e) { // catches a malformed URI
    console.error(e);
}
```

## ۲۱۲. چطوری محتوای یه صفحه رو پرینت می‌گیری؟

آجکت `window` یه متده `print` ارائه می‌کند که برای چاپ محتویات پنجره فعلی استفاده می‌شود. یه قادر محاوره ای چاپ رو باز می‌کند که بهمون این امکان رو میده که بین

گزینه‌های مختلف برای چاپ انتخاب کنیم. بیاین استفاده از روش چاپ رو تو یه مثال ببینیم،

```
<input type="button" value="Print" onclick="window.print()" />
```

**نکته:** بیشتر مرورگرها، زمانی که قادر چاپ بازه صفحه قفل می‌شه.

## ۲۱۳. تفاوت‌های بین eval و uneval چیا هستن؟

تابع 'uneval' منبع یه شی معین رو برمی‌گردونه. در حالی که تابع "eval" با ارزیابی اون کد منبع تو یه ناحیه حافظه متفاوت، بر عکس عمل می‌کنه. بیاین مثالی رو برای روشن شدن تفاوت ببینیم،

```
var msg = uneval(function greeting() { return 'Hello, Good morning'; });
var greeting = eval(msg);
greeting(); // returns "Hello, Good morning"
```

## ۲۱۴. تابع anonymous چیه؟

تابع ناشناس یه تابع بدون اسمه! توابع ناشناس معمولاً به یه نام متغیر اختصاص داده می‌شن یا به عنوان یه تابع callback استفاده می‌شن برمی‌یم یه مثال در موردش ببینیم،

```
function (optionalParameters) {
    //do something
}

const myFunction = function(){ //Anonymous function assigned to a variable
    //do something
};

[1, 2, 3].map(function(element){ //Anonymous function used as a callback function
    //do something
});
```

بیان تابع ناشناس بالا رو تو یه مثال ببینیم،

```
var x = function (a, b) {return a * b};
var z = x(5, 10);
console.log(z); // 50
```

## ۲۱۵. تفاوت تقدم بین متغیرهای local و global چطوریه؟

یه متغیر local بر یه متغیر global با همون اسم ارجاعیت داره. بیان این رفتار رو تو یه مثال ببینیم.

```
const msg = "Good morning";
function greeting() {
    msg = "Good Evening";
    console.log(msg);
}
greeting();
```

## ۲۱۶. accessor چیکار می‌کنن؟

ECMAScript 5 های آبجکت جاواسکریپت یا ویژگی‌های محاسبه شده رو از طریق گیرنده‌ها و تنظیم‌کننده‌ها معرفی کرد. Getters از کلمه کلیدی "get" استفاده میکنه در حالی که Setters از کلمه کلیدی "set" استفاده میکنه.

```
const user = {
    firstName: "John",
    lastName : "Abraham",
    language : "en",
    get lang() {
        return this.language;
    }
    set lang(lang) {
        this.language = lang;
    }
};
console.log(user.lang); // getter access lang as en
user.lang = 'fr';
console.log(user.lang); // setter used to set lang as fr
```

## ۲۱۷. چطوری روی Object یه مقدار تعريف می‌کنی؟

متده استاتیک `Object.defineProperty` برای تعريف یه ویژگی جدید به طور مستقیم بر روی یه آبجکت یا تغییر ویژگی موجود روی یه آبجکت استفاده می‌شه و آبجکت را برمی‌گردونه. بیاین مثالی رو ببینیم تا بدونیم چجوری ویژگی رو تعريف کنیم:

```
const newObject = {};

Object.defineProperty(newObject, 'newProperty', {
  value: 100,
  writable: false
});

console.log(newObject.newProperty); // 100

newObject.newProperty = 200; // It throws an error in strict
mode due to writable setting
```

## ۲۱۸. تفاوت‌های بین `defineProperty` و `get` چیا هست؟

هر دو نتایج مشابهی دارن مگه اینکه از کلاس‌ها استفاده کنیم. اگه از «`get`» استفاده می‌کنیم ویژگی روی `prototype` شی تعريف می‌شه، در حالی که با استفاده از `Object.defineProperty`، ویژگی روی نمونه‌ای که بهش اعمال می‌شه، تعريف می‌شه.

## ۲۱۹. مزایای استفاده از `Setter` و `Getter` چیه؟

۱. اونا `sadeh` syntax ساده تری ارائه میدن
۲. اونا برای تعريف ویژگی‌های محاسبه شده یا دسترسی‌ها در JS استفاده می‌شن
۳. برای ارائه رابطه هم ارزی بین خواص و روش‌ها مفیدن
۴. اونا می‌تونن کیفیت داده‌های بهتری رو ارائه بدن
۵. برای انجام کارها در پشت صحنه با منطق محصور شده مفیدن.

## ۲۲۰. می‌تونیم **defineProperty** را با استفاده از متدهای **getter** و **setter** تعريف کنیم؟

بله، می‌توانیم از روش `Object.defineProperty` برای اضافه کردن Getters و Setters استفاده کنیم. برای مثال، آبجکت شمارنده زیر از ویژگی‌های افزایش، کاهش، جمع و تفریق استفاده می‌کنیم:

```
const obj = {counter : 0};

// Define getters
Object.defineProperty(obj, "increment", {
  get : function () {this.counter++;}
});
Object.defineProperty(obj, "decrement", {
  get : function () {this.counter--;}
});

// Define setters
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
});
Object.defineProperty(obj, "subtract", {
  set : function (value) {this.counter -= value;}
});

obj.add = 10;
obj.subtract = 5;
console.log(obj.increment); //6
console.log(obj.decrement); //5
```

## ۲۲۱. هدف استفاده از **switch-case** چیه؟

عبارت `switch case` تو جاوااسکریپت برای اهداف تصمیم‌گیری استفاده می‌شود. تواند مورد، استفاده از دستور `switch case` راحت‌تر از `if-else` هست. برایم یه مثال در موردش بینیم:

```

switch (expression)
{
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;

    .
    .
    case valueN:
        statementN;
        break;
    default:
        statementDefault;
}

```

دستور چند حالته بالا یه راه آسون برای انجام عملیات مختلف بهمون میده که بر اساس مقدار مبنا میشه عملکردهای مختلفی رو در نظر گرفت.

## ۲۲۲. چه قواعدی برای استفاده از switch-case باید رعایت بشه؟

۱. عبارت میتونه از نوع عددی یا رشته‌ای باشه.
۲. مقادیر تکراری برای عبارت مجاز نیستن.
۳. بیانیه default اختیاریه. اگه عبارت ارسال شده به سوئیچ با هیچ مقدار case مطابقت نداشته باشه، دستور تو حالت پیش فرض اجرا میشه.
۴. دستور break در داخل سوئیچ برای پایان دادن به دنباله دستور استفاده میشه.
۵. عبارت break اختیاریه. اما اگه حذف شه، اجرا در مورد بعدی ادامه پیدا میکنه

## ۲۲۳. نوع داده‌های primitive کدوما هستن؟

- یه نوع داده primitive داده‌ایه که دارای یه مقدار اولیه اس (که هیچ ویژگی یا روشی نداره). ۵ نوع نوع داده اولیه وجود داره.
۱. string
  ۲. number

- ۳. boolean.
- ۴. null.
- ۵. undefined.

## ۲۲۴. روش‌های مختلف دسترسی به object‌های property کدوما هستن؟

۱. **دسترسی با نقطه:** از نقطه برای دسترسی به ویژگی‌ها استفاده می‌کنیم

```
objectName.property
```

۲. **دسترسی با کروشه:** از کروشه برای دسترسی به دیتا استفاده می‌کنیم

```
objectName["property"]
```

۳. **دسترسی عبارتی:** از عبارت توی کروشه استفاده می‌کنیم

```
[objectName[expression]]
```

## ۲۲۵. قوانین پارامترهای توابع کدوما هستن؟

۱. تعاریف تابع انواع داده‌ها رو برای پارامترها مشخص نمی‌کنیم.

۲. بررسی نوع آرگومان‌های ارسال شده رو انجام ندین.

۳. تعداد آرگومان‌های دریافتی رو بررسی نکنیم.

تابع زیر از قوانین بالا پیروی می‌کنند،

```
function functionName(parameter1, parameter2, parameter3) {
  console.log(parameter1); // ۱
}
functionName(۱);
```

## ۲۲۶. آبجکت error چیه؟

۲۲۶. آبجکت خطا (error object) یه موقع بروز خطا، اطلاعات خطا رو ارائه میده و این دو ویژگی رو داخلش داره: name و message. برای مثال، تابع زیر جزئیات خطا رو ثبت میکنه:

```
try {
    greeting("Welcome");
}
catch(err) {
    console.log(err.name + "<br>" + err.message);
}
```

## ۲۲۷. چه موقعی خطا (syntax error) دریافت می‌کنیم؟

اگه بخواییم کد رو با یه خطا syntax ارزیابی کنیم یه SyntaxError ارسال می‌شه. برای مثال، کد زیر برای پارامتر تابع یه خطا syntax ایجاد میکنه:

```
try {
    eval("greeting('welcome')"); // Missing ' will produce an
error
}
catch(err) {
    console.log(err.name);
}
```

## ۲۲۸. عنوان خطاها مختلف که روی error-object برمیگردند کدام هستن؟

توضیحات	نام خطا
خطایی تو تابع eval رخ داده	EvalError
خطایی با عدد "خارج از محدوده"	RangeError
خطا به دلیل ارجاع غیرقانونی	خطای مرجع
خطای ناشی از خطا syntax	SyntaxError

نام خطا	توضیحات
TypeError	خطای ناشی از خطای type
URIE	یه خطأ به دلیل encodeURIComponent

## ۲۲۹. عبارات مختلف که در هنگام مدیریت error استفاده میشن کدوما هستن؟

۱. این عبارت برای آزمایش یه بلوک کد برای خطاها استفاده میشه
۲. این عبارت برای رسیدگی به خطا استفاده میشه
۳. این عبارت برای ایجاد خطاهای سفارشی استفاده میشه.
۴. این عبارت برای اجرای کد پس از تلاش و گرفتن بدون توجه به نتیجه استفاده میشه.

## ۲۳۰. دو نوع مختلف حلقه‌ها تو جاواسکریپت کدوما هستن؟

۱. حلقه‌های کنترل شده توسط ورودی: تو این نوع حلقه، شرایط تست قبل از ورود به بدنه حلقه آزمایش میشه. برای مثال For Loop و While Loop تو این دسته قرار میگیرن.
۲. حلقه‌های کنترل شده توسط خروجی: در این نوع حلقه، شرایط تست در انتهای بدنه حلقه آزمایش یا ارزیابی میشه. یعنی بدنه حلقه حداقل یه بار بدون در نظر گرفتن شرایط تست true یا false اجرا میشه. برای مثال، حلقه do-while در این دسته قرار میگیره.

## ۲۳۱. nodejs چیه؟

Node.js یه پلتفرم سمت سروره که بر اساس زمان اجرا جاواسکریپت کروم برای ساخت آسان برنامه‌های شبکه سریع و مقیاس پذیر ساخته شده. این یه زمان اجرا ۰/۰/۱ ناهمزمان

مبتنی بر رویداد، غیر مسدود کننده اس که از موتور جاوا‌سکریپت V8 گوگل و کتابخونه libuv استفاده می‌کنه.

## ۲۳۳۲. آبجکت Intl چیه؟

آبجکت Intl فضای نامی برای ECMAScript Internationalization API اس که مقایسه رشته‌های حساس زبان، قالب بندی اعداد و قالب بندی تاریخ و زمان رو ارائه میده. دسترسی به چندین سازنده و توابع حساس به زبان رو فراهم می‌کنه.

## ۲۳۳۳. چطوری تاریخ و زمان رو بر اساس زبان جاری سیستم کاربر نمایش بدیم؟

می‌توnim از کلاس Intl.DateTimeFormat استفاده کنیم که سازنده آبجکت‌هایی که قالب بندی تاریخ و زمان حساس به زبان رو فعال می‌کنه. بیاین این رفتار رو با یه مثال ببینیم،

```
var date = new Date(Date.UTC(2019, 07, 07, 3, 0, 0));
console.log(new Intl.DateTimeFormat('en-GB').format(date)); // 07/08/2019
console.log(new Intl.DateTimeFormat('en-AU').format(date)); // 07/08/2019
```

## ۲۳۴۴. Iterator چیه؟

آبجکت‌ایه که پس از خاتمه، یه توالی و یه مقدار بازگشتی رو تعریف می‌کنه. پروتکل Iterator رو با متد «next» پیاده‌سازی می‌کنه که یه شی رو با دو ویژگی برمی‌گردونه: «value» (مقدار بعدی در دنباله) و «done» (که اگه آخرین مقدار در دنباله مصرف شده باشه درسته.).

## ۲۳۵۵. حلقه‌های synchronous (همزمان) چطوری کار می‌کنن؟

تکرار همزمان در ES6 معرفی شد و با مجموعه ای از اجزای زیر کار میکنه:  
**Iterable**: این یه آبجکت‌ایه که میتونه از طریق روشی که کلید اون `Symbol.iterator` هس تکرار شه.

**Iterator**: این یه آبجکت‌ایه که با فراخوانی «`[Symbol.iterator]`» بر روی یه تکرار برگردونده میشه. این آبجکت تکرار شونده هر عنصر تکرار شده رو تو یه آبجکت پیچیده میکنه و اوно از طریق متده «`next`» یکی برمی‌گردونه.

**IteratorResult**: این یه آبجکت‌ایه که با متده «`next`» برگردونده میشه. آبجکت شامل دو ویژگی است. ویژگی "value" حاوی یه عنصر تکرار شده و ویژگی "done" تعیین میکنه که آیا عنصر آخرین عنصر هست یا نه.

بیانی تکرار همزمان رو با آرایه ای مانند زیر نشون بدیم،

```
const iterable = ['one', 'two', 'three'];
const iterator = iterable[Symbol.iterator]();
console.log(iterator.next()); // { value: 'one', done: false }
console.log(iterator.next()); // { value: 'two', done: false }
console.log(iterator.next()); // { value: 'three', done: false }
}
console.log(iterator.next()); // { value: 'undefined', done: true }
```

## ۲۳۶. Event-loop چیه؟

event loop یه صف از توابع callback موقعی که یه تابع `async` اجرا میشه، تابع callback در صف قرار می‌گیرد. موتور جاوااسکریپت پردازش حلقه رویداد رو شروع نمیکنه تا زمانی که تابع `async` اجرای کد رو تموم کنه.

**نکته:** این به Node.js اجازه میده تا عملیات I/O غیر مسدود کننده رو انجام بده حتی اگه جاوااسکریپت تک رشته‌ای باشه.

## ۲۳۷. Call-stack چیه؟

Call Stack یه ساختار داده برای مفسران جاوااسکریپت‌هه تا فراخونی‌های تابع تو برنامه رو پیگیری کنه و دو عمل عمده داره،

۱. هر زمان که یه تابع رو برای اجرای آن فراخوانی می‌کنیم اونو به `stack` هدایت می‌شه.

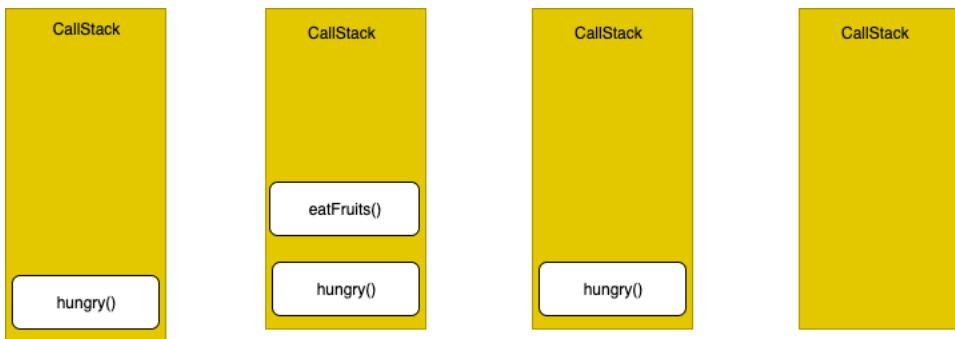
۲. هر زمان که اجرا کد تموم شه، تابع از stack خارج می‌شه.  
بیاین یه مثال و توضیح خلاصه اون در قالب نمودار رو باهم ببینیم:

```
function hungry() {
    eatFruits();
}

function eatFruits() {
    return "I'm eating fruits";
}

// Invoke the `hungry` function
hungry();
```

- کد بالا تو یه call-stack به صورت زیر پردازش می‌شه.
۱. تابع 'hungry' رو به لیست call-stack اضافه می‌شه و کد رو اجرا می‌شه.
  ۲. تابع 'eatFruits' رو به لیست call-stack اضافه می‌شه و کد رو اجرا می‌شه.
  ۳. تابع 'eatFruits' از لیست call-stack ما حذف می‌شه.
  ۴. تابع 'eatFruits' از لیست call-stack ما حذف می‌شه.



## ۲۳۸. چیه؟ Event-queue

Event-queue مسئول ارسال توابع جدید به stack برای پردازش. ساختارش به صورت صف داده اس تا توالی درستی رو نگه داره و همه عملیات باید برای اجرا ارسال شن.

## ۲۳۹. چیه؟ Decorator

عبارتیه که یه تابع رو ارزیابی میکنه و هدف، نام و توصیف‌کننده تزئین رو به عنوان آرگومان می‌گیره. همچنین، به صورت اختیاری یه توصیف‌گر دکوراتور رو برای نصب بر روی آبجکت مورد نظر برمی‌گردونه. بیاین در زمان طراحی، دکوراتور ادمین رو برای کلاس کاربر تعریف کنیم:

```
function admin(isAdmin) {
    return function(target) {
        target.isAdmin = isAdmin;
    }
}

@admin(true)
class User() {
}
console.log(User.isAdmin); //true

@admin(false)
class User() {
}
console.log(User.isAdmin); //false
```

## ۲۴۰. مقادیر موجود روی آبجکت Intl کدوما هستن؟

۱. آبجکت‌هایی هستن که مقایسه رشته‌های حساس به زبان رو امکان پذیر می‌کنن.
۲. آبجکت‌هایی هستن که قالب بندی تاریخ و زمان حساس به زبان رو فعال می‌کنن.
۳. آبجکت‌هایی هستن که قالب بندی لیست حساس به زبان رو فعال می‌کنن.
۴. آبجکت‌هایی که قالب بندی اعداد حساس به زبان رو فعال می‌کنن.
۵. آبجکت‌هایی که قالب بندی حساس به جمع و قوانین خاص زبان رو برای جمع فعال می‌کنن.
۶. آبجکت‌هایی که قالب بندی زمان نسبی حساس به زبان رو فعال می‌کنن.

## ۲۴۱. عملگر Unary چیه؟

عملگر unary (+) برای تبدیل یه متغیر به عدد استفاده می‌شه. اگه متغیر قابل تبدیل نباشه، همچنان به عدد تبدیل می‌شه اما با مقدار NaN. بیاین این رفتار رو تو یه عمل ببینیم:

