### آموزش وب دزاین

فهرست

[آموزش وب دزاین 1](#_Toc209605103)

[مفهوم Frontend (فرانت‌اند) 4](#_Toc209605104)

[تفاوت Frontend با Backend (یک مقایسه کلیدی) 6](#_Toc209605105)

[مهارت‌های مورد نیاز یک توسعه‌دهنده فرانت‌اند 7](#_Toc209605106)

[بررسی مفهوم Backend 7](#_Toc209605107)

[Backend (بک‌اند) چیست؟ 7](#_Toc209605108)

[اجزای اصلی توسعه بک‌اند (Core Components) 8](#_Toc209605109)

[مهارت‌های مورد نیاز یک توسعه‌دهنده بک‌اند 9](#_Toc209605110)

[معرفی HTML: 10](#_Toc209605111)

[HTML چیست؟ 10](#_Toc209605112)

[HTML چگونه کار می‌کند؟ 10](#_Toc209605113)

[مثال یک تگ ساده: 11](#_Toc209605114)

[توضیح بخش‌های مختلف: 11](#_Toc209605115)

[معرفی مهم‌ترین تگ‌های HTML 12](#_Toc209605116)

[ویژگی‌های تگ‌ها (Attributes) 14](#_Toc209605117)

[مثال‌های رایج: 14](#_Toc209605118)

[معرفی Css: 14](#_Toc209605119)

[CSS چیست؟ 15](#_Toc209605120)

[CSS چگونه کار می‌کند؟ 15](#_Toc209605121)

[روش‌های اتصال CSS به HTML 16](#_Toc209605122)

[2. Box Model (مدل جعبه‌ای) 17](#_Toc209605123)

[ویژگی‌های پرکاربرد CSS 19](#_Toc209605124)

[معرفی JavaScript 19](#_Toc209605125)

[ویژگی‌های کلیدی JavaScript: 20](#_Toc209605126)

[چندپارادایمی: پشتیبانی از برنامه‌نویسی شیءگرا، تابعی و رویه‌ای: توضیح چندپارادایمی بودن جاوااسکریپت 20](#_Toc209605127)

[1- برنامه نویسی تابعی یا رویه‌ایی: 20](#_Toc209605128)

[2- برنامه نویسی شیئ گرا (ObjectOriented Programming) 21](#_Toc209605129)

[3- برنامه‌نویسی تابعی (Functional Programming) 22](#_Toc209605130)

[آموزش استفاده از Map در جاوا اسکریپت 25](#_Toc209605131)

[1- مقدمه‌ای بر Map 25](#_Toc209605132)

[نکات پیشرفته 35](#_Toc209605133)

[تمرینات اضافه 38](#_Toc209605134)

[آموزش کار با createDocumentFragment() 44](#_Toc209605135)

[آموزش استفاده از filter 44](#_Toc209605136)

[آموزش جامع استفاده از متد filter() در جاوااسکریپت 45](#_Toc209605137)

[1- مقدمه‌ای بر متد filter() 45](#_Toc209605138)

[2- سینتکس و پارامترها 45](#_Toc209605139)

[3- موارد استفاده و مثال‌های کاربردی 46](#_Toc209605140)

[4- تفاوت filter با سایر متدهای آرایه 47](#_Toc209605141)

[5- بهترین practices و نکات مهم 48](#_Toc209605142)

[خلاصه مزایای استفاده از filter(): 52](#_Toc209605143)

[آموزش استفاده از reduce 52](#_Toc209605144)

[مثال‌های کاربردی 53](#_Toc209605145)

[تمرین‌ : 55](#_Toc209605146)

[آموزش استفاده از Rest parameters آموزش استفاده از Spread Operator: 57](#_Toc209605147)

[1. Rest Parameters (پارامترهای rest) 57](#_Toc209605148)

[2. Spread Operator (عملگر spread) 58](#_Toc209605149)

[آموزش کار با تابع join() 60](#_Toc209605150)

[آموزش Destructuring Assignment 61](#_Toc209605151)

[Destructuring آبجکت‌ها (Object Destructuring) 62](#_Toc209605152)

[کاربردهای عملی Destructuring 64](#_Toc209605153)

بررسی مفهمUI:

البته! در ادامه به بررسی مفهوم UI (رابط کاربری) به زبان فارسی میپردازیم.

UI (رابط کاربری) چیست؟

UI یا User Interface (رابط کاربری) به تمام بخش‌های بصری و گرافیکی یک محصول دیجیتال (مانند وب‌سایت، اپلیکیشن موبایل، نرم‌افزار) گفته می‌شود که کاربر با آن تعامل مستقیم دارد. هدف UI ایجاد یک رابط زیبا، intuitive (قابل درک و شهودی) و کارآمد است تا کاربر بتواند به راحتی با محصول ارتباط برقرار کند.

به زبان ساده، UI ظاهر و حس یک محصول است. هر چیزی که می‌بینید، کلیک می‌کنید، یا با آن تعامل دارید، بخشی از UI است.

اجزای اصلی UI (رابط کاربری)

رابط کاربری از عناصر مختلفی تشکیل شده است که با هم ترکیب می‌شوند تا یک تجربه بصری منسجم را ایجاد کنند. این اجزا شامل موارد زیر هستند:

|  |  |  |
| --- | --- | --- |
| دسته‌بندی | توضیح | مثال‌ها |
| ۱. عناصر بصری (Visual Elements) | اینها اجزایی هستند که ظاهر و حس کلی طراحی را شکل می‌دهند. | رنگ‌ها: پالت رنگی که برای طراحی انتخاب می‌شود.<br> تایپوگرافی: فونت‌ها، سایز متن‌ها و spacing.<br> آیکون‌ها: نمادهای کوچک برای انتقال مفاهیم.<br> عکس‌ها و تصاویر: محتوای بصری جذاب.<br> فضای خالی (White Space): فضای بین عناصر برای خوانایی بهتر. |
| ۲. عناصر تعاملی (Interactive Elements) | اینها المان‌هایی هستند که کاربر با آنها تعامل فیزیکی دارد. | دکمه‌ها (Buttons): برای انجام اقدامات (ثبت، ذخیره، ارسال).<br> فیلدهای فرم (Form Fields): برای وارد کردن اطلاعات (متون، ایمیل، رمز).<br> منوها (Menus): برای پیمایش در سایت یا اپلیکیشن.<br> اسلایدرها (Sliders) و کاروسل‌ها (Carousels): برای نمایش محتوا به صورت متغیر.<br> چک‌باکس‌ها (Checkboxes) و Radio Buttons: برای انتخاب گزینه‌ها. |
| ۳. چیدمان و ساختار (Layout & Structure) | این بخش به چگونگی سازماندهی و قرارگیری عناصر روی صفحه اشاره دارد. | | Grid System: سیستم شبکه‌ای برای مرتب کردن عناصر.<br> هدر (Header) و فوتر (Footer): بخش‌های ثابت بالا و پایین صفحه.<br> ناوبری (Navigation): منوها و مسیرهای حرکت کاربر در محصول.<br> کارد (Card) و لیست‌ها: روش‌های نمایش اطلاعات |

تفاوت UI و UX (یک تفاوت کلیدی)

این دو مفهوم بسیار به هم مرتبط هستند، اما یکی نیستند :

UI (رابط کاربری): به ظاهر و المان‌های گرافیکی محصول می‌پردازد. (زیبایی‌شناسی، رنگ‌ها، تایپوگرافی)

UX (تجربه کاربری): به حس و تجربه کلی کاربر از تعامل با محصول می‌پردازد. (آسانی استفاده، کارایی، رضایت)

یک مقایس ساده:

تصور کنید یک خودرو دارید:

UI شامل: فرمان، صفحه کیلومترشمار، دکمه‌های رادیو، رنگ و طراحی داخلی خودرو است.

UX شامل: این است که رانندگی با این خودرو چقدر راحت است، آیا فرمان به خوبی پاسخ می‌دهد، و آیا از رانندگی با آن لذت می‌برید یا نه.

یک UI زیبا ولی با UX ضعیف، مانند یک ماشین بسیار زیبا ولی با فرمان سخت و ناراحت است. برعکس، یک UX عالی با UI ضعیف، مانند یک ماشین بسیار راحت ولی با ظاهر زشت است. محصول موفق ترکیبی از هر دو است.

اصول طراحی UI خوب

یک طراح UI برای ایجاد یک رابط موثر، این اصول را رعایت می‌کند:

1. وضوح و سادگی (Clarity & Simplicity): طراحی باید ساده و بدون عناصر غیرضروری باشد تا کاربر به راحتی بتواند هدف را پیدا کند.

2. ثبات و یکنواختی (Consistency): استفاده از المان‌های یکسان در کل محصول (مثلاً دکمه‌های همشکل، رنگ‌های ثابت). این کار باعث می‌شود کاربر احساس آشنا بودن کند.

3. بازخورد (Feedback): رابط باید به کاربر بازخورد دهد. مثلاً وقتی دکمه‌ای کلیک می‌شود، تغییر رنگ دهد یا یک پیام "عملیات موفق" نمایش داده شود.

4. سلسله مراتب بصری (Visual Hierarchy): چیدمان عناصر به گونه‌ای باشد که مهم‌ترین اطلاعات یا اقدامات، توجه کاربر را جلب کنند.

5. دسترسی‌پذیری (Accessibility): طراحی باید به گونه‌ای باشد که افراد با disabilities (معلولیت‌ها) مانند کوررنگی یا مشکلات بینایی نیز بتوانند از آن استفاده کنند (مثلاً کنتراست رنگی کافی داشته باشد).

نتیجه‌گیری

UI یا رابط کاربری، جلوه بصری و لمسی یک محصول دیجیتال است که اولین impression (تاثیر) را روی کاربر می‌گذارد. یک UI خوب، نه تنها زیبا است، بلکه کاربردی، intuitive و لذت‌بخش است و به کاربر کمک می‌کند تا به هدف خود به راحتی برسد.

طراحی UI یک تخصص فنی خلاقانه است که در کنار طراحی UX، پایه و اساس ساخت محصولات دیجیتال موفق را تشکیل می‌دهد.

# مفهوم Frontend (فرانت‌اند)

Frontend (فرانت‌اند) چیست؟

Frontend یا سمت کلاینت (Client Side) به تمام بخش‌هایی از یک وب‌سایت یا اپلیکیشن گفته می‌شود که مستقیماً توسط کاربر دیده و با آن تعامل می‌شود . هر چیزی که هنگام باز کردن یک صفحه وب در مرورگر خود مشاهده می‌کنید—از طرحبندی و رنگ‌ها گرفته تا منوها، دکمه‌ها، تصاویر و انیمیشن‌ها—نتیجه کدهایی است که توسط توسعه‌دهنده فرانت‌اند نوشته شده است.

به بیان ساده، فرانت‌اند ظاهر، حس و تعامل یک وب‌سایت است.

اجزای اصلی توسعه فرانت‌اند (Three Pillars)

توسعه فرانت‌اند بر سه technology (فناوری) اصلی استوار است:

|  |  |  |
| --- | --- | --- |
| فناوری | نقش و مسئولیت | توضیح |
| HTML | اسکلت و ساختار | (HyperText Markup Language)  مسئول ایجاد ساختار و محتوای صفحه است. تمامی متن‌ها، تصاویر، لینک‌ها و عناصر اصلی صفحه توسط HTML تعریف می‌شوند. |
| CSS | ظاهر و استایل | (Cascading Style Sheets) مسئول زیباسازی و چیدمان عناصر HTML است. رنگ‌ها، فونت‌ها، انیمیشن‌ها، layout (چیدمان) responsive (واکنش‌گرا) و هر آنچه مربوط به ظاهر سایت است با CSS پیاده‌سازی می‌شود. |
| JavaScript | رفتار و تعامل | مسئول هوشمندسازی و تعاملی کردن صفحه است. هرگونه (رفتار پویا)—مانند کلیک بر روی دکمه‌ها، ارسال فرم‌ها، بارگذاری داده بدون رفرش صفحه (AJAX) و ایجاد انیمیشن‌های پیچیده—توسط JavaScript کنترل می‌شود. |

## تفاوت Frontend با Backend (یک مقایسه کلیدی)

برای درک کامل فرانت‌اند، مقایسه آن با backend (بک‌اند) ضروری است:

|  |  |  |
| --- | --- | --- |
| معیار | Frontend (فرانت‌اند) | Backend (بک‌اند) |
| محل اجرا | مرورگر کاربر (Client Side) | سرور (Server Side) |
| کاربری که می‌بیند | کاربر نهایی | توسعه‌دهندگان و صاحبان سایت |
| مسئولیت | چگونگی نمایش و تعامل با داده‌ها | پردازش، ذخیره‌سازی و مدیریت داده‌ها |
| مثال | ظاهر یک فرم ثبت‌نام | بررسی اطلاعات ورودی، ذخیره آن در دیتابیس و ارسال ایمیل تایید |
| تکنولوژی‌های رایج | HTML, CSS, JS, React, Angular, Vue | Python, PHP, Java, Node.js, Ruby دیتابیس‌هایی مثل MySQL |

یک مقایسه ساده:   
یک رستوران را در نظر بگیرید:

فرانت‌اند: فضای داخلی زیبای رستوران، میزها، صندلی‌ها، منوی غذایی که مشتری آن را می‌بیند و با آن تعامل دارد.

بک‌اند: آشپزخانه، محل ذخیره مواد غذایی، سیستم مدیریت سفارشات و آشپزها که در پشت صحنه کار می‌کنند.

هر دوی این بخش‌ها برای ارائه یک تجربه عالی به مشتری (کاربر) ضروری هستند.

## مهارت‌های مورد نیاز یک توسعه‌دهنده فرانت‌اند

یک توسعه‌دهنده فرانت‌اند مدرن، علاوه بر تسلط بر سه پایه اصلی، به مهارت‌های دیگری نیز نیاز دارد:

1. فریم‌ورک‌ها و کتابخانه‌های JavaScript: تسلط بر حداقل یکی از فریم‌ورک‌های محبوب مانند React ، Angular یا Vue.js امروزه ضروری هستند. این ابزارها توسعه اپلیکیشن‌های پیچیده و تعاملی را بسیار سریع‌تر و آسان‌تر می‌کنند.

2. Responsive Design (طراحی واکنش‌گرا): توانایی طراحی سایت‌هایی که ظاهر و کارایی خود را روی تمام دستگاه‌ها (موبایل، تبلت، دسکتاپ) به optimal (بهینه)ترین شکل حفظ می‌کنند. این کار با استفاده از CSS Media Queries و layoutهای انعطافپذیر مانند Flexbox و CSS Grid انجام می‌شود.

3. Version Control (کنترل نسخه): تسلط بر سیستم‌هایی مانند Git (و پلتفرم‌هایی مثل GitHub یا GitLab) برای مدیریت کدها و همکاری با دیگر توسعه‌دهندگان.

4. ابزارهای Build و Module Bundlers: آشنایی با ابزارهایی مانند Webpack ، Vite ، Babel که کدهای نوشته شده را برای production (استفاده نهایی) بهینه و bundle می‌کنند.

5. APIها: توانایی برقراری ارتباط و دریافت داده از backend melalui REST API یا GraphQL . (یعنی فراخوانی API و کار با داده JSON).

6. مفاهیم پایه سئو (SEO): آگاهی از اصول اولیه سئو برای تولید کدهایی که توسط موتورهای جستجو مانند Google به راحتی crawl و index شوند.

# بررسی مفهوم Backend

## Backend (بک‌اند) چیست؟

Backend یا سمت سرور (Server Side) به تمام بخش‌های پنهان یک وب‌سایت یا اپلیکیشن گفته می‌شود که کاربر به طور مستقیم آن را نمی‌بیند، اما برای عملکرد صحیح سیستم absolutely vital (کاملاً حیاتی) است . بک‌اند مسئول پردازش منطق کسب‌وکار، ذخیره‌سازی داده‌ها، مدیریت کاربران و در نهایت، پاسخگویی به درخواست‌هایی است که از سمت فرانت‌اند ارسال می‌شود.