```
const x = "100";
const y = + x;
console.log(typeof x, typeof y); // string, number

const a = "Hello";
const b = + a;
console.log(typeof a, typeof b, b); // string, number, NaN
```

## ۲۴۲. چطوری المنتهای موجود تو یه آرایه رو مرتب می‌کنی؟

متد sort برای مرتب سازی عناصر یه آرایه در جای خود استفاده می‌شه و آرایه مرتب شده رو برمی‌گردونه. برای مثال

```
const months = ["Aug", "Sep", "Jan", "June"];
months.sort();
console.log(months); // ["Aug", "Jan", "June", "Sep"]
```

## ۲۴۳. هدف از تابع مرتب‌سازی موقع استفاده از متد sort چیه؟

عنصر آرایه به رشته تبدیل می‌شن، سپس بر اساس مقدار نقطه کد یونیک هر کاراکتر مرتب می‌شون بیاین مثالی بزنیم تا کاربرد compareFunction رو ببینیم،

```
let numbers = [1, 2, 5, 3, 4];
numbers.sort((a, b) => b - a);
console.log(numbers); // [5, 4, 3, 2, 1]
```

## ۲۴۴. چطوری آیتم‌های یه آرایه رو معکوس مرتب کنیم؟

برای معکوس کردن عناصر یه آرایه می‌توانیم از متده reverse استفاده کنیم. این روش برای مرتب کردن یه آرایه به ترتیب نزولی مفید است. بیاین استفاده از متده reverse رو تو یه مثال ببینیم،

```
let numbers = [1, 2, 5, 3, 4];
numbers.sort((a, b) => b - a);
console.log(numbers); // [5, 4, 3, 2, 1]
```

## ۲۴۵. چطوری حداقل و حداکثر مقدار یه آرایه رو بدست بیاریم؟

می‌توانیم از روش‌های `Math.max` و `Math.min` روی متغیرهای آرایه برای یافتن حداقل و حداکثر عناصر تو یه آرایه استفاده کنیم. بیاین دو تابع برای پیدا کردن مقدار `min` و `max` تو یه آرایه ایجاد کنیم:

```
const marks = [50, 20, 70, 60, 45, 30];
function findMin(arr) {
    return Math.min(...arr)
}
function findMax(arr) {
    return Math.max(...arr));
}

console.log(findMin(marks));
console.log(findMax(marks));
```

## ۲۴۶. چطوری حداقل و حداکثر مقدار یه آرایه رو بدون استفاده از متدهای `Math` بدست بیاریم؟

ما می‌توانیم توابعی بنویسیم که تو یه آرایه حلقه می‌زن و هر مقدار را با کمترین یا بالاترین مقدار مقایسه می‌کنن تا مقادیر حداقل و حداکثر رو پیدا کنن. بریم یه مثال درمودش ببینیم:

```

const marks = [50, 20, 70, 60, 45, 30];
function findMin(arr) {
    let min = arr[0];
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] < min) {
            min = arr[i]
        }
    }
    return min;
}

function findMax(arr) {
    let max = arr[0];
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i]
        }
    }
    return max;
}

console.log(findMin(marks));
console.log(findMax(marks));

```

## ۲۴۷. عبارت خالی چیه و هدف از استفاده ازش چیه؟

سیمیکالن ; هس که نشون میده هیچ دستوری اجرا نمیشه، حتی اگه syntax جاواسکریپت به اون نیاز داشته باشه. از اونجایی که هیچ اقدامی با دستور خالی وجود نداره، ممکنه فکر کنیم که استفاده از اون خیلی کمه اما دستور خالی موقعی مفیده که می‌خواین یه حلقه ایجاد کنیم که بدنهاش خالیه. برای مثال، می‌تونیم یه آرایه با مقادیر صفر رو مثل کد زیر مقداردهی اولیه کنیم.

```
// Initialize an array a
for(int i=0; i < a.length; a[i++] = 0) ;
```

## ۲۴۸. چطوری metadata یه مازول رو بدست میاری؟

میتوانیم از آبجکت `import.meta` استفاده کنیم که یه ویژگی متعاره که متا داده‌های متنی خاص رو تو یه مازول جاوا اسکریپت قرار می‌ده. این شامل اطلاعاتی در مورد مازول فعلی، مانند URL مازوله. در مرورگرها، ممکنه متا داده‌های متفاوتی نسبت به NodeJS دریافت کنیم.

```
<script type="module" src="welcome-module.js"></script>

<script>
console.log(import.meta);
// { url: "file:///home/user/welcome-module.js" }
</script>
```

## ۲۴۹. عملگر comma چیه و چیکار میکنه؟

عملگر کاما برای ارزیابی هر یه از عملوندهاش از چپ به راست استفاده می‌شه و مقدار آخرین عملوند رو برمی‌گردونه. این کاملاً با استفاده از کاما در آرایه‌ها، اشیاء و آرگومان‌ها و پارامترهای تابع متفاوته. بریم یه مثال در موردش بینیم.

```
var x = 1;
x = (x++, x);

console.log(x); // 2
```

## ۲۵۰. مزایای استفاده از عملگر comma چیه؟

ممولاً برای گنجوندن چن تا عبارت تو جایی که یه عبارت واحد نیاز داره استفاده می‌شه. یکی از کاربردهای رایج این عملگر کاما، ارائه چندین پارامتر تو یه حلقه «`for`» اس. برای مثال، حلقه `for` زیر از چند عبارت تو یه مکان واحد با استفاده از عملگر کاما استفاده میکنه.

```
for (var a = 0, b = 10; a <= 10; a++, b--)
```

همچنین می‌تونیم از عملگر کاما تو یه عبارت بازگشتی استفاده کنیم جایی که قبل از بازگشت پردازش میکنه.

```
function myFunction() {
  let a = 1;
  return (a += 10, a); // 11
}
```

## ۲۵۱. چیه؟ TypeScript

یه ابر مجموعه تایپ شده از جاواسکریپت که توسط مایکروسافت ایجاد شده که انواع اختیاری، کلاس‌ها، و بسیاری `async/wait` ویژگی‌های دیگر را اضافه میکنه و به جاواسکریپت ساده کامپایل میکنه. `Angular` به طور کامل در `TypeScript` ساخته شده و به عنوان زبان اصلی استفاده می‌شه. می‌توانیم اونو به صورت گلوبال نصب کنیم:

```
npm install -g typescript
```

بیاین یه مثال ساده از استفاده از `TypeScript` رو بینیم،

```
function greeting(name: string): string {
  return "Hello, " + name;
}

let user = "Ali Karimi";

console.log(greeting(user));
```

متده `greeting` فقط نوع رشته رو به عنوان آرگومان مجاز میکنه.

## ۲۵۲. تفاوت‌های بین `typescript` و `javascript` کدوما هستن؟

javascript	typescript	ویژگی
زبان اسکریپت	زبان برنامه نویسی شی گرا	پارادایم زبان
دارای تایپ پویا	پشتیبانی از تایپ استاتیک	پشتیبانی از تایپ

javascript	typescript	ویژگی
پشتیبانی نمی‌شود	پشتیبانی شده	ماژول‌ها
از رابطهای پشتیبانی نمی‌کند	دارای مفهوم رابط	رابط
عدم پشتیبانی از پارامترهای اختیاری اختیاری برای توابع	توابع از پارامترهای اختیاری پشتیبانی می‌کنند	پارامترهای اختیاری

## ۲۵۳. مزایای javascript نسبت به typescript چیاست؟

۱. میتوانه خطاهای زمان کامپایل رو فقط در زمان توسعه پیدا کنه و باعث می‌شه خطاهای زمان اجرا کمتر شه. در حالی که جاواسکریپت يه زبان تفسیر شده است.
۲. TypeScript به شدت تایپ می‌شه یا از تایپ استاتیک پشتیبانی می‌کنه که امکان بررسی صحت نوع رو در زمان کامپایل فراهم می‌کنه. این در جاواسکریپت در دسترس نیست.
۳. کامپایلر TypeScript برخلاف ویژگی‌های ES6 جاواسکریپت که ممکنه در بعضی از مرورگرها پشتیبانی نشه، میتوانه فایل‌های ts رو در ES3، ES4 و ES5 کامپایل کنه.

## ۲۵۴. چیه object-initializer ؟

عبارتیه که مقدار دهی اولیه یه آبجکت رو توصیف می‌کنه. object-initializer syntax این عبارت به صورت فهرستی با کاما از صفر یا چند جفت نام ویژگی و مقادیر مرتبط یه آبجکت، محصور در براکت {} نشون داده می‌شه. این همچنین به عنوان نماد تحت اللفظی شناخته می‌شه. یکی از راههای ایجاد یه آبجکته.

```
const initObject = {a: 'John', b: 50, c: {}};

console.log(initObject.a); // John
```

## ۲۵۵. متد constructor چیه؟

متد constructor یه متد خاص برای ایجاد و مقداردهی اولیه یه آجکت ایجاد شده تو یه کلاسه. اگه متد constructor رو مشخص نکنیم از constructor پیش فرض استفاده می‌شه. بریم یه مثال در موردش ببینیم:

```
class Employee {
  constructor() {
    this.name = "John";
  }
}

const employeeObject = new Employee();

console.log(employeeObject.name); // John
```

## ۲۵۶. اگه متد constructor رو بیش از یه بار توی کلاس بنویسیم چی می‌شه؟

تو یه کلاس یه متد خاصه و باید فقط یه بار تو یه کلاس تعریف شه. اگه متد سازنده رو بیش از یه بار تو یه کلاس بنویسیم، یه خطای SyntaxError ایجاد میشه.

```
class Employee {
  constructor() {
    this.name = "John";
  }
  constructor() { // Uncaught SyntaxError: A class may only
    have one constructor
    this.age = 30;
  }
}

const employeeObject = new Employee();

console.log(employeeObject.name);
```

## ۲۵۷. چطوری متدهای constructor کلاس والد رو صدا بزنیم؟

می‌توانیم از کلمه کلیدی `super` برای فراخوانی `constructor` کلاس والد استفاده کنیم. یادمون باش که `super` باید قبل از استفاده از مرجع `this` فراخوانی شه. در غیر این صورت باعث خطای `Reference error` می‌شه. بیانی از اون استفاده کنیم:

```
class Square extends Rectangle {
    constructor(length) {
        super(length, length);
        this.name = 'Square';
    }

    get area() {
        return this.width * this.height;
    }

    set area(value) {
        this.area = value;
    }
}
```

## ۲۵۸. چطوری object یه prototype رو به دست میاری؟

می‌توانیم از روش `Object.getPrototypeOf(obj)` برای برگرداندن آبجکت مشخص شده استفاده کنیم. یعنی مقدار ویژگی `prototype` داخلی. اگه هیچ ویژگی ارشی وجود نداشته باشه، مقدار `null` برگردانده می‌شه.

```
; {} = const newPrototype
; (const newObject = Object.create(newPrototype)

console.log(Object.getPrototypeOf(newObject) === newPrototype);
// true
```

## ۲۵۹. اگه به متدهای `getPrototypeOf` رشته پاس بدیم چی می‌شه؟

در ES5، اگه پارامتر `obj` یه آبجکت نباشه، یه استثنای `TypeError` ایجاد میکنه. در حالی که در ES2015، پارامتر به یه آبجکت اجباری تبدیل می‌شه.

```
// ES5
Object.getPrototypeOf('James'); // TypeError: "James" is not an
object
// ES2015
Object.getPrototypeOf('James'); // String.prototype
```

## ۲۶۰. چطوری object یه object روی یه object دیگه سمت کنیم؟

می‌تونیم از متدهای `Object.setPrototypeOf` استفاده کنیم که (یعنی ویژگی داخلی «Prototype») یه آبجکت مشخص شده رو روی یه آبجکت دیگه یا تهی تنظیم می‌کنیم. برای مثال، اگه بخوایم `prototype` آبجکت `Square` رو روی آبجکت `Rectangle` تنظیم کنیم این شکلی می‌شه این کارو انجام داد:

```
Object.setPrototypeOf(Square.prototype, Rectangle.prototype);
Object.setPrototypeOf({}, null);
```

## ۲۶۱. چطوری بررسی می‌کنی که یه object قابل extend هست یا نه؟

متدهای `Object.isExtensible` برای تعیین اینکه یه آبجکت قابل توسعه هس یا نه (یعنی اینکه می‌توانه ویژگی‌های جدیدی به اون اضافه شه یا نه) استفاده می‌شه.

```
const newObject = {};
console.log(Object.isExtensible(newObject)); //true
```

نکته: به طور پیش فرض، همه ی آبجکت‌ها قابل گسترش هستن. برای مثال، ویژگی‌های جدید رو می‌تونیم اضافه یا تغییر بدیم.

## ۲۶۲. چطوری جلوی object یه extend رو بگیریم؟

متدهای `Object.preventExtensions` برای جلوگیری از افزودن ویژگی‌های جدید به یه آبجکت استفاده می‌شه. به عبارت دیگر، از پسوندهای بعدی به آبجکت جلوگیری می‌کنیم. بیان استفاده از این ویژگی رو ببینیم:

```

const newObject = {};
Object.preventExtensions(newObject); // NOT extendable

try {
  Object.defineProperty(newObject, 'newProperty', { // Adding
    new property
    value: 100
  });
} catch (e) {
  console.log(e); // TypeError: Cannot define property
  newProperty, object is not extensible
}

```

## ۲۶۳. روش‌های مختلف برای تبدیل یه object غیرقابل extend چیه؟

می‌تونیم یه آبجکت غیر قابل گسترش رو به ۳ روش علامت گذاری کنیم.

۱. Object.preventExtensions .

۲. Object.seal .

۳. Object.freeze .

```

var newObject = {};

Object.preventExtensions(newObject); // Prevent objects are
non-extensible
Object.isExtensible(newObject); // false

var sealedObject = Object.seal({}); // Sealed objects are non-
extensible
Object.isExtensible(sealedObject); // false

var frozenObject = Object.freeze({}); // Frozen objects are
non-extensible
Object.isExtensible(frozenObject); // false

```

## ۲۶۴. چطوری های متعددی رو روی یه object تعریف می‌کنی؟

متدهای `Object.defineProperties` برای تعریف یا اصلاح ویژگی‌های موجود مستقیماً روی یه آبجکت و برگرداندن آبجکت استفاده می‌شه. بیاین چندین ویژگی رو روی یه آبجکت خالی تعریف کنیم:

```
const newObject = {};

Object.defineProperties(newObject, {
  newProperty1: {
    value: 'John',
    writable: true
  },
  newProperty2: {}
});
```

## ۲۶۵. منظور از MEAN توی جاواسکریپت چیه؟

دسته MEAN (MongoDB، Express، AngularJS و Node.js) محبوب‌ترین دسته‌بندی فناوری نرمافزار جاواسکریپت منبع بازه که برای ساخت برنامه‌های وب پویا در دسترسه، جایی که می‌توانیم کدهای سمت سرور و سمت کلاینت پروژه وب رو کاملاً با جاواسکریپت بنویسیم.

## ۲۶۶. منظور از Obfuscation توی جاواسکریپت چیه و چیکار می‌کنه؟

عمل عمدى ایجاد کد جاواسکریپت مبهم (یعنی کد منبع یا ماشین) هس که درک اون برای انسان سخته. این چیزی شبیه به رمزگذاریه، اما یه ماشین می‌تونه کد رو درک کنه و اونو اجرا کنه. بیاین تابع زیر رو قبل از Obfuscation بینیم،

```
function greeting() {
  console.log('Hello, welcome to JS world');
}
```

و بعد از کد به صورت زیر ظاهر می‌شه

```

eval(function(p,a,c,k,e,d){e=function(c){return
c};if(!''.replace(/\^/,String)){while(c--)d[c]=k[c]||c;k=
[function(e){return d[e]}];e=function()
{return'\\w+'};c=1;while(c--){if(k[c]){p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c])}};return p}('2 1(){0.3('4, 7
6 5
ole|greeting|function|log|Hello|JS|to|welcome|world'.split(' '|'),0
({})

```

## ۲۶۷. چه نیازی به Obfuscate کردن داریم؟

۱. اندازه کد کمتر میشه. که باعث انتقال سریع تر داده بین سرور و کلاینت میشه.
۲. این منطق کسب و کار رو از دنیای خارج پنهان میکنه و از کد در برابر دیگران محافظت میکنه
۳. مهندسی معکوس کردن کد رو سخت میکنه
۴. زمان دانلود کاهش پیدا میکنه

## ۲۶۸. چیه؟ Minification

حذف تمام کاراکترهای غیر ضروریه (فضاهای خالی حذف می‌شن) و Minification متغیرها بدون تغییر در عملکرد اون تغییر نام میدن همچنین برای مبهم سازی کد هم استفاده میشه.

## ۲۶۹. مزایای minification یا کم حجم‌سازی چیه؟

به طور معمول توصیه می‌شه برای ترافیک سنگین و نیازهای فشرده منابع از Minification استفاده کنیم. اندازه فایل رو با مزایای زیر کاهش میده

۱. زمان بارگذاری یه صفحه وب رو کاهش میده
۲. در مصرف پهنای باند صرفه جویی میکنه

## ۲۷۰. تفاوت‌های بین Encryption و Obfuscation چیه؟

Encryption	Obfuscation	ویژگی
تغییر فرم اطلاعات به فرمت ناخوانا با استفاده از کلید	تغییر فرم هر داده به هر شکل دیگر	تعریف
برای رمزگشایی کلید لازمه	می‌شه اونو بدون هیچ کلید رمزگشایی کرد	کلیدی برای رمزگشایی
تبديل به فرمت ناخوانا	به فرم پیچیده تبدیل می‌شه	فرمت داده‌های هدف

## ۲۷۱. ابزارهای مختلف برای minification کدوما هستن؟

۱. کامپایلر بسته شدن گوگل UglifyJS
۲. jsmin
۳. /javascript-minifier.com
۴. prettydiff.com
- ۵.

## ۲۷۲. چطوری اعتبارسنجی فرم رو با javascript انجام میدی؟

می‌شه از جاواسکریپت برای اعتبارسنجی فرم HTML استفاده کرد. برای مثال، اگه فیلد فرم خالی باشه، تابع باید اطلاع بد و false رو برگردونه تا از ارسال فرم جلوگیری شه.  
بریم ورود کاربر رو در فرم html انجام بدیم:

```
<form name="myForm" onsubmit="return validateForm()" method="post">
    User name: <input type="text" name="uname">
    <input type="submit" value="Submit">
</form>
```

و اعتبار سنجی ورود کاربر به این شکل هست.

```
function validateForm() {
  const x = document.forms["myForm"]["uname"].value;
  if (x == "") {
    alert("The username shouldn't be empty");
    return false;
  }
}
```

## ۳۷۳. چطوری اعتبارسنجی فرم رو بدون javascript انجام میدی؟

می‌تونیم بدون استفاده از جاواسکریپت اعتبار سنجی فرم HTML رو به صورت خودکار انجام بدین. اعتبار سنجی با اعمال ویژگی `required` برای جلوگیری از ارسال فرم زمانی که `required` خالیه فعال می‌شه.