به بیان ساده، اگر فرانت‌اند را مغز زیبا و کاربرپسند یک فروشگاه بدانیم، بک‌اند انبار، سیستم حسابداری، مدیریت موجودی و logistics (لجستیک) پشت صحنه آن است.

## اجزای اصلی توسعه بک‌اند (Core Components)

یک سیستم بک‌اند معمولاً از چند بخش کلیدی تشکیل شده است:

|  |  |
| --- | --- |
| جزء | توضیح و مسئولیت |
| 1. سرور (Server) | یک کامپیوتر قدرتمند است که نرم‌افزار سرویس‌دهنده (مانند Nginx, Apache) روی آن نصب شده و مسئول دریافت و پاسخگویی به درخواست‌های کلاینت‌ها (مرورگرهای کاربران) است. |
| 2. برنامه (Application) | این همان کد اصلی ی است که توسعه‌دهنده بک‌اند می‌نویسد. این کد با استفاده از فریم‌ورک‌ها و زبان‌های برنامه‌نویسی (مانند Python/Django, PHP/Laravel, Node.js/Express) نوشته می‌شود و مسئول اجرای منطق کسب‌وکار (مثلاً محاسبه قیمت سبد خرید، بررسی اعتبار کاربر) است. |
| 3. پایگاه داده (Database) | | انبار داده برنامه است. تمام اطلاعات مهم مانند اطلاعات کاربران، محصولات، مقالات و تراکنش‌ها در اینجا ذخیره، مدیریت و بازیابی می‌شوند. مثال‌ها: MySQL, PostgreSQL, MongoDB. |
| Application Programming Interface API | پل ارتباطی بین فرانت‌اند و بک‌اند است. API一 قوانین و protocols (پروتکل‌هایی) است که مشخص می‌کند فرانت‌اند چگونه می‌تواند از بک‌اند درخواست داده کند یا داده ارسال کند (معمولاً از طریق REST یا GraphQL ). |

یک مقایسه ساده:

یک نمایش عروسکی را در نظر بگیرید:

فرانت‌اند: عروسک‌های زیبا و صحنه‌ای که تماشاگران می‌بینند.

بک‌اند: عروسک‌گردان‌ها، نویسنده داستان، نورپرداز و تمام عوامل پشت صحنه که نمایش را ممکن می‌کنند.

## مهارت‌های مورد نیاز یک توسعه‌دهنده بک‌اند

یک توسعه‌دهنده بک‌اند موفق به مهارت‌های فنی و مفهومی خاصی نیاز دارد:

1. تسلط بر یک زبان برنامه‌نویسی سمت سرور: مانند Python (با فریم‌ورک Django یا Flask)، JavaScript (با فریم‌ورک Node.js/Express)، PHP (با فریم‌ورک Laravel یا Symfony)، Java (با Spring) یا C (با .NET).

2. دانش پایگاه‌های داده و SQL: توانایی Design (طراحی) ، ایجاد و نوشتن Query های بهینه برای پایگاه‌های داده رابطه‌ای (مانند MySQL) و/یا غیررابطه‌ای (NoSQL مانند MongoDB).

3. کار با APIها: توانایی Design و توسعه API های RESTful یا GraphQL که قابل اعتماد، امن و well documented (دارای مستندات خوب) باشند.

4. امنیت (Security): آگاهی از حفره‌های امنیتی رایج (مثل SQL Injection, XSS, CSRF) و روش‌های مقابله با آنها. مدیریت احراز هویت (Authentication) و مجوزدهی (Authorization) کاربران.

5. مدیریت سرور و Deployment (استقرار): آشنایی با مبانی DevOps مانند کار با سرورهای لینوکس، استفاده از Docker برای containerization، و platforms (پلتفرم‌هایی) مانند AWS , Azure یا Google Cloud برای میزبانی برنامه.

6. حل مسئله و تفکر الگوریتمی: از آنجایی که بک‌اند با منطق و داده‌های حجیم سروکار دارد، توانایی حل مسئله و نوشتن الگوریتم‌های بهینه بسیار crucial (حیاتی) است.

# معرفی HTML:

## HTML چیست؟

HTML مخفف HyperText Markup Language (زبان نشانه‌گذاری ابرمتنی) است. HTML یک زبان برنامه‌نویسی نیست، بلکه یک زبان نشانه‌گذاری (Markup Language) است که برای ساختاردهی و سازماندهی محتوای صفحات وب استفاده می‌شود.

به بیان ساده، HTML اسکلت و ساختار اصلی یک صفحه وب را می‌سازد. تمام متون، تصاویر، لینک‌ها، جدول‌ها و عناصری که در مرورگر خود می‌بینید، ابتدا توسط HTML تعریف و ساخته می‌شوند.

## HTML چگونه کار می‌کند؟

HTML از المان‌هایی به نام تگ (Tag) تشکیل شده است. هر تگ محتوای داخل خود را تعریف می‌کند. مرورگر این تگ‌ها را می‌خواند و سپس بر اساس آنها، محتوای صفحه را Render (نمایش) می‌دهد.

## مثال یک تگ ساده:

|  |
| --- |
| <p>      این یک پاراگراف است.  </p> |

`<p>`: تگ باز کردن (شروع پاراگراف)

`این یک پاراگراف است.`: محتوای قابل مشاهده

`</p>`: تگ بستن (پایان پاراگراف)

ساختار اصلی یک سند HTML

هر صفحه HTML دارای یک ساختار پایه و استاندارد است:

|  |
| --- |
| <!DOCTYPE html>  <html lang="fa" dir="rtl">  <head>      <meta charset="UTF8">      <meta name="viewport" content="width=devicewidth, initialscale=1.0">      <title>عنوان صفحه من</title>  </head>  <body>      <! تمام محتوای قابل مشاهده صفحه اینجا قرار می‌گیرد >      <h1>این یک عنوان مهم است</h1>      <p>این یک پاراگراف نمونه است.</p>  </body>  </html> |

## توضیح بخش‌های مختلف:

1. <!DOCTYPE html>`: به مرورگر می‌گوید که این سند از نوع HTML5 است.

2. <html>`: تگ ریشه (Root) صفحه است. ویژگی `lang="fa"` زبان صفحه را فارسی و `dir="rtl"` جهت نوشتار را راست‌چین (RighttoLeft) تعیین می‌کند.

3. <head>`: حاوی MetaInformation (اطلاعات فرامیانی) درباره صفحه است که به کاربر نمایش داده نمی‌شود (مانند عنوان صفحه در تب مرورگر، لینک به فایل‌های CSS و JavaScript، کلمات کلیدی برای موتورهای جستجو).

4. <body>`: حاوی تمام محتوای قابل مشاهده صفحه است. هر چیزی که کاربر در مرورگر می‌بیند، داخل این تگ قرار می‌گیرد.

5. < comment !>`: این یک توضیح (Comment) است. مرورگر آن را نادیده می‌گیرد و فقط برای توسعه‌دهندگان قابل مشاهده است.

## معرفی مهم‌ترین تگ‌های HTML

تگ‌های HTML برای اهداف مختلفی استفاده می‌شوند. در جدول زیر برخی از پرکاربردترین آن‌ها را مشاهده می‌کنید:

|  |  |  |  |
| --- | --- | --- | --- |
| دسته‌بندی | تگ | توضیح | مثال |
| ساختار صفحه | >header>`, `<footer>`, `<nav>`, `<main>`, `<section>`, `<article>`, `<div> | برای سازماندهی و تقسیم‌بندی بخش‌های مختلف صفحه به کار می‌روند | >header>  …  </header> |
| سرتیترها | (Headings) `<h1>`,  `<h2>`, `<h3>`,  `<h4>`, `<h5>`, `<h6> | برای ایجاد عنوان استفاده می‌شوند. `<h1>` مهم‌ترین و بزرگ‌ترین و `<h6>` کم‌اهمیت‌ترین و کوچک‌ترین است. | >h1> عنوان اصلی </h1> |
| متن‌ها | >p< | برای ایجاد پاراگراف | `>p< متن پاراگراف.  >p< |
| لینک‌ها | >a< | برای ایجاد هایپرلینک به صفحات دیگر. | >a href="<https://google.com><"  گوگل  />a< |
| تصاویر | >img< | برای قرار دادن تصویر. این تگ selfclosing است (تگ بسته ندارد). | >img src="image.jpg" alt<"توضیح تصویر"= |
| لیست‌ها | >ul>`, `<ol>`,  `>li< | برای ایجاد لیست‌های ترتیبی (Ordered) و غیرترتیبی | Unordered: ` <ul> <li<  آیتم  >/li< >li>  آیتم ۱  </li>  </ul< |
| جدول‌ها | >table>`, `<tr>`, `<td>`,  `>th< | برای ایجاد و نمایش داده‌ها در قالب جدول. | >table> <tr> <td> داده </td> </tr> </table> |
| فرم‌ها | >form>`, `<input>`, `<button>`, `<textarea>`, `<select> | برای دریافت اطلاعات از کاربر (مانند فرم تماس یا فرم لاگین). | >form> <input type="text"> <button>  ارسال </button> </form> |

## ویژگی‌های تگ‌ها (Attributes)

هر تگ می‌تواند ویژگی‌هایی (Attributes) داشته باشد که اطلاعات بیشتری درباره آن تگ ارائه می‌دهند. ویژگی‌ها همیشه در تگ بازکننده و به صورت `name="value"` نوشته می‌شوند.

## مثال‌های رایج:

href` در تگ `<a>`: آدرس مقصد لینک را مشخص می‌کند.

|  |
| --- |
| <a href="https://www.example.com">  این یک لینک است  </a> |

src` و alt` در تگ <img src=’’ alt=’’> مسیر تصویر و `alt` متن جایگزین برای تصویر (در صورت عدم نمایش) را مشخص می‌کند.

|  |
| --- |
| <img src="cat.jpg" alt="یک گربه ناز"> |

class` و id`: برای style دادن به عناصر با CSS یا دسترسی به آنها با JavaScript استفاده می‌شوند.

|  |
| --- |
| <p class="importanttext" id="mainparagraph">  این متن مهم است.  </p> |

# معرفی Css:

## CSS چیست؟

CSS مخفف Cascading Style Sheets (شیوه‌نامه آبشاری) است. CSS یک زبان استایل‌دهی است که ظاهر، چیدمان و نمایش محتوای HTML را توصیف می‌کند.

به بیان ساده، اگر HTML را اسکلت و ساختار یک ساختمان در نظر بگیریم، CSS نما، رنگ دیوارها، دکوراسیون و چیدمان داخلی آن است. CSS است که یک صفحه متنی ساده را به یک وب‌سایت زیبا و جذاب تبدیل می‌کند.

## CSS چگونه کار می‌کند؟

CSS با انتخاب المان‌های HTML و اعمال استایل‌های مختلف (مانند رنگ، فونت، اندازه، موقعیت) روی آنها کار می‌کند. این کار با استفاده از قوانینی به نام Rule Set انجام می‌شود.

ساختار یک قانون CSS:

|  |
| --- |
| selector {      property: value;      property: value;  } |

Selector (انتخاب‌گر): مشخص می‌کند که کدام المان(های) HTML باید استایل بگیرند.

Property (ویژگی): مشخص می‌کند کدام aspect (جنبه) از المان را می‌خواهید تغییر دهید (مثلاً `color` برای رنگ متن).

Value (مقدار): مشخص می‌کند چگونه می‌خواهید آن ویژگی را تغییر دهید (مثلاً `red` برای رنگ قرمز).

Declaration (اعلامیه): هر جفت `property: value` یک اعلامیه است.

مثال:

|  |
| --- |
| p {  color: blue;  fontsize: 16px;  } |

این کد به مرورگر می‌گوید: "تمام تگ‌های `<>p>` (پاراگراف) را با رنگ آبی (`blue`) و اندازه فونت 16 پیکسل (`16px`) نمایش بده."

## روش‌های اتصال CSS به HTML

سه روش اصلی برای اضافه کردن استایل‌های CSS به یک صفحه وب وجود دارد:

|  |  |  |
| --- | --- | --- |
| روش | توضیح | مثال |
| | 1. External CSS (خارجی) | بهترین و توصیه‌شده‌ترین روش. قوانین CSS در یک فایل جداگانه با پسوند `.css` ذخیره می‌شوند و سپس به HTML لینک می‌شوند. | در فایل  style.css:  p { color: red; }  درفایل  HTML:  <link rel="stylesheet" href="style.css"> |
| 2. Internal CSS (داخلی) | قوانین CSS داخل تگ `<>style>` در بخش `<>head>` سند HTML نوشته می‌شوند. | >head> <style>  p { color: red; } </style>  </head> |
| 3. Inline CSS (درون‌خطی) | استایل مستقیماً به یک المان HTML خاص با استفاده از attribute (ویژگی) `style` اضافه می‌شود. این روش کمتر توصیه می‌شود. | >p style="color: red">  این متن قرمز است.  </p> |
|  |  |  |

مهم‌ترین مفاهیم و ویژگی‌های CSS

1. Selectors (انتخاب‌گرها)

انتخاب‌گرها مشخص می‌کنند که استایل روی کدام المان‌ها اعمال شود.

|  |  |  |
| --- | --- | --- |
| نوع انتخاب‌گر | مثال | توضیح |
| Element Selector | P | تمام تگ‌های `<p>` را انتخاب می‌کند. |
| Class Selector | .myclass | تمام المان‌هایی که class="myclass"  دارند را انتخاب می‌کند (متدوال‌ترین روش). |
| ID Selector | myid | المانی که  id="myid"  دارد را انتخاب می‌کند (منحصر به فرد) |
| Descendant Selector | div p | تمام تگ‌های  <p>  که درون یک  <div>    قرار دارند را انتخاب می‌کند. |

## 2. Box Model (مدل جعبه‌ای)

این یکی از مهم‌ترین مفاهیم CSS است. هر المان HTML در CSS به عنوان یک جعبه (Box) در نظر گرفته می‌شود که از چهار بخش تشکیل شده است:

Content (محتوای اصلی)

Padding (فاصله داخلی) بین محتوا و border

Border (حاشیه)

Margin (فاصله خارجی) بین border و المان‌های دیگر

|  |
| --- |
| div {      width: 300px; / عرض محتوا /      padding: 20px; / فاصله داخلی /      border: 5px solid black; / حاشیه /      margin: 10px; / فاصله خارجی /  } |

3. Layout (چیدمان)

کنترل موقعیت و چیدمان المان‌ها در صفحه.

Flexbox: یک مدل layout یک‌بعدی برای طراحی چیدمان‌های flexible (انعطاف‌پذیر) و responsive (واکنش‌گرا).

|  |
| --- |
| .container {  display: flex;  justifycontent: center;  alignitems: center;  } |

CSS Grid: یک مدل layout دو‌بعدی برای ایجاد چیدمان‌های شبکه‌ای پیچیده.

|  |
| --- |
| .container {  display: grid;  gridtemplatecolumns: 1fr 1fr 1fr;  gap: 10px;  } |

4. Responsive Design (طراحی واکنش‌گرا)

وب‌سایت در تمام دستگاه‌ها (موبایل، تبلت، دسکتاپ) به خوبی نمایش داده شود. این کار principalmente با Media Queries انجام می‌شود.

|  |
| --- |
| / استایل برای صفحه‌های با عرض کمتر از 600px (موبایل) /  @media (maxwidth: 600px) {  body {  backgroundcolor: lightblue;  }  .menu {  flexdirection: column;  }  } |

## ویژگی‌های پرکاربرد CSS

|  |  |  |  |
| --- | --- | --- | --- |
| دسته‌بندی | ویژگی | توضیح | مثال |
| متن و فونت | color | رنگ متن | color: ff0000; |
|  | fontfamily | نوع فونت | fontfamily: Arial; |
|  | fontsize | اندازه فونت | fontsize: 18px; |
|  | textalign | تراز متن | textalign: center; |
| پس‌زمینه | backgroundcolor | رنگ پس‌زمینه | backgroundcolor: blue; |
|  | backgroundimage | تصویر پس‌زمینه | backgroundimage: url('image.jpg'); |
| ابعاد و فاصله | width / height | عرض / ارتفاع | width: 100px; |
|  | padding | فاصله داخلی | padding: 10px; |
|  | margin | فاصله خارجی | margin: 20px; |
| حاشیه | border | حاشیه | border: 1px solid black; |
| موقعیت‌یابی | display | نوع نمایش | display: none; display: block; |
|  | position | نوع موقعیت‌یابی | position: relative; |

# معرفی JavaScript

JavaScript چیست؟JavaScript یک زبان برنامه‌نویسی سطح بالا، پویا و مفسری است که عمدتاً برای توسعه وب استفاده می‌شود. این زبان به شما امکان می‌دهد صفحات وب تعاملی ایجاد کنید.