```
<form method="post">
  <input type="text" name="uname" required>
  <input type="submit" value="Submit">
</form>
```

**نکته:** اعتبار سنجی فرم خودکار برای IE9 یا ورژن‌های قبل از اون کار نمیکنه.

## ۳۷۴. متدهای موجود روی DOM برای اعتبارسنجی کدوما هستن؟

۱. اگه یه عنصر ورودی حاوی داده‌های معتبر باشه، مقدار `true` رو برمی‌گردونه.

۲. برای تنظیم خاصیت `validationMessage` یه عنصر ورودی `setCustomValidity` استفاده می‌شه.

بیاین یه فرم ورود کاربر با اعتبارسنجی DOM بگیریم

```

function myFunction() {
    var userName = document.getElementById("uname");
    if (!userName.checkValidity()) {
        document.getElementById("message").innerHTML =
        userName.validationMessage;
    } else {
        document.getElementById("message").innerHTML = "Entered a
valid username";
    }
}

```

## ۲۷۵. مقادیر موجود روی DOM برای اعتبارسنجی کدوما هستن؟

۱. validity: فهرستی از ویژگی‌های بولین مربوط به اعتبار یه عنصر ورودی رو ارائه میده.

۲. validationMessage: زمانی که اعتبار نادرست باشه، پیام رو نمایش میده.

۳. willValidate: این نشون میده که آیا یه عنصر ورودی اعتبار سنجی می‌شه یا نه.

## ۲۷۶. مقادیر موجود روی input برای اعتبارسنجی کدوما هستن؟

ویژگی validity یه عنصر ورودی مجموعه ای از ویژگی‌های مربوط به اعتبارسنجی داده‌ها رو ارائه میده.

۱. customError: اگه یه پیام اعتبار سفارشی تنظیم شده باشه، true رو برمی‌گردونه.

۲. patternMismatch: اگه مقدار یه عنصر با ویژگی الگوی آن مطابقت نداشته باشه، مقدار true رو برمی‌گردونه.

۳. rangeOverflow: اگه مقدار یه عنصر از ویژگی max آن بیشتر باشه، مقدار true رو برمی‌گردونه.

۴. rangeUnderflow: اگه مقدار یه عنصر کمتر از ویژگی min باشه، مقدار true رو برمی‌گردونه.

۵. stepMismatch: اگه مقدار عنصر مطابق با ویژگی step نامعتبر باشه، مقدار true رو برمی‌گردونه.

۶. `tooLong`: اگه مقدار یه عنصر از ویژگی `maxLength` آن بیشتر شه، مقدار `true` رو برمی‌گردونه.

۷. `typeMismatch`: اگه مقدار یه عنصر بر اساس ویژگی نوع نامعتبر باشه، مقدار `true` رو برمی‌گردونه.

۸. `valueMissing`: اگه عنصری با ویژگی مورد نیاز ارزش نداشته باشه، مقدار `true` رو برمی‌گردونه.

۹. `valid`: اگه مقدار یه عنصر معتبر باشه، مقدار `true` رو برمی‌گردونه.

## ۲۷۷. یه مثال از استفاده ویژگی `rangeOverflow` می‌تونی بزنی؟

اگه مقدار یه عنصر از ویژگی `max` اون بیشتر باشه، ویژگی `rangeOverflow` مقدار `true` رو برمی‌گردونه. برای مثال، توی فرم پایین اگه مقدار ورودی بیش از 100 باشه، خطای میده.

```
<input id="age" type="number" max="100">
<button onclick="myOverflowFunction()">OK</button>
```

```
function myOverflowFunction() {
  if (document.getElementById("age").validity.rangeOverflow) {
    alert("The mentioned age is not allowed");
  }
}
```

## ۲۷۸. جاواسکریپت قابلیت استفاده از `enum` رو پیش‌فرض توی خودش داره؟

نه، جاواسکریپت به صورت بومی از `enum` ها پشتیبانی نمیکنه. اما انواع مختلفی از راه‌حل‌ها برای شبیه‌سازی اونا وجود داره، اگرچه ممکنه معادلهای دقیقی ارائه نکنن. برای مثال، می‌تونیم از `seal` یا `freeze` روی آبجکت استفاده کنیم:

```
const DaysEnum = Object.freeze({ "monday": 1, "tuesday": 2,
"wednesday": 3, ... })
```

## ۲۷۹. چیه enum؟

enum نوعیه که متغیرها را به مقدار از مجموعه ای از ثابت‌های از پیش تعریف شده محدود میکنه. جاواسکریپت هیچ enum نداره اما تایپ‌اسکریپت از enum داخلی پشتیبانی میکنه.

```
} enum Color
RED, GREEN, BLUE
{
```

## ۲۸۰. چطوری همه object‌هایی به property را به دست بیاریم؟

می‌تونیم از متده استفاده کنیم که آرایه‌ای از تمام ویژگی‌هایی را که مستقیماً تو آبجکت داده شده یافت می‌شه رو برمی‌گردونه. بیاین استفاده از اونو تو یه مثال بینیم:

```
const newObject = {
  a: 1,
  b: 2,
  c: 3
};

console.log(Object.getOwnPropertyNames(newObject));  ["a", "b",
"c"]
```

## ۲۸۱. چطوری property-descriptor‌هایی آبجکت را بدست بیاریم؟

می‌تونیم از متده استفاده کنیم که تمام توصیف‌گرهای ویژگی‌یه آبجکت معین رو برمی‌گردونه. بیاین استفاده از اونو توی یه مثال بینیم:

```

const newObject = {
  a: 1,
  b: 2,
  c: 3
};
const descriptorsObject =
Object.getOwnPropertyDescriptors(newObject);
console.log(descriptorsObject.a.writable); //true
console.log(descriptorsObject.a.configurable); //true
console.log(descriptorsObject.a.enumerable); //true
console.log(descriptorsObject.a.value); // 1

```

## ۲۸۲. گزینه‌هایی که موقع تعریف ویژگی **descriptor** با **object** داریم کدوما هستن؟

۱. value: ارزش مرتبط با پراپرتی
۲. writable: تعیین میکنه که آیا مقدار مرتبط با ویژگی قابل تغییر هس یا نه
۳. configurable: اگه بتونیم نوع descriptor این ویژگی رو تغییر بدیم و اگه بتونیم ویژگی رو از آبجکت مربوطه حذف کنیم، مقدار true رو برمی‌گردونه.
۴. enumerable: تعیین میکنه که ویژگی در موقع شمارش خصوصیات روی آبجکت مربوطه ظاهر می‌شه یا نه.
۵. set: تابعی که به عنوان تنظیم کننده برای ویژگی عمل میکنه
۶. get: تابعی که به عنوان یه گیرنده برای ملک عمل میکنه

## ۲۸۳. چطوری کلاس‌ها رو extend می‌کنی؟

کلمه کلیدی extends در اعلان‌ها/عبارات کلاس برای ایجاد کلاسی که فرزند کلاس دیگه ایه استفاده می‌شه. می‌شه از اون برای زیر کلاس بندی کلاس‌های سفارشی و همچنین اشیاء داخلی استفاده کرد. بریم یه مثال در موردش ببینیم،

```
class ChildClass extends ParentClass { ... }
```

بیاین یه نمونه از زیر کلاس مربع از کلاس والد Polygon رو مثال بزنیم:

```

class Square extends Rectangle {
    constructor(length) {
        super(length, length);
        this.name = 'Square';
    }

    get area() {
        return this.width * this.height;
    }

    set area(value) {
        this.area = value;
    }
}

```

## ۲۸۴. چطوری آدرس صفحه رو بدون رفرش صفحه عوض کنیم؟

برای تغییر url میشه از `window.location.url` استفاده کرد اما این کار باعث بارگیری دوباره صفحه میشه. HTML5 متد `history.pushState` و `history.replaceState` را معرفی کرد که به شما اجازه میده به ترتیب ورودی‌های تاریخ رو اضافه و تغییر بدین. برای مثال، می‌تونیم از `pushState` مثل کد زیر استفاده کنیم.

```
window.history.pushState('page2', 'Title', '/page2.html');
```

## ۲۸۵. چطوری بررسی می‌کنی که یه آرایه یه مقدار مشخص رو داره یا نه؟

متدهای `Array.includes` برای تعیین اینکه آیا یه آرایه مقدار خاصی رو بین ورودی‌های خودش داره یا نه با برگرداندن `true` یا `false` استفاده می‌شه. بیاین مثالی برای پیداکردن یه عنصر (عددی و رشته‌ای) تو یه آرایه بینیم:

```

const numericArray = [1, 2, 3, 4];
console.log(numericArray.includes(3)); // true

const stringArray = ['green', 'yellow', 'blue'];
console.log(stringArray.includes('blue')) //true

```

## ۲۸۶. چطوری آرایه‌های scalar را با هم مقایسه می‌کنی؟

می‌توانیم از `length` و هر روش آرایه برای مقایسه دو آرایه اسکالار (مقایسه مستقیم با استفاده از `==`) استفاده کنیم. ترکیب این دو تا روش می‌توانه نتیجه مورد نیازمون را به ما بده:

```
const arrayFirst = [1,2,3,4,5];
const arraySecond = [1,2,3,4,5];
console.log(arrayFirst.length === arraySecond.length &&
arrayFirst.every((value, index) => value ===
arraySecond[index])); // true
```

اگه بخوایم آرایه‌ها رو بدون توجه به ترتیب مقایسه کنیم باید اونا رو قبل از مقایسه، مرتب کنیم.

```
const arrayFirst = [2,3,1,4,5];
const arraySecond = [1,2,3,4,5];
console.log(arrayFirst.length === arraySecond.length &&
arrayFirst.sort().every((value, index) => value ===
arraySecond[index])); //true
```

## ۲۸۷. چطوری می‌شه پارامترهای صفحه رو از متدهای GET گرفت؟

کلاس `URL` رشته `url` رو می‌پذیرد و از ویژگی `searchParams` این آجکت می‌توانیم برای دسترسی به پارامترهای `get` استفاده کنیم. ممکنه برای دسترسی به URL در مرورگرهای قدیمی (از جمله IE) نیاز به استفاده از `polyfill` یا `window.location` داشته باشیم.

```
let urlString = "http://www.some-domain.com/about.html?
x=1&y=2&z=3"; //window.location.href
let url = new URL(urlString);
let parameterZ = url.searchParams.get("z");
console.log(parameterZ); // 3
```

## ۲۸۸. چطوری اعداد رو می‌شه سه رقم سه رقم جدا کرد؟

می‌توانیم از متده استفاده کنیم که رشته‌ای رو با نمایشی حساس به زبان مثل جداگانه هزار، ارز و غیره از این عدد برمی‌گردونه.

```
function convertToThousandFormat(x){
    return x.toLocaleString(); // 12,345.679
}

console.log(convertToThousandFormat(12345.6789));
```

## ۲۸۹. تفاوت بین javascript و java چیه؟

هر دو زبان برنامه نویسی کامل‌ا نامرتبط هستن و هیچ ارتباطی بین اونا وجود نداره. جاوا بصورت ایستا تایپ می‌شه، کامپایل می‌شه، روی ماشین مجازی خودش اجرا می‌شه. در حالی که جاواسکریپت به صورت پویا تایپ می‌شه، تفسیر می‌شه و در محیط‌های مرورگر و nodejs اجرا می‌شه. بیاین تفاوت‌های عمدہ رو در قالب جدولی بینیم:

جاواسکریپت	جاوا	ویژگی
این یه زبان تایپ شده پویاسن	این یه زبان قوی تایپ شده اس	تایپ شده
برنامه نویسی مبتنی بر prototype	برنامه نویسی شی گرا	پارادایم
محدوده عملکردی	محدوده بلوک	محدوده
مبتنی بر رویداد	بر اساس موضوع	همزمانی
از حافظه کمتری استفاده می‌کنه. از این رو برای صفحات وب استفاده خواهد شد	از حافظه بیشتر استفاده می‌کنه	حافظه

## ۲۹۰. آیا جاواسکریپت namespace رو پشتیبانی می‌کنه؟

جاواسکریپت به طور پیش فرض از namespace پشتیبانی نمی‌کنه. بنابراین اگه هر عنصری (تابع، متده، آبجکت، متغیر) ایجاد کنیم گلوبال می‌شه و namespace گلوبال رو آلوده

میکنے. بیان مثالی از تعریف دو تابع بدون namespace بزنیم،

```
function func1() {
    console.log("This is a first definition");

}

function func1() {
    console.log("This is a second definition");
}

func1(); // This is a second definition
```

همیشه تعریف تابع دوم رو فراخوانی میکنے. در این صورت namespace مشکل برخورد نام رو حل میکنے.

## ۲۹۱. چطوری namespace تعریف می‌کنی؟

حتی اگه جاواسکریپت قادر namespace باشه، میتونیم از IIFE برای ایجاد استفاده کنیم.

۱. استفاده از **Object literal**: بیان متغیرها و توابع رو درون یه Object literal بپیچیم که به عنوان namespace عمل میکنے. پس از اون میتونیم با استفاده از نماد آبجکت به اونا دسترسی داشته باشیم

```
const namespaceOne = {
    function func1() {
        console.log("This is a first definition");
    }
}
const namespaceTwo = {
    function func1() {
        console.log("This is a second definition");
    }
}
namespaceOne.func1(); // This is a first definition
namespaceTwo.func1(); // This is a second definition
```

۲. استفاده از IIFE (توابع بیانی بلافصله صدازده شده): جفت پرانتز بیرونی IIFE یه محدوده محلی برای تمام کدهای داخلش ایجاد میکنے و تابع ناشناس رو به یه عبارت تابع تبدیل میکنے. به همین دلیل، میتوانیم یه تابع رو در تو دو عبارت تابع مختلف ایجاد کنیم تا به عنوان namespace عمل کنه.

```
(function() {
    function fun1(){
        console.log("This is a first definition");
    } fun1();
}());

(function() {
    function fun1(){
        console.log("This is a second definition");
    } fun1();
}());
```

۳. استفاده کردن از بلوک و تعریف‌کننده `let` و `const`: در ES6، می‌توانیم از یه بلوک و یه اعلان `let` برای محدود کردن دامنه یه متغیر به یه بلوک استفاده کنیم.

```
{
    let myFunction= function fun1(){
        console.log("This is a first definition");
    }
    myFunction();
}
//myFunction(): ReferenceError: myFunction is not defined.

{
    let myFunction= function fun1(){
        console.log("This is a second definition");
    }
    myFunction();
}
//myFunction(): ReferenceError: myFunction is not defined.
```

۴. چطوری می‌توانیم تکه کد جاواسکریپت داخل یه `iframe` رو از صفحه والد صدا بزنیم؟

در ابتدا باید `iFrame` با استفاده از `document.getElementById` یا `iFrame` با استفاده از `contentWindow` به قابل دسترسی باشه. پس از اون ویژگی `targetFunction` دسترسی میده

```
document.getElementById('targetFrame').contentWindow.targetFunction();
window.frames[0].frameElement.contentWindow.targetFunction();
// Accessing iframe this way may not work in latest versions
chrome and firefox
```

## ۲۹۳. چطوری می‌شه اختلاف date رو از آجکت timezone بگیریم؟

می‌تونیم از روش `gettimezoneOffset` کلاس Date استفاده کنیم. این روش اختلاف منطقه زمانی رو بر حسب دقیقه از محلی فعلی (تنظیمات سیستم میزبان) به UTC برمی‌گردونه.

```
const offset = new Date().getTimezoneOffset();
console.log(offset); // -480
```

## ۲۹۴. چطوری فایل‌های CSS و JS رو به شکل داینامیک بارگذاری کنیم؟

می‌تونیم هر دو عنصر پیوند و اسکریپت رو توی DOM ایجاد کنیم و اونا رو به عنوان child به تگ head اضافه کنیم. بیانیه تابع برای اضافه کردن منابع اسکریپت و سبک مثل زیر ایجاد کنیم:

```
function loadAssets(filename, filetype) {
  if (filetype == "css") { // External CSS file
    const fileReference = document.createElement("link")
    fileReference.setAttribute("rel", "stylesheet");
    fileReference.setAttribute("type", "text/css");
    fileReference.setAttribute("href", filename);
  } else if (filetype == "js") { // External JavaScript file
    const fileReference = document.createElement('script');
    fileReference.setAttribute("type", "text/javascript");
    fileReference.setAttribute("src", filename);
  }
  if (typeof fileReference != "undefined")
    document.getElementsByTagName("head")
[0].appendChild(fileReference)
}
```

## ۲۹۵. روش‌های مختلف برای پیدا کردن element‌ها توی DOM کدوما هستن؟

اگه بخوایم به هر عنصری در صفحه HTML دسترسی داشته باشیم، باید با دسترسی به آبجکت document شروع کنیم. بعداً می‌تونیم از یکی از روش‌های زیر برای پیدا کردن عنصر HTML استفاده کنیم.

۱. (document.getElementById(id)): یه عنصر رو با id پیدا می‌کنه
۲. (document.getElementsByTagName(name)): یه عنصر رو با اسم تگ پیدا می‌کنه
۳. (document.getElementsByClassName(name)): یک عنصر رو با اسم کلاس پیدا می‌کنه

## ۲۹۶. jQuery چیه؟

jQuery یه کتابخونه جاواسکریپت متقابل مرورگر محبوبه که با به حداقل رسوندن اختلاف بین مرورگرها، پیمایش مدل آبجکت (DOM)، مدیریت رویداد، اینیمیشن‌ها و تعاملات AJAX رو فراهم می‌کنه. با فلسفه اش «کمتر بنویس، بیشتر انجام بد» شهرت زیادی داره. برای مثال، می‌تونیم پیام خوش آمدگویی رو موقع بارگذاری صفحه با استفاده از jQuery به صورت زیر نمایش بدین.

```
$(document).ready(function(){ // It selects the document and
    apply the function on page load
    alert('Welcome to jQuery world');
});
```

\*\*نکته:\*\* می‌تونیم اونو از سایت رسمی jquery دانلود کنیم یا از CDN‌ها مثل گوگل نصب کنیم.

## ۲۹۷. موتور V8 جاواسکریپت چیه؟

V8 یه موتور جاواسکریپت با کارایی بالا منبع بازه که توسط مرورگر Google Chrome استفاده می‌شه و به زبان C++ نوشته شده است. توی پروژه node.js استفاده می‌شه. macOS و WebAssembly رو پیاده‌سازی می‌کنه و روی ویندوز 7 یا بالاتر، ECMAScript

+10.12 و سیستم‌های لینوکس که از پردازنده‌های MIPS، IA-32، ARM x64 یا استفاده می‌کنند اجرا می‌شوند.

**نکته:** می‌توانه به صورت مستقل اجرا شوند یا می‌توانه در هر برنامه C++ تعبیه شوند.

## ۲۹۸. چرا ما جاواسکریپت رو به عنوان یه زبان داینامیک می‌شناسیم؟

جاوسکریپت یه زبان ساده تایپ شده یا یه زبان پویا‌عه چون متغیرها در جاواسکریپت مستقیماً با هیچ نوع مقدار خاصی مرتبط نیستند و هر متغیری رو می‌شوند با مقادیری از همه نوع تخصیص/تخصیص مجدد داد.