## ویژگی‌های کلیدی JavaScript:

سطح بالا: نحو خوانا و نزدیک به زبان انسان

پویا: نوع‌دهی پویا (Dynamic Typing)

مفسری: بدون نیاز به کامپایل قبل از اجرا

# بررسی مفهوم SEO

بررسی کامل مفهوم SEO (بهینه‌سازی موتورهای جستجو)

## تعریف SEO

SEO (Search Engine Optimization) به مجموعه‌ای از تکنیک‌ها و استراتژی‌ها گفته می‌شود که برای بهبود رتبه‌بندی یک وب‌سایت در نتایج جستجوی طبیعی (غیرپولی) موتورهای جستجو مانند گوگل استفاده می‌شود.

اهداف اصلی SEO

- افزایش visibility (قابلیت دیده شدن)

- جذب ترافیک ارگانیک (رایگان)

- افزایش نرخ تبدیل (Conversion Rate)

- بناء برند (Brand Building)

## سه رکن اصلی SEO

1. سئوی فنی (Technical SEO)

• بهینه‌سازی سرعت سایت

• ساختار URL بهینه

• طراحی ریسپانسیو (موبایل فرندلی)

• استفاده از SSL/HTTPS

• نقشه سایت (Sitemap)

• معماری اطلاعات مناسب

2. سئوی محتوا (On-Page SEO)

• تحقیق کلمات کلیدی (Keyword Research)

• تولید محتوای با کیفیت و ارزشمند

• بهینه‌سازی تگ‌های عنوان (Title Tags)

• متا دیسکریپشن (Meta Descriptions)

• استفاده از هدینگ‌های مناسب (H1, H2, H3)

• بهینه‌سازی تصاویر (Alt Text)

3. سئوی خارجی (Off-Page SEO)

• لینک‌سازی خارجی (Backlinks)

• فعالیت در شبکه‌های اجتماعی

• برندسازی (Branding)

• نظرات کاربران (Reviews)

• مشارکت در فروم‌ها و انجمن‌ها

الگوریتم‌های مهم گوگل

|  |  |
| --- | --- |
| الگوریتم | هدف |
| Panda | مبارزه با محتوای کم‌کیفیت |
| Penguin | مبارزه با لینک‌های اسپم |
| Hummingbird | درک بهتر intent کاربر |
| BERT | درک زبان طبیعی |
| Core Updates | به روزرسانی‌های کلی |

## مراحل اجرای پروژه SEO

مرحله ۱: تحلیل و بررسی

- آنالیز رقبا

- Audit فنی سایت

- تحلیل کلمات کلیدی

مرحله ۲: بهینه‌سازی

- رفع مشکلات فنی

- تولید و بهینه‌سازی محتوا

- لینک‌سازی داخلی

مرحله ۳: لینک‌سازی

- ساخت بک‌لینک‌های باکیفیت

- فعالیت در شبکه‌های اجتماعی

مرحله ۴: مانیتورینگ

- رصد رتبه‌ها

- تحلیل ترافیک

- گزارش‌گیری منظم

سئوی کلاه سیاه (منفی)

- کلیک‌فروشی (Click Fraud)

- محتوای کپی (Duplicate Content)

- لینک‌های اسپم (Spam Links)

- مخفی‌سازی محتوا (Cloaking)

روندهای جدید SEO در ۲۰۲۴

- تجربه کاربری (UX) اهمیت بیشتر

- جستجوی صوتی (Voice Search)

- AI و محتوای هوشمند

- E-A-T (تخصص، اعتماد، Authority)

- Core Web Vitals (معیارهای سرعت)

ابزارهای کاربردی SEO

- Google Search Console

- Google Analytics

- Ahrefs/SEMrush

- Screaming Frog

- GTmetrix

نکات کلیدی موفقیت در SEO

1. صبر و تحمل (فرآیند زمان‌بر است)

2. تمرکز بر کاربر (User-First)

3. به روز بودن (الگوریتم‌ها دائماً تغییر می‌کنند)

4. تحلیل داده‌محور

5. یکپارچگی با بازاریابی دیجیتال

SEO یک فرآیند مستمر است که نیاز به استراتژی بلندمدت، تخصص فنی و درک عمیق از رفتار کاربران دارد.

# بررسی مفهوم Progressive Enhancement بهبود تدریجی

بررسی مفهوم Progressive Enhancement بهبود تدریجی

تصور کنید می‌خواهید یک ساندویچ درست کنید:

Progressive Enhancement یعنی:

1. اول نان بگذارید (پایه اصلی)

2. بعد کره بمالید (اضافه کردن قابلیت اول)

3. سپس پنیر بگذارید (قابلیت دوم)

4. در آخر گوجه اضافه کنید (قابلیت پیشرفته)

نتیجه: حتی اگر گوجه نداشته باشید، باز هم یک ساندویچ دارید!

فلسفه اصلی به زبان ساده

"اول مطمئن شویم همه می‌توانند از پایه‌ای ترین حالت استفاده کنند، سپس برای کسانی که امکانات بهتر دارند، تجربه بهتری بسازیم."

مثال عملی: ساخت یک وبسایت خبری

مرحله ۱: پایه اصلی (همه می‌بینند)

کاربران با: موبایل قدیمی، اینترنت کم سرعت، مرورگر ساده

مرحله ۲: اضافه کردن استایل (زیبایی)

/ مثل رنگ کردن خانه /

کاربران با: مرورگرهای نسبتاً جدید

مرحله ۳: قابلیت‌های پیشرفته

// مثل نصب آسانسور در ساختمان

// نمایش نظرات بدون رفرش صفحه

// پیشنهاد اخبار مشابه

کاربران با: مرورگرهای جدید، اینترنت پرسرعت

مقایسه با دو روش دیگر

روش ۱: Progressive Enhancement (ما)

- اول متن ساده ⭠ سپس رنگ ⭠بعد انیمیشن

- مزیت: همه می‌توانند بخوانند

روش ۲: Graceful Degradation (برعکس ما)

- اول انیمیشن پیچیده اگر نشد،⭠ متن ساده نشان بده

- مشکل: ممکن است بعضی کاربران اصلاً محتوا نبینند

مثال‌های واقعی از زندگی

مثال ۱: ارسال پیام

- پایه: SMS ساده

- میانی: پیام با emoji

- پیشرفته: ویدیو، موقعیت مکانی، reactions

مثال ۲: تماشای فیلم

- پایه: دانلود با کیفیت پایین

- میانی: استریمینگ با کیفیت معمولی

- پیشرفته: 4K, HDR, واقعیت مجازی

نمودار درک مطلب

|  |
| --- |
| کاربران با امکانات محدود  ↓  محتوا و عملکرد اصلی ← همه می‌توانند استفاده کنند  ↓  کاربران با امکانات متوسط  ↓  ظاهر زیباتر + قابلیت‌های بیشتر  ↓  کاربران با امکانات پیشرفته  ↓  تجربه فوق‌العاده و تعاملی |

چرا این روش مهم است؟

۱. عدالت دیجیتال

- کاربران روستایی با اینترنت کند

- افراد با دستگاه‌های قدیمی

- افراد با معلولیت (نابینایان با screen reader)

۲. آمادگی برای شرایط مختلف

- وقتی اینترنت قطع می‌شود

- وقتی JavaScript load نمی‌شود

- وقتی مرورگر قدیمی است

۳. ساخت پایه محکم

- مثل ساختن ساختمان: اول اسکلت محکم، سپس نماکاری

تمرین فکری برای دانشجویان

سناریو: ساخت یک فرم ثبت‌نام

سوال: چگونه این فرم را با Progressive Enhancement بسازیم؟

پاسخ مرحله به مرحله:

1. پایه: فرم HTML ساده که بدون JavaScript کار می‌کند

2. مرحله ۲: اضافه کردن CSS برای زیبایی

3. مرحله ۳: JavaScript برای validation لحظه‌ای

4. مرحله ۴: autocomplete پیشرفته

## نکات کلیدی برای یادآوری

1. اول مفید، سپس زیبا، بعد پیشرفته

2. همیشه به کاربران با امکانات محدود فکر کنید

3. پایه محکم = تجربه قابل اعتماد برای همه

4. این روش شبیه زندگی واقعی است

## جمع‌بندی برای مبتدیان

## Progressive Enhancement یعنی:

"ما یک محصول دیجیتال را طوری می‌سازیم که اول برای همه قابل استفاده باشد، سپس برای کسانی که امکانات بهتری دارند، تجربه بهتری ایجاد کنیم."

این روش نه تنها فنی، بلکه انسانی و منصفانه است!

# بررسی مفهوم Responsive Web Design

بررسی مفهوم Responsive Web Design برای مبتدیان مطلق

توضیح با یک قیاس ساده: مبلمان اتاق

تصور کنید می‌خواهید اتاقی را با مبلمان چیدمان کنید:

اتاق کوچک (موبایل):

- یک مبل کوچک

- میز عسلی کوچک

- چیدمان ساده و خطی

اتاق متوسط (تبلت):

- یک مبل 3 نفره

- دو مبل تک نفره

- میز بزرگتر

اتاق بزرگ (دسکتاپ):

- چندین مبل بزرگ

- میز ناهارخوری

- گلدان و دکوراسیون اضافی

وب‌سایت ریسپانسیو هم دقیقاً همین کار را می‌کند!

مفهوم اصلی به زبان کودکانه

"یک وب‌سایت ریسپانسیو مثل شنل جادویی است که وقتی روی صفحه‌های مختلف نمایش داده می‌شود، شکل و اندازه‌اش عوض می‌کند!"

مشکل قدیمی را ببینید

وب‌سایت‌های قدیمی (غیر-ریسپانسیو):

<!-- مثل یک میز ثابت که نمی‌شود کوچکش کرد -->

موبایل: [=== باید اسکرول کنی ===]

دسکتاپ: [محتوای کامل دیده می‌شود]

وب‌سایت‌های جدید (ریسپانسیو):

<!-- مثل آب که شکل ظرف را می‌گیرد -->

موبایل: [محتوا]

[به صورت]

[عمودی]

دسکتاپ: [محتوا به صورت افقی کنار هم]

سه ابرقدرت Responsive Design

1. قدرت انعطاف‌پذیری (Flexible Layout)

- مثل کش لاستیکی که می‌تواند کوچک و بزرگ شود

- عناصر صفحه می‌توانند نسبت به صفحه بزرگ و کوچک شوند

2. قدرت شرطی (Media Queries)

- مثل دماسنج هوشمند که:

- اگر هوا سرد شد، بخاری روشن کند

- اگر هوا گرم شد، کولر روشن کند

- اگر صفحه کوچک بود، چیدمان موبایل نشان دهد

- اگر صفحه بزرگ بود، چیدمان دسکتاپ نشان دهد

3. قدرت تصاویر جادویی (Flexible Images)

- تصاویر مثل آب در لیوان می‌شوند

- خودکار بزرگ و کوچک می‌شوند

یک داستان تصویری

داستان: سفر یک وب‌سایت به دستگاه‌های مختلف

صحنه ۱: ورود به موبایل کوچک

- منو تبدیل به دکمه ☰ می‌شود

- عکس‌ها کوچک و شفاف می‌شوند

- متن‌ها در ستون باریک نمایش داده می‌شوند

صحنه ۲: ورود به تبلت

- منو به صورت افقی نمایش داده می‌شود

- دو ستون اطلاعات کنار هم قرار می‌گیرند

- عکس‌ها متوسط می‌شوند

صحنه ۳: ورود به دسکتاپ

- منو کامل با زیرمنوها نمایش داده می‌شود

- چندین ستون کنار هم قرار می‌گیرند

- عکس‌ها بزرگ و با کیفیت نمایش داده می‌شوند

نمودار درک مطلب برای دانشجویان

|  |
| --- |
| صفحه نمایش کوچک (موبایل)  ↓  محتوای اصلی + چیدمان عمودی  ↓  صفحه نمایش متوسط (تبلت)  ↓  چیدمان ۲ ستونی + منوی ساده  ↓  صفحه نمایش بزرگ (دسکتاپ)  ↓  چیدمان چندستونی + منوی کامل |

تمرین فکری: طراحی یک روزنامه

سوال: چگونه یک روزنامه را ریسپانسیو کنیم؟

پاسخ مرحله به مرحله:

1. موبایل:

- تیترها بزرگ

- ستون‌ها عمودی پشت سر هم

- عکس‌ها کامل عرض

2. تبلت:

- دو ستون کنار هم

- عکس‌ها متوسط

- فونت‌ها متوسط

3. دسکتاپ:

- چندین ستون کنار هم

- عکس‌ها بزرگ

- منوهای کناری

مثال‌های واقعی از زندگی

مثال ۱: سینما vs تلویزیون

- سینما (دسکتاپ): صفحه عریض، کیفیت بالا

- تلویزیون (تبلت): صفحه متوسط، کیفیت خوب

- موبایل (موبایل): صفحه کوچک، کیفیت مناسب

مثال ۲: کتاب چاپی vs کتاب الکترونیکی

- کتاب چاپی: layout ثابت، اندازه ثابت

- کتاب الکترونیکی: متن خودکار با صفحه سازگار می‌شود

چرا Responsive Design مهم است؟

۱. کاربران مختلف، دستگاه‌های مختلف

- پدربزرگ با تبلت

- پدر با دسکتاپ

- فرزند با موبایل

۲. آماده برای آینده

- دستگاه‌های جدید با اندازه‌های مختلف

- تلویزیون‌های هوشمند

- ساعت‌های هوشمند

۳. تجربه کاربری بهتر

- نیازی به زوم کردن نیست

- خواندن راحت‌تر است

- navigation ساده‌تر است

🚀 وقتی کدنویسی یاد گرفتید...

جمع‌بندی نهایی

Responsive Design یعنی:

"یک وب‌سایت باهوش که می‌داند روی چه دستگاهی نمایش داده می‌شود و خودش را با آن دستگاه وفق می‌دهد!"

سه کلید طلایی برای یادآوری:

1. انعطاف‌پذیر باش (مثل آب شکل ظرف را بگیر)

2. هوشمند باش (شرایط مختلف را تشخیص بده)

3. ساده باش (همه بتوانند استفاده کنند)

این مفهوم یکی از مهم‌ترین مهارت‌ها در طراحی وب امروزی است!

# بررسی مفهوم Internet

بررسی مفهوم اینترنت (Internet) برای مبتدیان

تعریف ساده اینترنت

اینترنت مثل یک شبکه عظیم جهانی از جاده‌های اطلاعاتی است که کامپیوترهای سراسر جهان را به هم وصل می‌کند.