```
let age = 50;      // age is a number now
age = 'old'; // age is a string now
age = true; // age is a boolean
```

## ۲۹۹. عملگر void چیکار می‌کنه؟

عملگر `void` عبارت داده شده رو ارزیابی می‌کند و بعد تعریف نشده (یعنی بدون برگشتن مقدار) رو برمی‌گردونه. بریم یه مثال در موردش ببینیم:

```
void (expression)
void expression
```

بیاین پیامی رو بدون هیچ گونه تغییر مسیر یا بارگیری مجدد نمایش بدیم

```
a href="javascript:void(alert('Welcome to JS world'))">Click>
<here to see a message</a>
```

**نکته:** این عملگر بیشتر برای بدست آوردن مقدار اولیه تعریف نشده با استفاده از `(void)()` استفاده می‌شوند.

## ۳۰۰. چطوری می‌شه نمایشگر موس صفحه رو به درحال لود تغییر داد؟

مکان نما رو می‌شه برای انتظار در جاواسکریپت با استفاده از ویژگی `cursor` تنظیم کرد.  
بیاین این رفتار رو در بارگذاری صفحه با استفاده از تابع زیر انجام بدیم:

```
function myFunction() {
    window.document.body.style.cursor = "wait";
}
```

و این تابع در بارگذاری صفحه فراخوانی می‌شه

```
<body onload="myFunction()">
```

## ۱. چطوری می‌شه یه حلقه بی‌نهایت درست کرد؟

می‌تونیم حلقه‌های بی‌نهایت با استفاده از حلقه‌های `for` و `while` بدون استفاده از هیچ عبارتی ایجاد کنیم. ساختار یا `syntax` حلقه `for` از نظر ESLint و ابزارهای بهینه ساز کد، رویکرد بهتری برای این کار هست.

```
for (;;) {}
while(true) { }
```

## ۲. چرا باید در استفاده از عبارت `with` تجدیدنظر کرد؟

دستور `With` جاواسکریپت در نظر گرفته شده بود که مختصراً برای نوشتن دسترسی‌های تکرارشونده به آبجکت ارائه بده. بنابراین می‌توانه با کاهش نیاز به تکرار یه مرجع طولانی بدون جرمیه عملکرد، به کاهش اندازه فایل کمک کنه. بیاین مثالی بزنیم که تو اون برای جلوگیری از افزونگی موقع چندین بار دسترسی به یه آبجکت استفاده می‌شه.

```
a.b.c.greeting = 'welcome';
a.b.c.age = 32;
```

استفاده از "with" کد رو به این شکل تبدیل می‌کنه:

```
with(a.b.c) {
    greeting = "welcome";
    age = 32;
}
```

اما این عبارت `with` مشکلات عملکردی ایجاد میکنه، چون نمی‌شه پیش‌بینی کرد که آیا یه `with` آرگومان به یه متغیر واقعی اشاره میکنه یا به یه ویژگی توی آرگومان.

## ۳۰۳. خروجی این حلقه‌ها چی می‌شه؟

```
for (var i = 0; i < 4; i++) { // global scope
    setTimeout(() => console.log(i));
}

for (let i = 0; i < 4; i++) { // block scope
    setTimeout(() => console.log(i));
}
```

خروجی حلقه‌های بالا ۰ ۱ ۲ ۴ ۴ ۴ ۴ ۰ ۳ است

**توضیح:** با توجه به صف رویداد/حلقه جاواسکریپت، تابع 'setTimeout' بعد از اجرای حلقه فراخونی می‌شه. از اونجایی که متغیر `i` با کلمه کلیدی `var` تعریف می‌شه، به یه متغیر گلوبال تبدیل می‌شه و با استفاده از تکرار زمانی که تابع `time setTimeout` فراخوانی می‌شه، مقدارش برابر با ۴. بنابراین، خروجی حلقه اول '۰ ۴ ۴ ۴' میشه. حالا توی حلقه دوم، متغیر `i` به عنوان کلمه کلیدی `let` تعریف می‌شه، به متغیری با محدوده بلوک تبدیل می‌شه و یه مقدار جدید (۰, ۱, ۲) برای هر تکرار داره. بنابراین، خروجی حلقه دوم «۰ ۱ ۲ ۳» میشه.

## ۳۰۴. می‌تونی یه سری از ویژگی‌های ES6 رو اسم ببری؟

۱. پشتیبانی از ثابت‌ها یا متغیرهای تغییرناپذیر
۲. پشتیبانی Block-scope برای متغیرها، ثابت‌ها و توابع Arrow functions.
۳. پارامترهای پیش فرض

۵.	پارامترهای Rest and Spread
۶.	Template Literals
۷.	Multi-line Strings
۸.	Destructuring Assignment
۹.	Enhanced Object Literals
۱۰.	Promises
۱۱.	Classes
۱۲.	Modules

## ۳۰۵. ES6 چیه؟

ES6 ششمین نسخه از زبان جاواسکریپت‌ه و در ژوئن 2015 منتشر شد. در ابتدا با نام ECMAScript 6 (ES6) شناخته شد و بعداً به ECMAScript 2015 تغییر نام داد. تقریباً همه مرورگرهای مدرن از ES6 پشتیبانی می‌کنند اما برای مرورگرهای قدیمی ترانسپایلرهای زیادی وجود داره، مثل Babel.js و غیره

## ۳۰۶. آیا می‌توانیم متغیرهای تعریف شده با let و const را مجدداً تعریف کنیم؟

نه، ما نمی‌توانیم متغیرهای let و const را مجدداً تعریف کنیم. اگه این کار رو انجام بدیم، یه خطای syntax دریافت می‌کنیم:

```
Uncaught SyntaxError: Identifier 'someVariable' has already
been declared
```

**توضیح:** اعلان متغیر با کلمه کلیدی "var" به یه محدوده تابع اشاره داره و با متغیر به دلیل ویژگی hoisting به گونه ای رفتار می‌شنه که مثلاً در بالای محدوده محصور تعریف شده. پس همه اعلان‌های چندگانه بدون هیچ خطایی تو ایجاد یه متغیر hoisted مشترک نقش دارن. بیاین مثالی از اعلان مجدد متغیرها تو یه محدوده برای متغیرهای var و let/const بزنیم.

```

var name = 'John';
function myFunc() {
    var name = 'Nick';
    var name = 'Abraham'; // Re-assigned in the same function
block
    alert(name); // Abraham
}
myFunc();
alert(name); // John

```

اعلان چندگانه با محدوده بلوک، خطای syntax ایجاد میکنه،

```

let name = 'John';
function myFunc() {
    let name = 'Nick';
    let name = 'Abraham'; // Uncaught SyntaxError: Identifier
'name' has already been declared
    alert(name);
}

myFunc();
alert(name);

```

## ۳۰۷. آیا استفاده از const برای تعریف متغیر او نا رو immutable میکنه؟

نه، متغیر const مقدار رو تغییرناپذیر نمیکنه. اما تخصیص‌های بعدی رو مجاز نمی‌دونه (یعنی می‌تونیم با ایجاد متغیر اعلام کنیم اما بعداً نمی‌تونیم مقدار دیگه ای رو یا نوع اون متغیر رو عوض کنیم)

```

const userList = [];
userList.push('John'); // Can mutate even though it can't re-
assign
console.log(userList); // ['John']
uerList = 123; //throws an error

```

## ۳۰۸. parameter های پیشفرض چی هستن؟

در ES5، برای مدیریت مقادیر پیش‌فرض پارامترهای تابع، باید به عملگرهای OR منطقی وابسته باشیم. ولی در ES6، ویژگی پارامترهای تابع پیش‌فرض اجازه میده تا پارامترها مقادیر پیش‌فرض مقداردهی اولیه بشن، اگه مقداری یا تعریف‌نشده ارسال نشه. بیاین رفتار رو با یه مثال مقایسه کنیم:

```
//ES5
var calculateArea = function(height, width) {
    height = height || 50;
    width = width || 60;

    return width * height;
}
console.log(calculateArea()); //300
```

پارامترهای پیش‌فرض، مقداردهی اولیه رو ساده‌تر میکنه،

```
//ES6
var calculateArea = function(height = 50, width = 60) {
    return width * height;
}

console.log(calculateArea()); //300
```

## ۳۰۹. **چی هستن؟ template-literal**

حروف الفبای الگو یا رشته‌های الگو، حروف الفبای رشته‌ای هستن که امکان عبارات تعبیه‌شده رو میدن. اینا به جای گیومه‌های دوتایی یا تکی با کاراکتر بک تیک (`) محصور میشن در ES6، این ویژگی استفاده از عبارات پویا رو به شرح زیر امکان پذیر میکنه.

```
const greeting = `Welcome to JS World, Mr. ${firstName}
${lastName}.`
```

توی ES5 لازمه که ما کدمون رو این شکلی بنویسیم.

```
var greeting = 'Welcome to JS World, Mr. ' + firstName + ' ' +
lastName.`
```

**نکته:** می‌تونیم از رشته‌های چند خطی و ویژگی‌های درون‌یابی رشته‌ای با الفبای الگو استفاده کنیم.

## ۳۱۰. چطوری رشته‌های چند خطی رو توی template-literal می‌نویسیم؟

در ES5، برای بدست آوردن رشته‌های چند خطی، باید از کاراکترهای فرار از خط جدید (\n) و نمادهای الحاقی (+) استفاده کنیم.

```
console.log('This is string sentence 1\n' +
'This is string sentence 2');
```

در حالی که در ES6، نیازی به ذکر کاراکتر دنباله خط جدید نیست،

```
console.log(`This is string sentence
'This is string sentence 2');
```

## ۳۱۱. الگوی تودرتو چی هستن؟

الگوی تودرتو یه ویژگیه که در نحو تحت لفظی الگو پشتیبانی می‌شه تا امکان بکتیکهای درونی تو یه مکان‌نمای \${ } رو در قالب فراهم کنه. برای مثال، الگوی تودرتو زیر برای نمایش نمادها بر اساس مجوزهای کاربر استفاده می‌شه، در حالی که الگوی بیرونی نوع پلت فرم رو بررسی می‌کنه.

```
const iconStyles = `icon ${ isMobilePlatform() ? '' :
`icon-${user.isAuthorized ? 'submit' : 'disabled'}` }`;
```

می‌تونیم مورد استفاده بالا رو بدون ویژگیهای الگوی تودرتو هم بنویسین. با این حال، ویژگی الگوی تودرتو فشرده‌تر و خواناتر است.

```
//Without nesting templates
const iconStyles = `icon ${ isMobilePlatform() ? '' :
(user.isAuthorized ? 'icon-submit' : 'icon-disabled') }`;
```

## ۳۱۲. الگوهای tagged-template چی هستن؟

الگوهای برچسب‌گذاری شده شکل پیشرفته‌ای از قالب‌ها هستن که تو اون برچسب‌ها بهمون اجازه میدن تا کلمات قالب رو با یه تابع تجزیه کنیم. تابع تگ اولین پارامتر رو به

عنوان آرایه ای از رشته‌ها و پارامترهای باقی مانده رو به عنوان strings می‌گیره. این تابع همچنین می‌توانه رشته‌های دستکاری شده رو بر اساس پارامترها برگردانه. بیاین نحوه استفاده از رفتار الگوی برچسب گذاری شده مجموعه مهارت‌های حرفه‌ای فناوری اطلاعات تویه سازمان رو ببینیم.

```

var user1 = 'John';
var skill1 = 'JavaScript';
var experience1 = 15;

var user2 = 'Kane';
var skill2 = 'JavaScript';
var experience2 = 5;

function myInfoTag(strings, userExp, experienceExp, skillExp) {
  var str0 = strings[0]; // "Mr/Ms. "
  var str1 = strings[1]; // " is a/an "
  var str2 = strings[2]; // "in"

  var expertiseStr;
  if (experienceExp > 10){
    expertiseStr = 'expert developer';
  } else if(skillExp > 5 && skillExp <= 10) {
    expertiseStr = 'senior developer';
  } else {
    expertiseStr = 'junior developer';
  }

  return
`${str0}${userExp}${str1}${expertiseStr}${str2}${skillExp}`;
}

var output1 = myInfoTag`Mr/Ms. ${ user1 } is a/an ${{
experience1 } in ${skill1}`;
var output2 = myInfoTag`Mr/Ms. ${ user2 } is a/an ${{
experience2 } in ${skill2}`;

console.log(output1); // Mr/Ms. John is a/an expert developer in
// JavaScript
console.log(output2); // Mr/Ms. Kane is a/an junior developer in
// JavaScript

```

رشته‌های خام چی هستن؟

ES6 با استفاده از روش «String.raw» ویژگی رشته‌های خام رو ارائه میکنه که برای دریافت شکل رشته خام رشته‌های الگو استفاده می‌شه. این ویژگی به ما این امکان رو میده تا به رشته‌های خام همونطور که وارد شده‌اند، بدون پردازش دنباله‌های فرار دسترسی داشته باشیم. بریم یه مثال در موردش بینیم،

```
var calculationString = String.raw `The sum of numbers is
\n${1+2+3+4}!`;
console.log(calculationString); // The sum of numbers is 10
```

اگه از رشته‌های خام استفاده نمی‌کنیم دنباله کاراکترهای خط جدید با نمایش خروجی در چندین خط پردازش می‌شه.

```
var calculationString = `The sum of numbers is \n${1+2+3+4}!`;
console.log(calculationString);
// The sum of numbers is
// 10
```

همچنین، ویژگی خام در اولین آرگومان تابع تگ موجود است

```
function tag(strings) {
  console.log(strings.raw[0]);
}
```

## ۳۱۴. کردن با **destructuring assign** چیه و چطوری انجام می‌شه؟

تخصیص **destructuring** به عبارت جاواسکریپتیه که امکان باز کردن مقادیر آرایه‌ها یا خصوصیات از اشیاء به متغیرهای مجزا رو فراهم میکنه. بیاین مقادیر ماه رو از یه آرایه با استفاده از تخصیص ساختارشکن به دست آوریم

```
var [one, two, three] = ['JAN', 'FEB', 'MARCH'];

console.log(one); // "JAN"
console.log(two); // "FEB"
console.log(three); // "MARCH"
```

و می‌تونیم ویژگی‌های کاربر یه آبجکت رو با استفاده از انتساب **destructuring** به دست بیاریم،

```
var {name, age} = {name: 'John', age: 32};

console.log(name); // John
console.log(age); // 32
```

## ۳۱۵. موقع **assign** کردن با **destructuring** چطوری می‌شه مقدار اولیه تعریف کرد؟

زمانی که مقدار باز شده از آرایه یا شیء در طول تخصیص ساختارشکن تعریف نشده باشد، می‌شه به يه متغیر يه مقدار پیش فرض اختصاص داد. این کمک میکنه تا از تنظیم مقادیر پیش فرض جداگانه برای هر انتساب جلوگیری کنیم. بیاین برای هر دو آرایه و موارد استفاده از شی مثالی بزنیم،

### :Arrays destructuring

```
var x, y, z;

[x=2, y=4, z=6] = [10];
console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

### :Objects destructuring

```
var {x=2, y=4, z=6} = {x: 10};

console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

## ۳۱۶. چطوری می‌تونیم مقدار یه آرایه رو با استفاده از **destructuring**-**assignment** تعویض کنیم؟

اگه از destructuring-assignment استفاده نمی‌کنیم تعویض دو مقدار به يه متغیر وقت نیاز داره. در حالی که با استفاده از يه ویژگی ساختارشکن، دو مقدار متغیر رو می‌شه

تو یه عبارت ساختار شکن جایگزین کرد. بیاین دو متغیر عددی رو در انتساب ساختار زدایی آرایه با هم عوض کنیم:

```
var x = 10, y = 20;

[x, y] = [y, x];
console.log(x); // 20
console.log(y); // 10
```

## ۳۱۷ Enhanced-object-literal چی هستن؟

ایجاد سریع اجسام با ویژگی‌های درون براکت رو آسان میکنه. برای مثال، نحو کوتاهتری رو برای تعریف ویژگی شی مشترک به شرح زیر ارائه میکنه.

```
//ES6
var x = 10, y = 20
obj = { x, y }
console.log(obj); // {x: 10, y:20}
//ES5
var x = 10, y = 20
obj = { x : x, y : y}
console.log(obj); // {x: 10, y:20}
```

## ۳۱۸ import چی هستن؟

واردات پویا با استفاده از نحو تابع «import» به ما اجازه میده تا ماثول‌ها رو در صورت تقاضا با استفاده از دستورات یا دستور async/await بارگذاری کنیم. در حال حاضر این ویژگی در [پیشنهاد مرحله 4] (<https://github.com/tc39/proposal-dynamic-import>) هستش. مزیت اصلی واردات پویا کاهش اندازه بسته‌های ما، پاسخ حجم/بار بار درخواست‌های ما و بهبود کلی در تجربه کاربر است.

```
import('./Module').then(Module => Module.method());
```

## ۳۱۹ کاربرد import چیه؟

در زیر بعضی از موارد استفاده از واردات پویا نسبت به واردات استاتیک آورده شده است.

۱. یه مازول رو به صورت درخواستی یا مشروط وارد کنیم. برای مثال، اگه می‌خواین  
یه `polyfill` رو در مرورگر قدیمی بارگذاری کنیم

```
if (isLegacyBrowser()) {
    import('...').
    .then(...);
}
```

۲. تعیین کننده مازول رو در زمان اجرا محاسبه کنیم. برای مثال می‌تونیم از اون برای  
گلوبال سازی استفاده کنیم.

```
import(`messages_${getLocale()}.js`).then(...);
```

۳. یه مازول رو از داخل یه اسکریپت معمولی به جای یه مازول وارد کنیم.

## ۳۲۰. آرایه‌های نوع‌دار (typed-arrays) چیه؟

آرایه‌های تایپ شده آبجکت‌هایی آرایه مانند از API 6 ECMAScript برای مدیریت داده‌های  
باپری هستن. جاوا‌سکریپت 8 نوع آرایه تایپ شده رو ارائه میده،

۱. آرایه ای از اعداد صحیح امضا شده 8 بیتی: `Int8Array`
۲. آرایه ای از اعداد صحیح امضا شده 16 بیتی: `Int16Array`
۳. آرایه ای از اعداد صحیح امضا شده 32 بیتی: `Int32Array`
۴. آرایه ای از اعداد صحیح بدون علامت 8 بیتی: `Uint8Array`
۵. آرایه ای از اعداد صحیح بدون علامت 16 بیتی: `Uint16Array`
۶. آرایه ای از اعداد صحیح بدون علامت 32 بیتی: `Uint32Array`
۷. آرایه ای از اعداد ممیز شناور 32 بیتی: `Float32Array`
۸. آرایه ای از اعداد ممیز شناور 64 بیتی: `Float64Array`

برای مثال، می‌تونیم یه آرایه از اعداد صحیح امضا شده 8 بیتی مانند زیر ایجاد کنیم

```
const a = new Int8Array();
// You can pre-allocate n bytes
const bytes = 1024
const a = new Int8Array(bytes)
```

## ۳۲۱. مزایای لودر ماژول‌ها چیه؟

ماژول لودر ویژگی‌های زیر را ارائه میده

.۱. Dynamic loading.

.۲. State isolation.

.۳. Global namespace isolation.

.۴. Compilation hooks.

.۵. Nested virtualization.

## ۳۲۲. collation چیه؟

برای مرتب سازی مجموعه‌ی رشته‌ها و جستجو در مجموعه‌ای از رشته‌ها استفاده می‌شده. این پارامتر توسط محلی و از Unicode آگاهه. بیان ویژگی‌های مقایسه و مرتب سازی را در نظر بگیریم،

:Comparison .۱

```
const list = [ "ä", "a", "z" ]; // In German, "ä" sorts with
                                "a" Whereas in Swedish, "ä" sorts after "z"
const l10nDE = new Intl.Collator("de");
const l10nSV = new Intl.Collator("sv");
console.log(l10nDE.compare("ä", "z") === -1); // true
console.log(l10nSV.compare("ä", "z") === +1); // true
```

:Sorting .۲

```
const list = [ "ä", "a", "z" ]; // In German, "ä" sorts with
                                "a" Whereas in Swedish, "ä" sorts after "z"
const l10nDE = new Intl.Collator("de");
const l10nSV = new Intl.Collator("sv");
console.log(list.sort(l10nDE.compare)) // [ "a", "ä", "z" ]
console.log(list.sort(l10nSV.compare)) // [ "a", "z", "ä" ]
```

## ۳۲۳. عبارت for...of چیه؟

دستور `for...of` یه حلقه تکرار بر روی اشیاء یا عناصر قابل تکرار مثل رشته داخلی، آرایه، اشیاء آرایه مانند (مثل آرگومان‌ها یا `Set`، `Map`، `TypedArray` و `NodeList`) و تکرارهای تعریف شده توسط کاربر ایجاد میکنه.