قیاس ساده: شبکه پست و نامه‌رسانی

تصور کنید اینترنت مثل سیستم پست بین‌المللی است:

- کامپیوترها = خانه‌های مردم

- داده‌ها = نامه‌ها و بسته‌ها

- روترها = اداره‌های پست

- کابل‌ها = جاده‌ها و مسیرهای حمل و نقل

اینترنت چگونه کار می‌کند؟ - یک داستان

داستان ارسال یک عکس از تهران به اصفهان:

1. فرستنده (تهران): عکس را به قطعات کوچک تقسیم می‌کند

2. بسته‌بندی: هر قطعه در یک پاکت با آدرس می‌رود

3. مسیریابی: هر پاکت از چند اداره پست (روتر) عبور می‌کند

4. دریافت (اصفهان): قطعات دوباره جمع می‌شوند و عکس ساخته می‌شود

اجزای اصلی اینترنت

1. سرورها (Servers) - کتابخانه‌های عمومی

- کامپیوترهای همیشه روشن

- اطلاعات را ذخیره و ارائه می‌دهند

- مثل: گوگل، فیسبوک، یوتیوب

2. کلاینت‌ها (Clients) - شما و من

- کامپیوترها، موبایل‌ها، تبلت‌ها

- اطلاعات را request می‌کنند

- مثل: لپ‌تاپ شما، موبایل دوستتان

3. روترها (Routers) - ایستگاه‌های راهنمایی

- داده‌ها را به مسیر درست هدایت می‌کنند

- از ترافیک جلوگیری می‌کنند

4. کابل‌ها (Cables) - جاده‌های اطلاعات

- کابل‌های زمینی، زیردریایی

- داده‌ها را حمل می‌کنند

انواع اتصال به اینترنت

1. ADSL - خط تلفن قدیمی

- مثل استفاده از همان لوله آب برای آشپزی و شستشو

- سرعت متوسط، در دسترس همه

2. فیبر نوری - لوله‌های جدید

- مثل اتوبان 8 بانده

- سرعت بسیار بالا

3. وای‌فای (WiFi) - بی‌سیم

- مثل ارتباط رادیویی

- در محدوده مشخص کار می‌کند

4. دیتا موبایل - اینترنت همراه

- مثل ارتباط تلفنی

- همه جا در دسترس

تاریخچه مختصر اینترنت

دهه 1960: تولد در جنگ سرد

- آمریکا می‌خواست شبکه‌ای بسازد که حتی با حمله اتمی هم کار کند

- نام اولیه: ARPANET

دهه 1980: شبکه دانشگاهی

- دانشگاه‌ها به هم وصل شدند

- ایمیل اختراع شد

دهه 1990: وب جهانی متولد شد

- تیم برنرزلی WWW را اختراع کرد

- مرورگرها ساخته شدند

دهه 2000: انفجار اینترنت

- گوگل، فیسبوک، یوتیوب

- اینترنت پرسرعت

امروز: اینترنت اشیا

- یخچال، ماشین، خانه هوشمند

- همه چیز به اینترنت وصل است

پروتکل‌ها - قوانین اینترنت

HTTP/HTTPS - زبان وب‌سایت‌ها

- مثل زبان رسمی برای صحبت با وب‌سایت‌ها

- HTTPS = HTTP + رمزنگاری (امن)

TCP/IP - زبان اصلی اینترنت

- مثل قوانین رانندگی بین‌المللی

- همه باید این زبان را بلد باشند

SMTP - زبان ایمیل

- مثل قوانین پست برای نامه‌ها

خدمات اصلی اینترنت

1. وب جهانی (WWW)

- وب‌سایت‌ها و صفحات

- مثل: گوگل، آمازون

2. پست الکترونیک (Email)

- ارسال نامه دیجیتال

- مثل: Gmail, Outlook

3. رسانه‌های اجتماعی

- ارتباط با دیگران

- مثل: اینستاگرام، تلگرام

4. ذخیره‌سازی ابری

- نگهداری فایل در اینترنت

- مثل: Google Drive, Dropbox

اینترنت چگونه زندگی ما را تغییر داده؟

ارتباطات:

- قبل: نامه 2 هفته‌ای

- بعد: پیام فوری 2 ثانیه‌ای

اطلاعات:

- قبل: رفتن به کتابخانه

- بعد: جستجوی 2 ثانیه‌ای در گوگل

کسب و کار:

- قبل: مغازه فیزیکی

- بعد: فروشگاه اینترنتی

آموزش:

- قبل: فقط کلاس حضوری

- بعد: دوره‌های آنلاین

امنیت در اینترنت - مثل خانه خود

قفل‌های دیجیتال:

- رمز عبور قوی = کلید محکم

- آنتی‌ویروس = سیستم alarm

- فایروال = دیوار محافظ

خطرات:

- هکرها = سارقان دیجیتال

- ویروس‌ها = بیماری‌های کامپیوتری

- فیشینگ = کلاهبرداری اینترنتی

آینده اینترنت

1. اینترنت 5G

- سرعت باورنکردنی

- تاخیر بسیار کم

2. اینترنت اشیا (IoT)

- همه دستگاه‌ها متصل

- خانه‌های هوشمند

3. واقعیت مجازی

- دنیاهای دیجیتال

- تجربیات غوطه‌وری

4. هوش مصنوعی

- اینترنت هوشمندتر

- پاسخ‌های شخصی‌سازی شده

نکات مهم برای مبتدیان

اینترنت چیست:

- شبکه جهانی کامپیوترها

- سیستم انتقال اطلاعات

- ابزار ارتباط و اطلاعات

اینترنت چیست نیست:

- یک کامپیوتر غول‌پیکر نیست

- یک مکان فیزیکی نیست

- فقط وب‌سایت نیست

جمع‌بندی نهایی

اینترنت مثل سیستم عصبی زمین است:

- مغز: سرورهای بزرگ

- اعصاب: کابل‌ها و امواج

- حواس: کاربران (ما)

سه نکته طلایی:

1. اینترنت شبکه‌ای از شبکه‌هاست

2. همه چیز با قوانین مشخص کار می‌کند

3. ابزاری قدرتمند برای ارتباط و اطلاعات است

اینترنت یکی از مهم‌ترین اختراعات بشر است که جهان را به دهکده‌ای جهانی تبدیل کرده است!

# بررسی مفهوم Protocol

## تعریف ساده پروتکل

پروتکل یک سری قوانین و مقررات است که مشخص می‌کند چگونه دو یا چند سیستم با هم ارتباط برقرار کنند.

قیاس روزمره: قوانین رانندگی

تصور کنید پروتکل مثل قوانین راهنمایی و رانندگی است:

|  |  |
| --- | --- |
| قوانین رانندگی | معادل در پروتکل |
| چراغ قرمز = توقف | ERROR = Stop |
| خط کشی = مسیر حرکت | PATH = Route |
| بوق = سیگنال | SIGNAL = Alert |
| گواهینامه = مجوز | AUTH = Authentication |

پروتکل در ارتباطات - یک داستان

داستان دو نفر که می‌خواهند با زبان‌های مختلف صحبت کنند:

بدون پروتکل:

- 👨‍💼: "Hello!" (انگلیسی)

- 👩‍💼: "؟من متوجه نمی‌شوم" (فارسی)

- نتیجه: ارتباط ناموفق

با پروتکل:

- 👨‍💼: "Hello!" (انگلیسی)

- 👩‍💼: "Hello!" (انگلیسی - طبق پروتکل)

- نتیجه: ارتباط موفق

## چرا پروتکل لازم است؟

### 1. زبان مشترک

- کامپیوترهای مختلف (ویندوز، مک، لینوکس)

- دستگاه‌های مختلف (موبایل، تبلت، کامپیوتر)

- همه باید یک زبان را بفهمند

### 2. قابل پیش‌بینی بودن

- مثل دستورالعمل آشپزی

- همه مراحل مشخص است

- نتیجه یکسان می‌شود

### 3. خطایابی آسان

- اگر مشکلی پیش بیاید، می‌دانیم کجای کار اشتباه شده

- مثل نقشه راه برای عیب‌یابی

## انواع پروتکل‌های اینترنتی

### 1. TCP/IP - پادشاه پروتکل‌ها

مثل سیستم پست پیشرفته

- بسته‌ها را شماره‌گذاری می‌کند

- رسیدن را تأیید می‌کند

- اگر بسته گم شد، دوباره می‌فرستد

### 2. HTTP/HTTPS - پروتکل وب

مثل زبان وب‌سایت‌ها

HTTP: "لطفاً صفحه گوگل را بفرست"

HTTPS: "لطفاً صفحه گوگل را بفرست (رمزگذاری شده)"

### 3. FTP - پروتکل انتقال فایل

مثل پیک مخصوص فایل‌ها

"لطفاً این فایل 1GB را منتقل کن"

### 4. SMTP/POP3 - پروتکل ایمیل

مثل پستچی ایمیل

SMTP: "این ایمیل را بفرست"