```
let arrayIterable = [10, 20, 30, 40, 50];

for (let value of arrayIterable) {
    value++;
    console.log(value); // 11 21 31 41 51
}
```

## ۳۲۴. خروجی عملگر `spread` روی آرایه زیر چیه؟

```
[... 'John Resig']
```

خروجی آرایه `[... 'John Resig']` است.

**توضیح:** رشته یه نوع تکرارپذیره و عملگر `spread` تو یه آرایه هر کاراکتر یه تکرارپذیر رو به یه عنصر نگاشت میکنه. از این رو، هر کاراکتر یه رشته به عنصری تو یه آرایه تبدیل می‌شه.

## ۳۲۵. آیا `PostMessage` امنه؟

بله، تا زمانی که برنامه‌نویس/توسعه‌دهنده مراقب منشأ و منبع پیام دریافتی باشد، `postMessages` رو می‌شه بسیار امن در نظر گرفت. اما اگه بخواییم پیامی رو بدون تأیید منبع آن ارسال یا دریافت کنیم حملات اسکریپت بین سایتی ایجاد می‌شه.

## ۳۲۶. مشکلات استفاده از `wildcard` روی `origin` چیه؟

آرگومان دوم متده `postMessage` مشخص میکنه که کدوم مبدأ مجاز به دریافت پیامه. اگه از علامت "\*" به عنوان آرگومان استفاده کنیم هر منبعی مجاز به دریافت پیامه. در این حالت، هیچ راهی برای پنجره فرستنده وجود نداره که بفهمه پنجره هدف در موقع ارسال

پیام در مبدأ هدف قرار داره یا نه. اگه پنجره هدف به مبدأ دیگری هدایت شه، مبدا دیگر داده‌ها رو دریافت میکنه. از این رو، این ممکنه منجر به آسیب پذیری‌های XSS شه.

```
targetWindow.postMessage(message, '*');
```

## ۳۲۷. چطوری از دریافت postMessage‌های ناخواسته و نامن از طرف هکرها جلوگیری کنیم؟

از اونجایی که شنونده به هر پیامی گوش میده، مهاجم میتونه برنامه رو با ارسال پیامی از مبدأ مهاجم فریب بده که این تصور رو ایجاد میکنه که گیرنده پیام رو از پنجره فرستنده واقعی دریافت کرده. میتونیم با اعتبارسنجی مبدا پیام در انتهای گیرنده با استفاده از ویژگی "message.origin" از این مشکل جلوگیری کنیم. برای مثال، اجازه بدین مبدا فرستنده `www.some-receiver.com` در سمت گیرنده `http://www.some-sender.com` (receiver.com-`(www.some`

```
//Listener on http://www.some-receiver.com/
window.addEventListener("message", function(message){
    if(/^http://www\.some-sender\.com$/.test(message.origin)){
        console.log('You received the data from valid sender',
        message.data);
    }
});
```

## ۳۲۸. میتوانیم کلا postMessage‌ها رو غیرفعال کنیم؟

نمیتوانیم به طور کامل (یا ۱۰۰٪) از postMessages استفاده نکنیم. حتی اگه برنامه‌مون با توجه به خطرات از postMessage استفاده نمیکنه، بسیاری از اسکریپتهای شخص ثالث از postMessage برای برقراری ارتباط با سرویس شخص ثالث استفاده میکنن. بنابراین ممکنه برنامه بدون اطلاع شما از postMessage استفاده کنه.

## ۳۲۹. آیا آنها به صورت synchronous و همزمان کار می‌کنن؟

در مرورگر IE8 synchronous postMessages هستن اما در IE9 و سایر مرورگرهای مدرن دیگر (یعنی IE9+, Firefox، Chrome، Safari) asynchronous postMessage برگردانده می‌شود. به دلیل این رفتار زمانی که callback برگردانده می‌شود، از مکانیزم asynchrone استفاده می‌کنیم.

## ۳۳۰. پارادیم زبان جاوااسکریپت چیه؟

جاوااسکریپت یه زبان چند پارادایم که از برنامه نویسی امری/روشی، برنامه نویسی شی گرا و برنامه نویسی تابعی پشتیبانی می‌کنه. جاوااسکریپت از برنامه نویسی شی گرا با وراثت اولیه پشتیبانی می‌کنه.

## ۳۳۱. تفاوت‌های بین جاوااسکریپت داخلی و خارجی چیه؟

**جاوااسکریپت داخلی:** کد منبع درون تگ اسکریپته.

**جاوااسکریپت خارجی:** کد منبع تو یه فایل خارجی (ذخیره شده با پسوند js) ذخیره می‌شود و در تگ ارجاع می‌شود.

## ۳۳۲. آیا جاوااسکریپت سریعتر از اسکریپت‌های سمت سرور است؟

بله، جاوااسکریپت سریعتر از اسکریپت سمت سروره. از اونجایی که جاوااسکریپت یه اسکریپت سمت کلاینته برای محاسبات یا محاسبات خودش به کمک سرور وب نیازی نداره. بنابراین جاوااسکریپت همیشه سریعتر از هر اسکریپت سمت سرور مانند ASP، PHP و غیره این.

## ۳۳۳. چطوری وضعیت چک بودن یه checkbox رو بدست بیاریم؟

می‌تونیم ویژگی checked رو در کادر انتخاب شده در DOM اعمال کنیم. اگه مقدار "True" باشه به این معنیه که چک باکس علامت زده شده در غیر این صورت علامت اونو برداریم.

برای مثال، عنصر چک باکس HTML زیر رو می‌شه با استفاده از جاواسکریپت به صورت زیر در دسترس قرار داد:

```
<input type="checkbox" name="checkboxname" value="Agree">
Agree the conditions<br>
```

```
console.log(document.getElementById('checkboxname').checked);
// true or false
```

## ۳۳۴. هدف از عملگر double-tilde چیه؟

عملگر دابل `~~` به عنوان عملگر `bitwise double NOT` شناخته می‌شه. این عملگر قراره جایگزین سریع تری برای `.Math.floor`

## ۳۳۵. چطوری یه کاراکتر رو به کد ASCII تبدیل کنیم؟

برای تبدیل کاراکترهای رشته به اعداد اسکی می‌تونیم از متدهای `String.prototype.charCodeAt` استفاده کنیم. برای مثال، بیاین کد ASCII رو برای حرف اول رشته «ABC» پیدا کنیم:

```
"ABC".charCodeAt(0) // returns 65
```

در حالی که روش `ASCII` اعداد رو به کاراکترهای `String.fromCharCode` برابر تبدیل می‌کنه.

```
String.fromCharCode(65, 66, 67); // returns 'ABC'
```

## ۳۳۶. ArrayBuffer چیه؟

یه گلاس `ArrayBuffer` برای نشون دادن یه بافر داده باینری خام عمومی با طول ثابت استفاده می‌شه. می‌تونیم اوно به صورت زیر ایجاد کنیم:

```
const buffer = new ArrayBuffer(16); // create a buffer of length 16
alert(buffer.byteLength); // 16
```

برای دستکاری یه `DataView` استفاده کنیم.

```
//Create a DataView referring to the buffer
const view = new DataView(buffer);
```

## ۳۳۷. خروجی کد زیر چی خواهد بود؟

```
console.log("Welcome to JS world"[0])
```

خروجی عبارت بالا "W" عه.

**توضیح:** نماد برآکت با شاخص خاص روی یه رشته کاراکتر رو تو یه مکان خاص برمی‌گردونه. از این رو، کاراکتر "W" رشته رو برمی‌گردونه. از اونجایی که این مورد در نسخه‌های IE7 و پایین‌تر پشتیبانی نمی‌شه، ممکنه لازم باشه از متدهای `charAt` برای به دست آوردن نتیجه دلخواه استفاده کنیم.

## ۳۳۸. هدف از Error-object چیه؟

کلاس `Error` یه آبجکت error ایجاد میکنه و نمونه‌هایی از آبجکت‌های خطای خطا موقع رخ دادن خطاهای زمان اجرا ارسال میشن، آبجکت `Error` همچنین میتوانه به عنوان یه آبجکت پایه برای استثناهای تعریف شده توسط کاربر استفاده شه. برای مثال

```
new Error([message[, fileName[, lineNumber]]])
```

می‌تونیم استثناهای یا خطاهای تعریف شده توسط کاربر رو با استفاده از آبجکت `Error` در بلوک `try...catch` مثل کد زیر ارسال کنیم.

```

try {
  if(withdraw > balance)
    throw new Error("Oops! You don't have enough balance");
} catch (e) {
  console.log(e.name + ': ' + e.message);
}

```

## ۳۳۹. هدف از EvalError-object چیه؟

آبجکت `EvalError` یه خطأ راجع به استفاده از تابع `eval` رو نشون میده. البته دیگه این استثنای توسط جاواسکریپت ایجاد نمی‌شه و آبجکت `EvalError` فقط برای سازگاری با نسخه‌های باقی مونده. برای مثال

```
new EvalError([message[, fileName[, lineNumber]]])
```

می‌تونیم `EvalError` رو با بلوک `try...catch` مثل کد زیر ارسال کنیم.

```

try {
  throw new EvalError('Eval function error', 'someFile.js',
100);
} catch (e) {
  console.log(e.message, e.name, e.fileName); // "Eval function error", "EvalError", "someFile.js"
}

```

## ۳۴۰. خطاهایی که در حالت strict-mode رخ میدن ولی در غیر اون وجود ندارن کدوما هستن؟

وقتی `use strict` رو اعمال می‌کنیم، `syntax`، بعضی از موارد زیر قبل از اجرای `SyntaxError` ایجاد می‌کنن ۱. وقتی از دستور `Octal` استفاده می‌کنیم

```
const n = 022;
```

۲. استفاده از عبارت `with`.

۳. وقتی از عملگر حذف روی نام متغیر استفاده می‌کنیم
۴. استفاده از eval یا آرگومان‌ها به عنوان متغیر یا نام آرگومان تابع
۵. موقعی که از کلمات کلیدی رزرو شده جدید استفاده می‌کنیم
۶. موقعی که یه تابع رو تو یه بلوک اعلام می‌کنیم

```
if (someCondition) { function f() {} }
```

از این رو، خطاهای موارد بالا برای جلوگیری از خطا در محیط‌های توسعه/تولید مفید هستند.

## ۱. ۳۴۱. آیا همه object دارای prototype هستند؟

خیر. همه object دارای prototype هستند به جز object پایه که توسط کاربر ایجاد می‌شود یا آبجکت‌ای که با استفاده از کلمه کلیدی new ایجاد می‌شود.

## ۲. ۳۴۲. تفاوت‌های بین argument و parameter چیه؟

نام متغیر تعریف یه تابعه در حالی که یه parameter نشون دهنده مقدار داده شده به تابع موقع فراخونای شدنش. بیاین این رو با یه تابع ساده توضیح بدیم

```
function myFunction(parameter1, parameter2, parameter3) {
  console.log(arguments[0]) // "argument1"
  console.log(arguments[1]) // "argument2"
  console.log(arguments[2]) // "argument3"
}
myFunction("argument1", "argument2", "argument3")
```

## ۳. ۳۴۳. هدف از متد some روی آرایه‌ها چیه؟

متد some برای تست این که حداقل یه عنصر در آرایه، از تست پیاده‌سازی شده توسط تابع ارائه شده، عبور می‌کنه یا نه استفاده می‌شود. متد یه مقدار بولین برمی‌گردونه. بیاین مثالی بزنیم تا هر عنصر رو بر اساس داشتن عدد زوج آزمایش کنیم:

```
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const odd = element => element % 2 !== 0;

console.log(array.some(odd)); // true (the odd element exists)
```

## ۳۴۴. چطوری دو یا تعداد بیشتری از آرایه‌ها رو با هم ترکیب کنیم؟

متدهای concat برای ترکیب دو یا چند آرایه با برگرداندن یه آرایه جدید حاوی تمام عناصر استفاده می‌شوند. مثال:

```
array1.concat(array2, array3, ..., arrayX)
```

بیاین مثالی از الحاق آرایه با آرایه‌های veggies و fruits رو مثال بزنیم.

```
const veggies = ["Tomato", "Carrot", "Cabbage"];
const fruits = ["Apple", "Orange", "Pears"];
const veggiesAndFruits = veggies.concat(fruits);
console.log(veggiesAndFruits); // Tomato, Carrot, Cabbage,
Apple, Orange, Pears
```

## ۳۴۵. تفاوت‌های بین Deep و Shallow کپی چیه؟

**کپی کم عمق:**

کپی کم عمق یه کپی بیتی از یه آبجکته. یه آبجکت جدید ایجاد می‌شوند که یه کپی دقیق از مقادیر موجود در آبجکت اصلی رو داره. اگه هر یه از فیلد‌های آبجکت ارجاع به آبجکت‌های دیگه باشه، فقط آدرس‌های مرجع کپی می‌شنوند یعنی فقط آدرس توی حافظه کپی می‌شوند.

**مثال**

```
const empDetails = {
  name: "John", age: 25, expertise: "Software Developer"
}
```

برای ایجاد یه نسخه تکراری

```
const empDetailsShallowCopy = empDetails //Shallow copying!
```

اگه مقداری از ویژگی رو تو یه تکراری به این صورت تغییر بدیم:

```
empDetailsShallowCopy.name = "Johnson"
```

دستور بالا همچنین نام `empDetails` رو تغییر میده، چون ما یه کپی کم عمق داریم. یعنی ما داده‌های اصلی رو هم از دست می‌دیم.

### کپی عمیق(Deep)

یه کپی عمیق همه فیلدها رو کپی میکنه و از حافظه تخصیص یافته به صورت پویا که توسط فیلدها به آن اشاره می‌شه کپی میکنه. یه کپی عمیق زمانی اتفاق می‌یافته که یه آبجکت همراه با پراپرتی‌هایی که به اون اشاره داره کپی شه.

### Example

```
const empDetails = {
  name: "John", age: 25, expertise: "Software Developer"
}
```

یه کپی عمیق با استفاده از خواص از آبجکت اصلی در متغیر جدید ایجاد کنیم:

```
const empDetailsDeepCopy = {
  name: empDetails.name,
  age: empDetails.age,
  expertise: empDetails.expertise
}
```

اگه `empDetailsDeepCopy` رو تغییر بدین، فقط `empDetailsDeepCopy.name` رو تغییر بدین، یعنی `empDetailsDeepCopy.name` و نه `empDetails` را تحت تأثیر قرار میده.

## ۳۴۶. چطوری می‌تونیم به یه تعداد مشخص از یه رشته کپی کنیم؟

متند `repeat` برای ساخت و برگرداندن یه رشته جدید استفاده می‌شه که حاوی تعداد مشخصی از کپی‌های رشته‌ای هس که روی اون فراخوانی شده و به هم پیوسته شدن. یادتون باشه که این روش به مشخصات ECMAScript 2015 اضافه شده است. بیاین یه مثال از یه رشته `Hello` رو برای تکرارش ۴ بار در نظر بگیریم:

```
'Hello'.repeat(4); // 'HelloHelloHelloHello'
```

## ۳۴۷. چطوری همه string های match شده با يه regular-expression را برگردانيم؟

متدها `matchAll` می‌تواند برای برگرداندن یه تکرارکننده از تمام نتایجی که یه رشته با يه عبارت منظم مطابقت دارن، استفاده شه. مثال زیر آرایه ای از نتایج رشته منطبق رو در برابر یه عبارت منظم برمی‌گردونه:

```
let regexp = /Hello(\d?)/g;
let greeting = 'Hello1Hello2Hello3';

let greetingList = [...greeting.matchAll(regexp)];

console.log(greetingList[0]); //Hello1
console.log(greetingList[1]); //Hello2
console.log(greetingList[2]); //Hello3
```

## ۳۴۸. چطوری يه رشته رو از اول یا از آخر trim کنيم؟

روش `trim` برای برش دادن دو طرف یه رشته استفاده می‌شه. اما اگه بخوايم بهخصوص در ابتدا یا انتهای رشته رو برش بدیم، می‌تونیم از روش‌های `trimEnd/trimRight` و `trimStart/trimLeft` استفاده کنيم. بیاین نمونه ای از اين روش‌ها رو در پیام `greeting` ببینيم:

```
const greeting = 'Hello, Goodmorning!  ';

console.log(greeting); // "Hello, Goodmorning! "
console.log(greeting.trimStart()); // "Hello, Goodmorning! "
console.log(greeting.trimLeft()); // "Hello, Goodmorning! "

console.log(greeting.trimEnd()); // "Hello, Goodmorning!"
console.log(greeting.trimRight()); // "Hello, Goodmorning!"
```

## ۳۴۹. خروجي کنسول زير با عملگر unary چي ميشد؟

```
console.log(+ 'Hello');
```

خروجی دستور `log` کنسول بالا `NaN` را برمی‌گردونه. از اونجا که عنصر توسط عملگر `unary` پیشونده و مفسر جاواسکریپت سعی میکنه اون عنصر رو به یه نوع عدد تبدیل کنه. از اونجایی که تبدیل با شکست مواجه می‌شه، مقدار عبارت به مقدار `NaN` منجر می‌شه.

## ۳۵۰. آیا جاواسکریپت از `mixin`‌ها استفاده میکنه؟

- یک اصطلاح برنامه نویسی شی گرا عمومیه: کلاسی که شامل متدهایی برای کلاس‌های دیگه اس. برخی از زبان‌های دیگر به ارت بردن چندگانه اجازه میدن. جاواسکریپت از وراثت چندگانه پشتیبانی نمیکنه، اما `Mixin`‌ها رو میشه با کپی کردن متدها در نمونه اولیه پیاده سازی کرد.

## ۳۵۱. تابع `thunk` چیه و چیکار میکنه؟

فقط تابعیه که ارزیابی مقدار رو به تأخیر می‌ندازه. هیچ آرگومانی نمی‌گیره، اما هر زمان که `thunk` رو فراخوانی می‌کنیم مقدار رو میده. برای مثال، از اون استفاده می‌شه که الان اجرا نشه، اما زمانی در آینده اجرا می‌شه. بیاین یه مثال `sync` بگیریم:

```
const add = (x,y) => x + y;
const thunk = () => add(2,3);
thunk() // 5
```

## ۳۵۲. `thunk` چیکار می‌کنن؟

ای از درخواست‌های شبکه مفید هستن. بیاین نمونه `Thunk` برای ایجاد درخواست‌های شبکه مفید هستن. بیاین نمونه ای از درخواست‌های شبکه رو ببینیم:

```

function fetchData(fn){
  fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then(response => response.json())
    .then(json => fn(json))
}

const asyncThunk = function (){
  return fetchData(function getData(data){
    console.log(data)
  })
}

asyncThunk()

```

تابع `fetchData` فوراً فراخونی نمی‌شه و تنها زمان فراخونی می‌شه که داده‌ها از نقطه پایانی API در دسترس باشن. تابع `setTimeout` هم برای ناهمزمان کردن کد ما استفاده می‌شه. بهترین مثال زمان واقعی، کتابخونه مدیریت حالت redux هس که از `thunk` ناهمزمان برای به تأخیر انداختن اعمال برای ارسال استفاده می‌کنه.

## ۳۵۳. خروجی فراخوانی‌های توابع زیر چی می‌شه؟

قسمت کد:

```

const circle = {
  radius: 20,
  diameter() {
    return this.radius * 2;
  },
  perimeter: () => 2 * Math.PI * this.radius
};

console.log(circle.diameter());
console.log(circle.perimeter());

```

خروجی:

خروجی 40 و NaN هس. یادتون باشه که `diameter` یه تابع منظمه در حالی که مقدار `arrow function` یه `arrow function` کلمه کلیدی `this` یه تابع معمولی (یعنی `diameter`) به محدوده اطراف که یه کلاسه (یعنی آبجکت `circle`) اشاره داره. در حالی که این کلمه کلیدی تابع `perimeter` به محدوده اطراف که یه آبجکت `window` هس اشاره

داره. از اونجایی که هیچ ویژگی radius در آبجکتهاي window وجود نداره، يه مقدار undefined برمی‌گردونه و ضرب مقدار، مقدار NaN را برمی‌گردونه.

## ۳۵۴. چطوری همه خطوط جدید رو از به رشته حذف کرد؟

ساده ترین روش استفاده از regex برای شناسایی و جایگزینی خطوط جدید توی رشته اس. در این حالت از تابع replace به همراه رشته برای جایگزینی استفاده می‌کنیم که در این مثال یه رشته خالیه:

```
function remove_linebreaks( var message ) {
    return message.replace( /[\r\n]+/gm, "" );
}
```

تو عبارت بالا g و m برای flag‌های سراسری و چند خطی هستن.

## ۳۵۵. تفاوت بین repaint و reflow چیه؟

repaint می‌زمانی اتفاق می‌افته که تغییراتی ایجاد می‌شه که روی دید یه عنصر تأثیر می‌ذاره، اما روی طرح اون تأثیر نمی‌ذاره. نمونه‌هایی از این موارد شامل طرح کلی، نمایان بودن یا رنگ پس زمینه اس. یه reflow شامل تغییراتیه که بر طرح بندی بخشی از صفحه (یا کل صفحه) تأثیر می‌ذاره. تغییر اندازه پنجره مرورگر، تغییر فونت، تغییر محتوا (مانند تایپ متن توسط کاربر)، استفاده از روش‌های جاواسکریپت شامل سبک‌های محاسبه‌شده، اضافه کردن یا حذف عناصر از DOM، و تغییر کلاس‌های یه عنصر چند مورد از مواردی هستن که می‌تونن جریان مجدد رو آغاز کنن. جریان مجدد یه عنصر باعث جریان مجدد بعدی همه عناصر فرزند و اجداد و همچنین هر عنصری که به دنبال اون توی DOM هس می‌شه.

## ۳۵۶. اگه قبل از یه آرایه عملگر نفی «!» بزاریم چی می‌شه؟

نفی یه آرایه با کاراکتر «!»، آرایه رو به یه بولین تبدیل می‌کنه. از اونجایی که آرایه‌ها در نظر گرفته می‌شن، پس نفی اون false رو برمی‌گردونه.

```
console.log(![]); // false
```

## ۳۵۷. اگه دو تا آرایه رو با هم جمع ببندیم چی می‌شه؟

اگه دو آرایه رو با هم اضافه کنیم هر دو اونا رو به رشته تبدیل می‌کنه و اونا رو به هم متصل می‌کنه. برای برمی‌یه مثال در موردش ببینیم:

```
console.log(['a'] + ['b']); // "ab"
console.log([] + []); // ""
console.log(![] + []); // "false", because ![] returns false.
```

## ۳۵۸. اگه عملگر جمع «+» روی قبل از مقادیر falsy قرار بدیم چی می‌شه؟

اگه عملگر (+) روی مقادیر نادرست (null, undefined, NaN, false) additive (+) را روی مقادیر falsy به مقدار عددی صفر تبدیل می‌شه. بیاین اونا رو توی کنسول مرورگر به صورت زیر نمایش بدیم:

```
console.log(+null); // 0
console.log(+undefined); // NaN
console.log(+false); // 0
console.log(+NaN); // NaN
console.log(+ ""); // 0
```

## ۳۵۹. چطوری با استفاده از آرایه‌ها و عملگرهای منطقی می‌تونیم رشته self رو تولید کنیم؟

رشته self رو می‌شه با ترکیب کاراکترهای ()!+[] تشکیل داد. برای رسیدن به این الگو باید موارد زیر رو بدونیم:

۱. از اونجایی که آرایه‌ها مقادیر true هستن، با نفی آرایه‌ها false تولید می‌شه: ![] == false

۲. طبق قوانین اجباری جاوا‌اسکریپت، اضافه کردن آرایه‌ها به هم اونا رو به رشته‌بندی تبدیل می‌کنه: []+[] == ""

۳. Prepend یه آرایه با عملگر + یه آرایه رو به نادرست تبدیل می‌کنه، انکار اونو درست

می‌کنه و در نهایت تبدیل نتیجه مقدار '1' رو تولید می‌کنه: +()+(!) == 1

با اعمال قوانین بالا می‌تونیم شرایط زیر رو استخراج کنیم:

```
![] + [] === "false"
+!+[] === 1
```

اکنون الگوی کاراکتر به صورت زیر ایجاد می‌شه:

s	e	l	f
~~~~~	~~~~~	~~~~~	~~~~~

```
( ![] + [] ) [3] + ( ![] + [] ) [4] + ( ![] + [] ) [2] + ( ![] + [] ) [0]
~~~~~ ~~~~~ ~~~~~ ~~~~~
( ![] + [] ) [+!+[ ]+!+[ ]+!+[ ] ] +
( ![] + [] ) [+!+[ ]+!+[ ]+!+[ ]+!+[ ] ] +
( ![] + [] ) [+!+[ ]+!+[ ] ] +
( ![] + [] ) [+[] ]
~~~~~ ~~~~~ ~~~~~ ~~~~~
( ![]+[] ) [+!+[ ]+!+[ ]+!+[ ]]+( ![]+[] ) [+!+[ ]+!+[ ]+!+[ ]]+( ![]+
[] ) [+!+[ ]+!+[ ]]+( ![]+[] ) [+[]]
```

## ۳۶۰. چطوری می‌تونیم مقادیر falsy را از آرایه حذف کنیم؟

می‌تونیم با وارد کدن Boolean به عنوان پارامتر، روش فیلتر را روی آرایه اعمال کنیم. به این ترتیب تمام مقادیر false، undefined، null و "" از آرایه حذف می‌شه.

```
const myArray = [false, null, 1, 5, undefined]
myArray.filter(Boolean); // [1, 5] // is same as
myArray.filter(x => x);
```

## ۳۶۱. چطوری مقادیر تکراری را از یه آرایه حذف کنیم؟

می‌تونیم مقادیر یونیک یه آرایه را با ترکیب دستور "Set" و (...)rest expression/spread دریافت کنیم.

```
console.log([...new Set([1, 2, 4, 4, 3])]); // [1, 2, 4, 3]
```

## ۳۶۲. چطوری هم‌زمان با **destructuring alias** کار می‌کنی؟

گاهی اوقات لازم داریم یه متغیر `destructure` شده با نامی متفاوت از نام ویژگی داشته باشیم. در این صورت، از یه `newName` برای تعیین اسم برای متغیر استفاده می‌کنیم. این فرآیند `destructuring alias` نامیده می‌شه.

```
const obj = { x: 1 };
// Grabs obj.x as as { otherName }
const { x: otherName } = obj;
```

## ۳۶۳. چطوری آیتم‌های یه آرایه رو بدون استفاده از متد **map** پیمایش کنیم؟

می‌تونیم مقادیر آرایه‌ها رو بدون استفاده از متد `map` تنها با استفاده از روش آرایه `from` ترسیم کنیم. بیاین نام شهرها رو از آرایه کشورها ترسیم کنیم:

```
const countries = [
  { name: 'India', capital: 'Delhi' },
  { name: 'US', capital: 'Washington' },
  { name: 'Russia', capital: 'Moscow' },
  { name: 'Singapore', capital: 'Singapore' },
  { name: 'China', capital: 'Beijing' },
  { name: 'France', capital: 'Paris' },
];
const cityNames = Array.from(countries, ({ capital }) =>
capital);
console.log(cityNames); // ['Delhi', 'Washington', 'Moscow',
'Singapore', 'Beijing', 'Paris']
```

## ۳۶۴. چطوری یه آرایه رو خالی کنیم؟

با صفر کردن `length` آرایه می‌تونیم به سرعت یه آرایه رو خالی کنیم:

```
let cities = ['Singapore', 'Delhi', 'London'];
cities.length = 0; // cities becomes []
```

## ۳۶۵. چطوری اعداد رو با تعداد رقم اعشار مشخص رند می‌کنی؟

می‌تونیم با استفاده از روش `toFixed` از جاواسکریپت، اعداد رو به تعداد معینی از اعشار گرد کنیم.

```
let pie = 3.141592653;
pie = pie.toFixed(3); // 3.142
```

## ۳۶۶. ساده‌ترین روش برای تبدیل آرایه به object چیه؟

می‌تونیم با استفاده از عملگر `(...)` یه آرایه رو به یه آبجکت با همون داده تبدیل کنیم.

```
var fruits = ["banana", "apple", "orange", "watermelon"];
var fruitsObject = {...fruits};
console.log(fruitsObject); // {0: "banana", 1: "apple", 2: "orange", 3: "watermelon"}
```

## ۳۶۷. چطوری یه آرایه با یه سری داده درست کنیم؟

می‌تونیم با استفاده از روش `fill` یه آرایه با مقداری داده یا یه آرایه با همون مقادیر ایجاد کنیم.

```
var newArray = new Array(5).fill("0");
console.log(newArray); // ["0", "0", "0", "0", "0"]
```

## ۳۶۸. متغیرهای موجود روی آبجکت `console` کدوما هستن؟

۱. ۰% - یه آبجکت رو می‌گیرد،
  ۲. ۵% - یه رشته می‌گیره،
  ۳. ۫% - برای اعشار یا عدد صحیح استفاده می‌شده
- این متغیرها رو می‌شه توی `console.log` به صورت زیر نشون داد

```
const user = { "name": "John", "id": 1, "city": "Delhi"};
console.log("Hello %s, your details %o are available in the
object form", "John", user); // Hello John, your details {name:
"John", id: 1, city: "Delhi"} are available in object
```

## ۳۶۹. می‌شه پیام‌های کنسول رو استایل دهی کرد؟

بله، می‌تونیم سبک‌های CSS رو برای پیام‌های کنسول مشابه متن html در صفحه وب اعمال کنیم.

```
console.log('%c The text has blue color, with large font and
red background', 'color: blue; font-size: x-large; background:
red');
```

متن به صورت زیر نمایش داده می‌شه

> console.log('%c Color of the text', 'color: blue; font-size: x-large; background:
red');

**Color of the text**

vendors~main.51281d83.chunk.js:

**نکته:** تمام سبک‌های CSS رو می‌شه برای پیام‌های کنسول اعمال کرد.

## ۳۷۰. هدف از متدهای `dir` روی آبجکت `console` چیه؟

برای نمایش یه لیست تعاملی از ویژگی‌های آبجکت جاواسکریپت مشخص شده به عنوان JSON استفاده می‌شه.

```
const user = { "name": "John", "id": 1, "city": "Delhi"};
console.dir(user);
```

آبجکت user نمایش داده شده توى حالت JSON

```
> const user = { "name": "John", "id": 1, "city": "Delhi"};
  console.dir(user);

▼ Object ⓘ
  name: "John"
  id: 1
  city: "Delhi"
  ► __proto__: Object
```

## ۳۷۱. آیا می‌شه المنهای HTML رو توی console دیباگ کرد؟

بله، دریافت و اشکال زدایی عناصر HTML توی کنسول، درست مثل بررسی عناصر، امکان پذیره.

```
const element = document.getElementsByTagName("body")[0];
console.log(element);
```

این عنصر HTML رو توی کنسول چاپ میکنه،

```
> const element = document.getElementsByTagName("body")[0];
< undefined
> console.log(element);
  ► <body class="question-page unified-theme">...</body>
< undefined
> |
```

## ۳۷۲. چطوری می‌شه داده‌ها رو به شکل جدولی توی console نمایش بدم؟

برای نمایش داده‌ها توی کنسول توی یه قالب جدولی برای تجسم آرایه‌ها یا آبجکت‌های پیچیده استفاده می‌شه.

```
const users = [{ "name": "John", "id": 1, "city": "Delhi"}, {
  "name": "Max", "id": 2, "city": "London"}, { "name": "Rod", "id": 3,
  "city": "Paris"}];
console.table(users);
```

### داده‌هایی که در قالب جدول مشاهده می‌شون

```
> const users = [{ "name": "John", "id": 1, "city": "Delhi"},  
  { "name": "Max", "id": 2, "city": "London"},  
  { "name": "Rod", "id": 3, "city": "Paris"}];  
< undefined  
> console.table(users);
```

VM92:1

(index)	name	id	city
0	"John"	1	"Delhi"
1	"Max"	2	"London"
2	"Rod"	3	"Paris"

▶ Array(3)

نکته: یادتون باشه `console.table` توی مرورگر IE پشتیبانی نمی‌شه 😞

### ۳۷۳. چطوری می‌شه بررسی کرد که یه پارامتر Number هست یا نه؟

ترکیبی از روش‌های `isNaN` و `isFinite` برای تأیید عدد بودن یا نبودن آرگومان استفاده می‌شه.

```
function isNumber(n){  
    return !isNaN(parseFloat(n)) && isFinite(n);  
}
```

### ۳۷۴. چطوری یه متن رو می‌تونیم به clipboard کپی کنیم؟

ما باید محتوا (با استفاده از روش `select`. عنصر ورودی رو انتخاب کنیم و دستور `copy` رو با `execCommand('copy')` اجرا کنیم (یعنی `execCommand('copy')`). همچنین می‌توانیم سایر دستورات سیستم مثل `cut` و `paste` رو اجرا کنیم.

```
document.querySelector("#copy-button").onclick = function() {  
    // Select the content  
    document.querySelector("#copy-input").select();  
    // Copy to the clipboard  
    document.execCommand('copy');  
};
```

### ۳۷۵. چطوری می‌شه timestamp رو بدست آورد؟

می‌توانیم از `Date.getTime` برای دریافت مهر زمانی فعلی استفاده کنیم. یه میانبر جایگزین برای دریافت مقدار وجود داره.

```
console.log(+new Date());
console.log(Date.now());
```

## ۳۷۶. چطوری یه آرایه چندسطحی رو تک سطحی کنیم؟

مسطح کردن آرایه‌های دو بعدی با عملگر `Spread` کاربرد نداره.

```
const biDimensionalArr = [11, [22, 33], [44, 55], [66, 77], 88, 99];
const flattenArr = [].concat(...biDimensionalArr); // [11, 22, 33, 44, 55, 66, 77, 88, 99]
```

اما ما می‌توانیم اونو با آرایه‌های چند بعدی با فراخوانی‌های `recursive` کال کنیم:

```
function flattenMultiArray(arr) {
  const flattened = [].concat(...arr);
  return flattened.some(item => Array.isArray(item)) ?
    flattenMultiArray(flattened) : flattened;
}
const multiDimensionalArr = [11, [22, 33], [44, [55, 66, [77, [88]], 99]]];
const flatArr = flattenMultiArray(multiDimensionalArr); // [11, 22, 33, 44, 55, 66, 77, 88, 99]
```

## ۳۷۷. ساده‌ترین روش برای بررسی چندشرطی چیه؟

می‌توانیم از تابع `indexOf` « برای مقایسه ورودی با چندین مقدار به جای بررسی هر مقدار به عنوان یه شرط استفاده کنیم.

```
// Verbose approach
if (input === 'first' || input === 1 || input === 'second' ||
input === 2) {
    someFunction();
}
// Shortcut
if (['first', 1, 'second', 2].indexOf(input) !== -1) {
    someFunction();
}
```

## ۳۷۸. چطوری کلیک روی دکمه برگشت مرورگر رو متوجه بشیم؟

روش `window.onbeforeunload` برای ضبط رویدادهای دکمه بازگشت مرورگر استفاده می‌شود. این برای هشدار دادن به کاربرها در مورد از دست دادن داده‌های فعلی می‌توانه استفاده. `syntax` تعریف کردنش به صورت زیر است:

```
window.onbeforeunload = function() {
    alert("You work will be lost");
};
```

## ۳۷۹. چطوری می‌توانیم کلیک راست رو غیرفعال کنیم؟

کلیک راست روی صفحه رو می‌شه با برگرداندن `false` از ویژگی `oncontextmenu` در تگ `body` غیرفعال کرد.

```
<body oncontextmenu="return false;">
```

## ۳۸۰. object-hا چی هستن؟

مقادیر اولیه مانند رشته، عدد و بولین پرایپری و متدهای ندارن، اما زمانی که می‌خوایم کارهایی رو روی اونا انجام بدیم، به طور موقت به یه آبجکت (Object) تبدیل می‌شون. برای مثال، اگه متدهای `UpperCase` رو روی یه مقدار رشته اولیه اعمال کنیم خطایی ایجاد نمی‌کنه، اما حروف بزرگ رشته رو برمی‌گردونه.

```
let name = "john";

console.log(name.toUpperCase()); // Behind the scenes treated
as console.log(new String(name).toUpperCase());
```

یعنی هر اولیه به جز null و undefined دارای هس و لیست اشیاء BigInt و String، Number، Boolean، Symbol wrapper عبارتند از wrapper

## ۳۸۱. **AJAX چیه؟**

مخفف **Ajax** هس و گروهی از فناوری‌های مرتبط **Asynchronous JavaScript XML** (HTML، CSS، JavaScript، XMLHttpRequest API) که برای نمایش داده‌ها به صورت ناهمزمان استفاده می‌شود. یعنی ما می‌توانیم داده‌ها را به سرور ارسال کنیم و بدون بارگیری مجدد صفحه وب، داده‌ها را از سرور دریافت کنیم.

## ۳۸۲. **روش‌های مختلف مدیریت یه کد Asynchronous چیه؟**

۱. **callback** ها
۲. **Promise** ها
۳. **Async/await**
۴. کتابخونه‌های شخص ثالث مانند **async.js**، **bluebird** و غیره

## ۳۸۳. **چطوری یه درخواست fetch رو کنسل کنیم؟**

یکی از ضعف‌های **Promise**‌ها اینه که native راه مستقیمی برای لغو درخواست **fetch** نیست. اما **AbortController** جدید از مشخصات js به ما امکان میده که از یه سیگنال واسه لغو یک یا چند درخواست **fetch** استفاده کنیم. جریان اصلی لغو یه درخواست **fetch** اینجوری می‌شود. ۱. یه کلاس **AbortController** ایجاد کنیم

۲. ویژگی سیگنال اون آبجکت ساخته شده رو دریافت کنیم و سیگنال رو به عنوان یه option به متده استفاده می‌کنن با ویژگی fetch ارسال کنیم

۳. برای لغو تمام fetch‌هایی که از اون سیگنال استفاده می‌کنن با ویژگی abort از AbortController رو فراخوانی کنیم.

برای مثال، بیاین یه سیگنال رو به چندین درخواست fetch ارسال کنیم، مهم درخواست‌ها با اون سیگنال لغو می‌شن.

```
const controller = new AbortController();
const { signal } = controller;

fetch("http://localhost:8000", { signal }).then(response => {
  console.log(`Request 1 is complete!`);
}).catch(e => {
  if(e.name === "AbortError") {
    // We know it's been canceled!
  }
});

fetch("http://localhost:8000", { signal }).then(response => {
  console.log(`Request 2 is complete!`);
}).catch(e => {
  if(e.name === "AbortError") {
    // We know it's been canceled!
  }
});

// Wait 2 seconds to abort both requests
setTimeout(() => controller.abort(), 2000);
```

## ۳۸۴. Speech-API چیه؟

API گفتار وب برای فعال کردن مرورگرهای مدرن برای شناسایی و ترکیب گفتار (یعنی داده‌های صوتی در برنامه‌های وب) استفاده می‌شود. این API توسط انجمن W3C در سال 2012 معرفی شد و دارای دو بخش اصلیه.

۱. **تشخیص گفتار (تشخیص گفتار ناهمزمان یا گفتار به متن):** این امکان رو فراهم می‌کند که زمینه صدا رو از ورودی صوتی تشخیص داده و به اون پاسخ بدین. این توسط رابط "SpeechRecognition" قابل دسترسی‌بود.

مثال زیر نحوه استفاده از این API برای دریافت متن از گفتار رو نشون میده.

```

window.SpeechRecognition = window.webkitSpeechRecognition ||  

window.SpeechRecognition; // webkitSpeechRecognition for  

Chrome and SpeechRecognition for FF  

const recognition = new window.SpeechRecognition();  

recognition.onresult = (event) => { // SpeechRecognitionEvent  

type  

const speechToText = event.results[0][0].transcript;  

console.log(speechToText);  

}  

recognition.start();

```

در این API، مرورگر برای استفاده از میکروفون کاربر ارش اجازه می‌خواهد. **۲. SpeechSynthesis (Text-to-Speech)**: این امکان را فراهم میکنند تا یه متن رو به صدا تبدیل کنیم و به وسیله "SpeechSynthesisAPI" قابل دسترسیه. برای مثال، کد زیر برای دریافت صدا/گفتار از متن استفاده می‌شه.

```

if('speechSynthesis' in window){  

    const speech = new SpeechSynthesisUtterance('Hello  

World!');  

    speech.lang = 'en-US';  

    window.speechSynthesis.speak(speech);  

}

```

نمونه‌های بالا رو می‌شه روی کنسول برنامه‌نویس مرورگر کروم (+33) تست کرد.  
**توجه:** این API هنوز یه پیش‌نویس فعله و فقط روی مرورگرهای کروم و فایرفاکس وجود دارد(البته کروم فقط مشخصات رو اجرا میکنه)

## ۳۸۵. حداقل timeout توی throttling چقدر؟

هم مرورگر و هم محیط‌های جاواسکریپت NodeJS با حداقل تاخیری که بیشتر از 0 میلی ثانیه اس throttles رو انجام میدن. یعنی حتی اگه تنظیم یه تاخیر 0ms به طور آنی اتفاق نیوفته.

**مرورگرها:** حداقل 4 میلی ثانیه تاخیر دارن. این throttles زمانی اتفاق می‌یافته که تماس‌های متوالی به دلیل تودرتوی Callback (عمق معین) یا پس از تعداد معینی فواصل متوالی آغاز شه.

**توجه:** مرورگرهای قدیمی حداقل 10 میلی ثانیه تاخیر دارن.  
**Nodejs:** حداقل 1ms تاخیر دارن. این throttles زمانی اتفاق می‌یافته که تاخیر بزرگتر از

2147483647 یا کمتر از 1 باشد.

بهترین مثال برای توضیح این رفتار throttling وقفه، ترتیب قطعه کد.

```
function runMeFirst() {
    console.log('My script is initialized');
}
setTimeout(runMeFirst, 0);
console.log('Script loaded');
```

و خروجی

```
Script loaded
My script is initialized
```

اگه از «`setTimeout`» استفاده نمی‌کنیم ترتیب گزارش‌ها این شکلی می‌شه.

```
function runMeFirst() {
    console.log('My script is initialized');
}
runMeFirst();
console.log('Script loaded');
```

و خروجی

```
My script is initialized
Script loaded
```

## ۳۸۶. چطوری می‌شه یه timeout صفر توی مرورگر اجرا کرد؟

به دلیل حداقل تاخیر بیش از 0 میلی ثانیه، نمی‌توانیم از `setTimeout(fn, 0)` برای اجرای فوری کد استفاده کنیم، اما برای دستیابی به این رفتار می‌توانیم از `window.postMessage()` استفاده کنیم.

## ۳۸۷. task‌ها توی event-loop چی هستن؟

وظیفه هر کد/برنامه جاوااسکریپتیه که قراره توسط مکانیسم‌های استانداره اجرا شه، مثل شروع اولیه اجرای یه برنامه، اجرای یه رویداد، callback یا یه بازه زمانی یا وقفه در حال

- اجرا. همه این وظایف تو یه صف کار برنامه ریزی میشن موارد استفاده برای افزودن وظایف به صف کار اینا هستن.
۱. موقعی که یه برنامه جاوااسکریپت جدید مستقیماً از کنسول اجرا میشه یا توسط عنصر `<script>` اجرا میشه، وظیفه به صف کار اضافه میشه.
  ۲. موقعی که یه رویداد فراخونی میشه، callback رویداد به صف کار اضافه میشه
  ۳. وقتی به یه `setInterval` یا `setTimeout` رسید، callback مربوطه به صف کار اضافه میشه

## ۳۸۸. **microtask‌ها چی هستن؟**

کد جاوااسکریپته که باید بلافصله پس از تکمیل task/Microtask در حال اجرا اجرا شه. اونا در واقع به نوعی مسدود کننده هستن. یعنی تا زمانی که صف microtask خالی نشه، رشته اصلی مسدود میشه.

منابع اصلی Microtask‌ها عبارتند از `Promise.resolve`, `Promise.reject`, `MutationObservers`, `IntersectionObservers` و غیره.

**توجه:** همه این Microtask‌ها توی همون چرخش event-loop پردازش میشن.

## ۳۸۹. **event-loop‌های مختلف کدام هستن؟**

- این حلقه اصلی یه برنامه اس. به طور معمول این تو پایین صفحه اصلیه. خروج معمولاً نشون دهنده تمایل به بسته شدن برنامه اس. توی هر برنامه فقط یکی از این موارد میتونه وجود داشته باشه.

- این حلقه ایه که معمولاً تو پایین رویه اصلی یه UI یافت میشه.

## ۳۹۰. **هدف از queueMicrotask چیه؟**

صف میکروتسک به کد اجازه میده بدون تداخل با کدهای دیگه که در حالت تعليق هستن، با اولویت بالاتری اجرا بشه.

## ۳۹۱. چطوری می‌شه از کتابخونه‌های جاوااسکریپت توی فایل استفاده کرد؟

مشخصه که همهی کتابخونه‌ها یا چارچوب‌های جاوااسکریپت دارای فایل‌های اعلان TypeScript نیستن. اما اگه هنوزم می‌خواین از کتابخونه‌ها یا فریمورک‌ها تو فایل‌های بدون دریافت خطاهای کامپایل استفاده کنیم تنها راحل کلمه کلیدی TypeScript به همراه یه اعلان متغیره. برای مثال، بیاین تصور کنیم که یه کتابخونه به نام `declare customLibrary` دارید که اعلان TypeScript نداره و فضای نامی به نام "customLibrary" توی فضای نام گلوبال داره. می‌تونیم از این کتابخونه توی کد تایپ‌اسکریپت به صورت زیر استفاده کنیم.

```
declare var customLibrary;
```

در زمان اجرا، تایپ‌اسکریپت نوع اونو به متغیر `customLibrary` به صورت any ارائه می‌کنه. جایگزین دیگه بدون استفاده از کلمه کلیدی `declare` رو تو مثال زیر ببینیم:

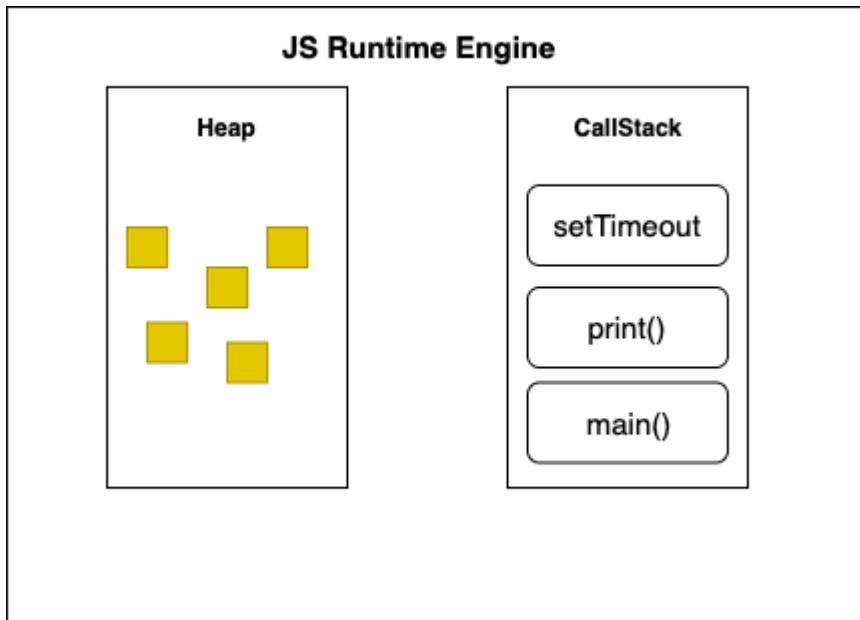
```
var customLibrary: any;
```

## ۳۹۲. تفاوت‌های بین observable‌ها و Promise‌ها کداما هستن؟

observable‌ها	Promise‌ها
چندین مقدار رو تو یه دوره زمانی منتشر می‌کنه (جريان مقادیری از 0 تا چندگانه)	فقط یه مقدار رو تو یه زمان منتشر می‌کنه
اونا برای فراخوانی نیاز به اشتراک دارن	قراره فوراً فراخوانی شن
observable‌ها می‌توون همزمان یا ناهمزمان	همیشه ناهمزمانه Promise حتی اگه بلافضله حل شه
اپراتورهایی مانند map, forEach, filter, reduce, retryWhen و retry و غیره رو ارائه میده.	هیچ اپراتور ارائه نمیده
با استفاده از روش unsubscribe لغو می‌شن	قابل لغو نیست

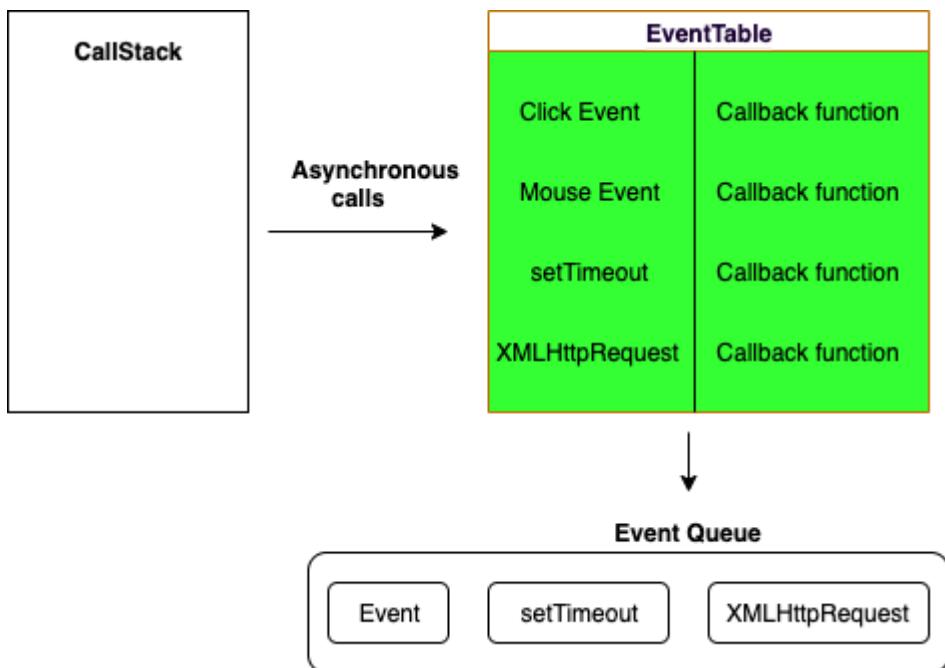
## چیه؟ heap .۳۹۱۳

(memory heap) محلیه که تو اون آبجکت‌ها موقع تعریف متغیرها ذخیره می‌شن یعنی این محلیه که تمام تخصیص حافظه و عدم تخصیص تو اون انجام می‌شه. هر دو call-stack و heap دو ظرف زمان اجرا JS هستن. هر زمان که زمان اجرا با متغیرها و اعلان‌های تابع در کد مواجه می‌شه، اونا رو توی Heap ذخیره می‌کنه.



## چیه؟ event-table .۳۹۱۴

**Event-Table** یه ساختار داده‌ایه که تمام رویدادهایی رو که به صورت ناهمزمان اجرا می‌شن، مثل بعد از مدتی فاصله زمانی یا پس از رفع بعضی از درخواست‌های API، ذخیره و ردیابی می‌کنه. یعنی هر زمان که یه تابع `setTimeout` رو فراخوانی کنیم یا عملیات `async` رو فراخوانی کنیم به جدول رویداد اضافه می‌شه. توابع رو به تنها‌یی اجرا نمی‌کنه. هدف اصلی جدول رویدادها پیگیری رویدادها و فرستادنشون به صف رویداد همونطور که توی نمودار زیر می‌بینیم.



## ۳۹۵. صف **microTask** چیه؟

صف **Microtask Queue** جدیدیه که تو اون تمام وظایف آغاز شده توسط آبجکت **Promise** قبل از صف برگشت پردازش میشن **microtask** از کارهای رندر و نقاشی بعدی پردازش میشه. اما اگه این ریزکارها برای مدت طولانی اجرا شن، منجر به مشکل بصری میشن.

## ۳۹۶. تفاوت بین **polyfill** و **shim** چیه؟

کتابخونه‌ایه که یه API جدید رو با استفاده از ابزارهای اون محیط به یه محیط قدیمی تر میاره. لزوماً محدود به یه برنامه وب نیست. برای مثال، es5-shim.js برای شبیه سازی **ویژگی‌های ES5** توی مرورگرهای قدیمی (عمدتاً قبل از IE9) استفاده میشه. در حالی که **polyfill** یه قطعه کد (یا افزونه) اس که فناوری رو ارائه میکنه که شما توسعه‌دهنده، از مرورگر انتظار دارید که به صورت بومی ارائه کنه. تو یه جمله ساده، **API**‌های مرورگر. **shim** یه **polyfill**.

## ۳۹۷. چطوری متوجه primitive یا غیر primitive بودن یه نوع داده میشیم؟

در جاوااسکریپت، دیتای primitive عبارتند از `boolean`, `string`, `number`, `undefined`, `BigInt`, `null`, `Symbol` و `Object`ها می‌شوند. اما با تابع زیر می‌توانیم به راحتی اونا رو شناسایی کنیم

```
const myPrimitive = 30;
const myNonPrimitive = {};
function isPrimitive(val) {
    return Object(val) !== val;
}

isPrimitive(myPrimitive);
isPrimitive(myNonPrimitive);
```

اگه مقدار یه نوع داده اولیه باشه، سازنده `Object` یه آبجکت wrapper جدید برای مقدار `Object` میکنه. اما اگه مقدار یه نوع داده non-primitive (یک آبجکت) باشه، سازنده همون آبجکت رو میده.

## ۳۹۸. آیا babel چیه؟

Babel یه ترانسپایلر جاوااسکریپت برای تبدیل کد ECMAScript 2015 + به یه نسخه سازگار جاوااسکریپت تو مرورگرها یا محیط‌های فعلی و قدیمی تره. که میشه موارد زیر رو درموردش نوشت:

۱. تبدیل syntax
۲. ویژگی‌های Polyfill که در محیط هدف شما وجود نداره (با استفاده از `(babel/polyfill@`)
۳. تبدیل کد منبع

## ۳۹۹. آیا Node.js به شکل کامل تک thread کار میکنه؟

Node یه رشته است، اما بعضی از توابع موجود توی کتابخونه استانداره Node.js (برای مثال، توابع ماژول `fs`) تک رشته‌ای نیستن. یعنی منطق اونا خارج از رشته Node اجرا

می‌شه تا سرعت و عملکرد یه برنامه رو بهبود بخشد.

## ۴۰۰. کاربردهای مرسوم observable ها کدوما هستن؟

بعضی از رایج‌ترین موارد استفاده اش، سوکت‌های وب با اعلان‌ها، تغییرات ورودی کاربر، فواصل تکراری و غیره اس.

## ۴۰۱. RxJS چیه؟

RxJS (افزونه‌های واکنش‌گرا برای جاواسکریپت) کتابخونه‌ای برای پیاده‌سازی برنامه‌نویسی واکنش‌گرا با استفاده از `observables` که نوشتن کد ناهمزمان یا مبتنی بر تماس رو آسان‌تر می‌کنه. همچنین توابع کاربردی رو برای ایجاد و کار با مشاهده پذیرها فراهم می‌کنه.

## ۴۰۲. تفاوت بین function-declaration و Function-constructor چیه؟

توابعی که با `Function-constructor` ایجاد می‌شن، برای زمینه‌های ایجاد خود بسته ایجاد نمی‌کنن، اما همیشه در محدوده جهانی ایجاد می‌شن یعنی تابع فقط می‌توانه به متغیرهای محلی خود و متغیرهای دامنه جهانی دسترسی داشته باشه. در حالی که اعلان‌های تابع می‌توانن به متغیرهای تابع بیرونی (بسته شدن) هم دسترسی داشته باشن.

بیاین این تفاوت رو با یه مثال ببینیم،

### :Function Constructor

```
var a = 100;
function createFunction() {
  var a = 200;
  return new Function('return a;');
}
console.log(createFunction()()); // 100
```

### :Function declaration

```

var a = 100;
function createFunction() {
  var a = 200;
  return function func() {
    return a;
  }
}
console.log(createFunction()()); // 200

```

## ۴۰۳. شرط Short-circuit یا اتصال کوتاه چیه؟

شرایط اتصال کوتاه برای روش خلاصه شده نوشتمن دستورات if ساده استفاده میشه. بیاین سناریو رو با استفاده از يه مثال نشون بدیم. اگه بخوایم وارد پورتالی با شرایط احراز هویت بشیم، کد جاوااسکریپتون به این شکل نوشته میشه:

```

if (authenticate) {
  loginToPorta();
}

```

از اونجایی که عملگرهای منطقی جاوااسکریپت از چپ به راست ارزیابی میشن عبارت بالا رو میشه با استفاده از عملگر منطقی `&&` ساده کرد.

```
authenticate && loginToPorta();
```

## ۴۰۴. ساده‌ترین روش برای تغییر سایز یه آرایه چیه؟

ویژگی `length` یه آرایه برای تغییر اندازه یا خالی کردن سریع آرایه اس. بیاین ویژگی رو روی آرایه اعداد اعمال کنیم تا تعداد عناصر رو از 5 به 2 تغییر بدیم:

```

const array = [1, 2, 3, 4, 5];
console.log(array.length); // 5

array.length = 2;
console.log(array.length); // 2
console.log(array); // [1,2]

```

و آرایه رو هم می‌شه خالی کرد:

```
const array = [1, 2, 3, 4, 5];
array.length = 0;
console.log(array.length); // 0
console.log(array); // []
```

## ۱۰۵ observable چیه؟

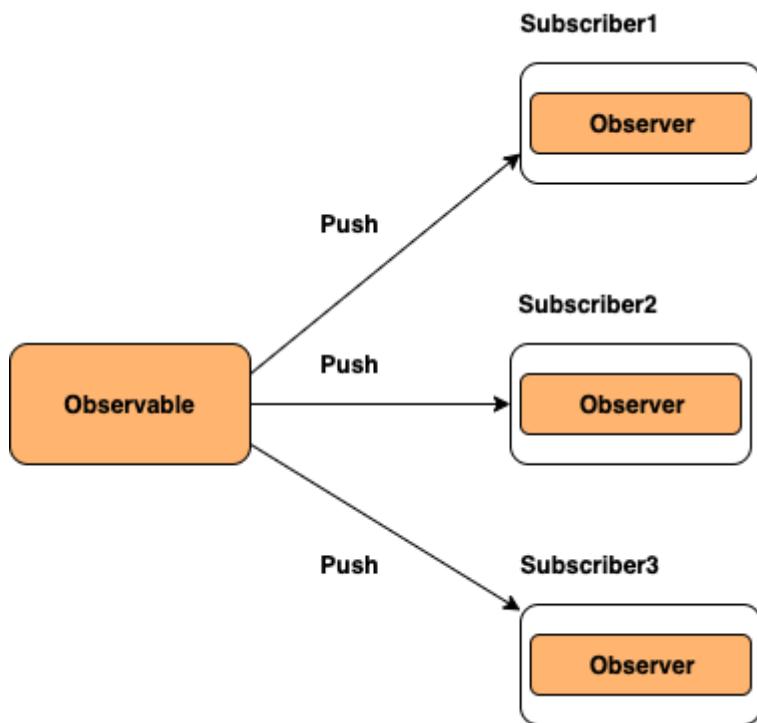
اساساً Observable تابعیه که می‌تونه جریانی از مقادیر رو به صورت همزمان یا ناهمزمان به یه ناظر در طول زمان برگردونه. مصرف کننده می‌تونه با فراخوانی متده subscribe مقدار رو دریافت کنه.

بیاین به یه مثال ساده از یه Observable نگاه کنیم:

```
import { Observable } from 'rxjs';

const observable = new Observable(observer => {
  setTimeout(() => {
    observer.next('Message from a Observable!');
  }, 3000);
});

observable.subscribe(value => console.log(value));
```



**توجه:** Observable‌ها هنوز بخشی از زبان جاواسکریپت نیستن اما پیشنهاد شده که به زبان اضافه بشن.

## ۴۰۶. تفاوت‌های بین تابع و کلاس‌ها چیه؟

تفاوت اصلی بین اعلان‌های تابع و اعلان‌های کلاس `hoisting` هس. اعلان‌های تابع

میشن `hoist` اما اعلان‌های کلاس نه.

**:Classes**

```
const user = new User(); // ReferenceError
class User {}
```

## :Constructor Function

```
const user = new User(); // No error
function User() {
}
```

## تابع `async` چیه؟ ۴۰۷

یه تابع `async` تابعیه که با کلمه کلیدی `async` اعلام شده که با اجتناب از زنجیره پرامیس رفتار ناهمزمان و مبتنی بر قول به سبک تمیزتری نوشته شه. این توابع میتوانن شامل صفر یا بیشتر عبارت `await` باشن.  
بیاین یه مثال تابع همگام زیر رو در نظر بگیریم،

```
async function logger() {  
  
  let data = await fetch('http://someapi.com/users'); // pause  
  until fetch returns  
  console.log(data)  
}  
logger();
```

این اساساً خوبی نحوی بیش از پرامیس‌ها و توابع جنریتور ES2015 هست.

## چطوری خطاهای ایجاد شده هنگام استفاده از `Promise`‌ها رو کنترل کنیم؟ ۴۰۸

موقع استفاده از کدهای `async` و ناهمزمان، `Promise`‌های ES6 میتوانن زندگی برنامه‌نویس رو بدون داشتن ترس از `callback` و مدیریت خطاهای آسان‌تر می‌کنن. اما `Promise`‌ها مشکلاتی دارن و بزرگ‌ترین اویا به طور بیش‌فرض مخفی کردن خطاهاس. فرض کنیم انتظار دارین برای تمام موارد زیر یه خطای توی کنسول چاپ بشه.

```
Promise.resolve('promised value').then(function() {  
  throw new Error('error');  
});  
  
Promise.reject('error value').catch(function() {  
  throw new Error('error');  
});  
  
new Promise(function(resolve, reject) {  
  throw new Error('error');  
});
```

اما خیلی از محیط‌های جاواسکریپت مدرن وجود دارن که هیچ خطایی رو چاپ نمی‌کنن. می‌تونیم این مشکل رو به روش‌های مختلف حل کنیم.

۱. اضافه کردن یه بلوک **catch** به اخر هر زنجیره: ما میتوانیم یه بلوک **catch** به آخر زنجیره پرمیس‌هایمان اضافه کنیم

```
Promise.resolve('promised value').then(function() {
    throw new Error('error');
}).catch(function(error) {
    console.error(error.stack);
});
```

اما تایپ کردن برای هر زنجیره پرمیس‌ها و پرماخاب هم خیلی سخته.

۲. اضافه کردن متدهای **done**: میتوانیم ابتدا راه حل‌ها رو جایگزین کنیم و بعد با روش انجام شده بلوک‌ها رو بگیریم

```
Promise.resolve('promised value').done(function() {
    throw new Error('error');
});
```

فرض کنیم میخواهیم داده‌ها رو با استفاده از **HTTP fetch** کنیم و بعداً پردازش داده‌های حاصل رو به صورت ناهمزمان انجام بدیم. میتوانیم بلوک **done** رو به صورت زیر بنویسیم.

```
getDataFromHttp()
    .then(function(result) {
        return processDataAsync(result);
    })
    .done(function(processed) {
        displayData(processed);
    });
});
```

در آینده، اگه API کتابخونه پردازش به همگام تغییر کنه، میتوانیم بلوک **done** رو حذف کنیم.

```
getDataFromHttp()
    .then(function(result) {
        return displayData(processDataAsyn(result));
    })
});
```

سپس فراموش کردیم که بلوک «انجام شد» رو به بلوک **then** اضافه کنیم که منجر به خطاهای خاموش میشه.

### ۳. Extend ES6 Promises by Bluebird

Bluebird-API میاد **Promise**‌های اکما اسکریپت رو گسترش میده تا در راه حل دوم مشکلی ایجاد نشه. این کتابخونه به کنترل کننده "پیش فرض" در

Rejection هس که تمام خطاهای را از Promises رد شده به stderr چاپ میکنه. پس از نصب، می‌توانیم ردهای کنترل نشده را پردازش کنیم

```
Promise.onPossiblyUnhandledRejection(function(error){
    throw error;
});
```

یه reject را انجام بدین فقط با یه catch خالی اونو مدیریت کنیم

```
Promise.reject('error value').catch(function() {});
```

## ۴۰۹. Deno چیه؟

یه ران تایم (run-time) ساده، مدرن و ایمن برای جاوااسکریپت و تایپ‌اسکریپتیه که از موتور جاوااسکریپت V8 و زبان برنامه نویسی Rust استفاده میکنه و توسط رایان دال، خالق نود جی اس استارت توسعه‌اش زده شده.

## ۴۱۰. توی جاوااسکریپت چطوری یه object قابل پیمایش درست کنیم؟

به طور پیش فرض، آبجکتها argument ساده قابل تکرار نیستن. اما می‌توانیم با تعریف ویژگی «Symbol.iterator» روی اون شی رو قابل تکرار کنیم. بیاین این رو با یه مثال نشون بدیم،

```

const collection = {
  one: 1,
  two: 2,
  three: 3,
  [Symbol.iterator]() {
    const values = Object.keys(this);
    let i = 0;
    return {
      next: () => {
        return {
          value: this[values[i++]],
          done: i > values.length
        }
      }
    };
  }
};

const iterator = collection[Symbol.iterator]();

console.log(iterator.next());    // → {value: 1, done: false}
console.log(iterator.next());    // → {value: 2, done: false}
console.log(iterator.next());    // → {value: 3, done: false}
console.log(iterator.next());    // → {value: undefined, done:
true}

```

فرآیند بالا رو می‌شه با استفاده از یه تابع مولد ساده کرد:

```

const collection = {
  one: 1,
  two: 2,
  three: 3,
  [Symbol.iterator]: function * () {
    for (let key in this) {
      yield this[key];
    }
  }
};
const iterator = collection[Symbol.iterator]();
console.log(iterator.next());    // {value: 1, done: false}
console.log(iterator.next());    // {value: 2, done: false}
console.log(iterator.next());    // {value: 3, done: false}
console.log(iterator.next());    // {value: undefined, done:
true}

```

## ۴۱۱. روش مناسب برای فراخوانی توابع بازگشتی چیه؟

فراخونی دنباله یه فراخوانی فرعی یا تابعیه که به عنوان آخرین عمل یه تابع فراخوانی انجام می‌شه. در حالی که **فراخوانی دنباله مناسب (PTC)** تکنیکیه که تو اون برنامه یا کد فریم‌های پشته‌ای (stack) اضافی برای بازگشت ایجاد نمی‌کنه، زمانی که فراخوانی تابع یه فراخوانی دنباله اس.

برای مثال، بازگشت کلاسیک یا سر تابع فاکتوریل پایین به پشته (stack) برای هر مرحله بستگی داره. هر مرحله باید تا  $n * \text{فاکتوریل}(n - 1)$  پردازش شه:

```
function factorial(n) {
  if (n === 0) {
    return 1
  }
  return n * factorial(n - 1)
}
console.log(factorial(5)); //120
```

اما اگه از Tail callback استفاده می‌کنیم اونا تمام داده‌های لازم رو که بهش نیاز داره رو بدون تکیه بر پشته (stack)، موقع برگشت به پایین منتقل می‌کنن.

```
function factorial(n, acc = 1) {
  if (n === 0) {
    return acc
  }
  return factorial(n - 1, n * acc)
}
console.log(factorial(5)); //120
```

الگوی بالا همون خروجی مورد اول رو برمی‌گردونه. اما اນباشت کننده کل رو به عنوان آرگومان بدون استفاده از حافظه پشته توی callBack دیابی می‌کنه.

## ۴۱۲. چطوری بررسی کنیم که یه آبجکت Promise هست یا نه؟

اگه نمیدونیم یه مقدار یه Promise هس یا نه، مقدار رو به صورت `Promise.resolve(value)` بپیچید که یه قول رو برمی‌گردونه.

```

function isPromise(object){
  if(Promise && Promise.resolve){
    return Promise.resolve(object) == object;
  }else{
    throw "Promise not supported in your environment"
  }
}

const i = 1;
const promise = new Promise(function(resolve,reject){
  resolve()
});

console.log(isPromise(i)); // false
console.log(isPromise(p)); // true

```

راه دیگر اینه که نوع `handler.then` رو بررسی کنیم

```

function isPromise(value) {
  return Boolean(value && typeof value.then === 'function');
}
const i = 1;
const promise = new Promise(function(resolve,reject){
  resolve()
});

console.log(isPromise(i)) // false
console.log(isPromise(promise)); // true

```

## چطوری متوجه بشیم که یا تابع با تابع `constructor` صدا زده شده یا نه؟

می‌توانیم از ویژگی شبیه `new.target` برای تشخیص اینکه آیا یه تابع به عنوان سازنده (با استفاده از عملگر جدید) فراخوانی شده یا به عنوان یه فراخوانی تابع معمولی استفاده کنیم.

۱. اگه سازنده یا تابعی با استفاده از عملگر جدید فراخوانی شه، `new.target` یه مرجع به سازنده یا تابع برمی‌گردونه.
۲. برای فراخوانی تابع، `new.target` تعریف نشده اس.

```

function Myfunc() {
  if (new.target) {
    console.log('called with new');
  } else {
    console.log('not called with new');
  }
}

new Myfunc(); // called with new
Myfunc(); // not called with new
Myfunc.call({}); //not called with new

```

## ۴.۱۴. تفاوت‌های بین آبجکت rest و پارامتر argument چیه؟

۱. آبجکت argument آرایه ماننده اما آرایه نیست. در حالی که توی Rest پارامترها آرایه هستن.

۲. آبجکت argument از روش‌هایی مانند sort، map، forEach pop یا پشتیبانی نمیکنه. در حالی که این روش‌ها رو می‌شه رو پارامترهای Rest استفاده کرد.  
۳. توی Rest پارامترها فقط اونایی هستن که نام جداگانه ای به اونا داده نشده در حالی که آبجکت argument شامل تمام آرگومان‌های ارسال شده به تابعه.

## ۴.۱۵. تفاوت‌های بین عملگر rest و پارامتر spread چیه؟

پارامتر Rest تمام عناصر باقی مانده رو تو یه آرایه جمع آوری میکنه. در حالی که عملگر Spread به تکرارپذیرها (آرایه‌ها / اشیاء / رشته‌ها) اجازه میده تا به آرگومان‌ها / عناصر منفرد گسترش پیدا کنن. یعنی پارامتر Rest مخالف عملگر spread هس.

## ۴.۱۶. نوع‌های مختلف generator کدوما هستن؟

۱. تعریف بیانی تابع : generator

```
function* myGenFunc() {
    yield 1;
    yield 2;
    yield 3;
}
const genObj = myGenFunc();
```

#### ۲. تعریف عبارتی تابع **generator**:

```
const myGenFunc = function* () {
    yield 1;
    yield 2;
    yield 3;
};
const genObj = myGenFunc();
```

#### ۳. تعریف بیانی تابع **generator** در آبجکت به صورت متد:

```
const myObj = {
    * myGeneratorMethod() {
        yield 1;
        yield 2;
        yield 3;
    }
};
const genObj = myObj.myGeneratorMethod();
```

#### ۴. تعریف تابع **generator** در کلاس‌ها:

```
class MyClass {
    * myGeneratorMethod() {
        yield 1;
        yield 2;
        yield 3;
    }
}
const myObject = new MyClass();
const genObj = myObject.myGeneratorMethod();
```

#### ۵. تعریف تابع **generator** به عنوان یک ویژگی محاسبه شده:

```

const SomeObj = {
  *[Symbol.iterator] () {
    yield 1;
    yield 2;
    yield 3;
  }
}

console.log(Array.from(SomeObj)); // [ 1, 2, 3 ]

```

## ۴۱۷. کدام built-in های iterable هستند؟

۱. آرایه‌ها و TypedArrays
۲. رشته‌ها: روی هر کاراکتر یا نقاط کد یونیکد تکرار کنیم
۳. Map: روی جفت‌های کلید-مقدار آن تکرار شه
۴. مجموعه‌ها: روی عناصر خود تکرار می‌شه
۵. آرگومان‌ها: یه متغیر خاص آرایه مانند در توابع NodeList DOM مانند
۶. مجموعه

## ۴۱۸. تفاوت‌های بین حلقه for...in و for...of چیه؟

- هم برای...in و هم برای...az دستورات روی ساختارهای داده js تکرار می‌شن. تنها تفاوت در مورد چیزیه که اونا تکرار می‌کنن:
۱. روی تمام کلیدهای خصوصیت شمارش پذیر یه شی تکرار می‌شه
  ۲. بیش از مقادیر یه شی قابل تکرار.
- بیاین این تفاوت رو توی یه مثال ببینیم،

```

let arr = ['a', 'b', 'c'];

arr.newProp = 'newValue';

// key are the property keys
for (let key in arr) {
  console.log(key);
}

// value are the property values
for (let value of arr) {
  console.log(value);
}

```

از اونجا که حلقه for..in روی کلیدهای شی تکرار می‌شه حلقه اول، ۰، ۱، ۲ و رو در حین تکرار روی شی آرایه ثبت می‌کنه. حلقه for..of روی مقادیر یه ساختار داده arr تکرار می‌شه و a، b، c رو تو کنسول ثبت می‌کنه.

## ۴۱۹. چطوری property و instance و غیری تعیین می‌کنی؟

خصوصیات Instance باید در داخل متدهای کلاس تعیین بشن. برای مثال، مشخصات نام و سن سازنده داخلی هم مثل مثال پایین تعیین می‌شون.

```

class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

```

اما خصوصیات داده prototype و (Static(class) ClassBody تعیین بشن. بیان مقدار سن رو برای کلاس Person به صورت زیر اختصاص بدیم.

```

Person.staticAge = 30;
Person.prototype.prototypeAge = 40;

```

## ۴۲۰. تفاوت‌های بین Number.isNaN و isNaN هستن؟

۱. تابع سراسری «`isNaN`» آرگومان رو به عدد تبدیل میکنه و اگه مقدار حاصل `NaN` باشه، `true` رو برمی‌گردونه.

۲. این روش آرگومان رو تبدیل نمیکنه. اما زمانی که نوع یه عدد و مقدار `NaN` باشه مقدار `true` رو برمی‌گردونه.

بیاین تفاوت رو با یه مثال ببینیم:

```
isNaN('hello'); // true  
Number.isNaN('hello'); // false
```