POP3: "ایمیل‌های جدید را بگیر"

پروتکل در عمل: ارسال یک ایمیل

مرحله ۱: نوشتن ایمیل (SMTP)

فرستنده: ali@example.com

گیرنده: sara@test.com

موضوع: سلام

متن: چگونه هستی؟

مرحله ۲: بسته‌بندی (TCP)

بسته ۱: [هدر ایمیل]

بسته ۲: [موضوع]

بسته ۳: [متن ایمیل]

مرحله ۳: ارسال (IP)

از: IP آلی

به: IP سارا

مسیر: روتر ۱ → روتر ۲ → روتر ۳

مرحله ۴: دریافت (POP3)

سارا: "ایمیل جدید دارم؟"

سرور: "بله، یک ایمیل از علی"

## ویژگی‌های یک پروتکل خوب

### 1. قابل اعتماد (Reliable)

- داده‌ها سالم می‌رسند

- اگر خطایی باشد، تشخیص داده می‌شود

### 2. کارا (Efficient)

- سریع کار می‌کند

- منابع کم مصرف می‌کند

### 3. امن (Secure)

- ✅ دیگران نمی‌توانند بخوانند

- ✅ هکرها نمی‌توانند تغییر دهند

### 4. قابل توسعه (Extensible)

- می‌توان آن را بهبود داد

- با تکنولوژی‌های جدید کار می‌کند

## پروتکل‌های معروف در زندگی روزمره

### 1. بلوتوث - پروتکل بی‌سیم کوتاه‌برد

- هدفون بی‌سیم

- ماوس و کیبورد

- انتقال فایل بین موبایل‌ها

### 2. WiFi - پروتکل شبکه بی‌سیم

- اینترنت بدون سیم

- ارتباط در محدوده ساختمان

### 3. NFC - پروتکل ارتباط نزدیک

- پرداخت با موبایل

- کارت‌های هوشمند

## مثال ساده: پروتکل دست دادن

پروتکل دست دادن بین دو نفر:

1. قدم ۱: تماس چشمی برقرار کن

2. قدم ۲: دست راست را جلو ببر

3. قدم ۳: محکم اما نه خیلی سفت بگیر

4. قدم ۴: 2-3 بار تکان بده

5. قدم ۵: رها کن

اگر کسی این پروتکل را رعایت نکند:

|  |
| --- |
| - دست دادن خیلی شل → بی‌احترامی  - دست دادن خیلی سفت → تهاجمی  - دست دادن طولانی → ناخوشایند |

## پروتکل‌های امنیتی

### 1. SSL/TLS - پروتکل امن

- مثل حمل پول با خودروی زرهی

- اطلاعات را رمزگذاری می‌کند

- در آدرس وب می‌بینید: `https://`

### 2. VPN - پروتکل شبکه خصوصی

- مثل تونل امن زیرزمینی

- ارتباط را مخفی می‌کند

### 3. SSH - پروتکل امن برای سرور

- مثل کلید ویژه برای اتاق کنترل

- امنیت بالا برای مدیریت سرور

## آینده پروتکل‌ها

1. پروتکل‌های کوانتومی

- امنیت بسیار بالا

- استفاده از فیزیک کوانتوم

2. پروتکل‌های هوش مصنوعی

- ارتباط هوشمند بین دستگاه‌ها

- پیش‌بینی نیازها

3. پروتکل‌های اینترنت اشیا

- میلیاردها دستگاه متصل

- ارتباط کارا و کم‌مصرف

نکات کلیدی برای مبتدیان

## پروتکل چیست:

- مجموعه قوانین ارتباطی

- زبان مشترک بین سیستم‌ها

- تضمین کننده موفقیت ارتباط

## پروتکل چیست نیست:

- یک برنامه یا نرم‌افزار نیست

- سخت‌افزار نیست

- اختیاری نیست (ضروری است)

## جمع‌بندی نهایی

پروتکل مثل زبان مشترک بین ملت‌هاست:

- دستور زبان: قواعد پایه

- واژگان: پیام‌های قابل فهم

- لهجه: انواع مختلف پروتکل

## سه نکته طلایی:

1. پروتکل‌ها زبان مشترک دستگاه‌ها هستند

2. بدون پروتکل‌ها، ارتباط غیرممکن است

3. هر نوع ارتباط، پروتکل مخصوص خود را دارد

پروتکل‌ها پایه و اساس تمام ارتباطات دیجیتال هستند!

# بررسی مفهوم WEB

## تعریف ساده وب

وب مثل یک کتابخانه عظیم جهانی است که صفحات آن در سراسر جهان پخش شده و با هم مرتبط شده‌اند.

قیاس ساده: شهر جهانی اطلاعات

تصور کنید وب مثل یک شهر بسیار بزرگ است:

|  |  |
| --- | --- |
| بخش‌های شهر | معادل در وب |
| خانه‌ها | وب‌سایت‌ها |
| خیابان‌ها | لینک‌ها |
| تابلوهای راهنما | موتورهای جستجو |
| پست‌چی‌ها | مرورگرها |

## تفاوت اینترنت و وب

- اینترنت: جاده‌ها و زیرساخت (سیم‌ها، روترها، سرورها)

- وب: ماشین‌هایی که روی جاده حرکت می‌کنند (وب‌سایت‌ها)

|  |
| --- |
| اینترنت (زیرساخت)  ↓  حمل و نقل داده‌ها  ↓  وب (محتوای قابل مشاهده) |

سه تکنولوژی اصلی وب

1. HTML - اسکلت ساختمان

<!-- مثل نقشه ساختمان -->

<title>فروشگاه من</title>

<header>سربرگ</header>

<nav>منوی</nav>

<main>محتوای اصلی</main>

<footer>پاورقی</footer>

2. CSS - دکوراسیون و زیبایی

/ مثل رنگ، مبلمان، دکور /

|  |
| --- |
| body {  background: lightblue;  font-family: Arial;  }  Header{  color: white;  background: darkblue;  } |

3. JavaScript - برق و سیستم‌های هوشمند

// مثل آسانسور، دزدگیر، سیستم روشنایی

// وقتی کاربر کلیک کند، اتفاقی بیفتد

|  |
| --- |
| button.addEventListener('click', function(){  alert('خوش آمدید!');  }); |

## تاریخچه وب به زبان ساده

1989: تولد وب

- تیم برنرزلی (Tim Berners-Lee) در CERN

- ایده: ارتباط اسناد با هایپرلینک

- اولین وب‌سایت: http://info.cern.ch

1990s: وب 1.0 - وب خواندنی

- صفحات استاتیک

- کاربر فقط می‌توانست بخواند

- مثل روزنامه آنلاین

2000s: وب 2.0 - وب خواندنی/نوشتنی

- صفحات داینامیک

- کاربران محتوا تولید می‌کنند

- مثل فیسبوک، ویکیپدیا

2010s: وب 3.0 - وب هوشمند

- هوش مصنوعی

- شخصی‌سازی محتوا

- اینترنت اشیا

## انواع وب‌سایت‌ها

### 1. وب‌سایت استاتیک (ایستا)

- مثل بروشور کاغذی

- محتوا ثابت است

- تغییر سخت

- مثال: رزومه آنلاین

### 2. وب‌سایت داینامیک (پویا)

- مثل مجله زنده

- محتوا تغییر می‌کند

- کاربران تعامل می‌کنند

- مثال: شبکه‌های اجتماعی

### 3. وب اپلیکیشن‌ها

- مثل نرم‌افزار آنلاین

- عملکرد پیچیده

- مثال: Gmail, Google Docs

## چگونه وب کار می‌کند؟ - یک داستان

داستان بازدید از یک وب‌سایت:

1. شما: آدرس را در مرورگر تایپ می‌کنید

https://www.example.com

2. مرورگر: با DNS صحبت می‌کند

"آدرس IP example.com چیست؟"

"IP آن 93.184.216.34 است"

3. مرورگر: با سرور ارتباط برقرار می‌کند

"لطفاً صفحه اصلی را بفرست"

4. سرور: صفحه را می‌فرستد

"این HTML، CSS و JavaScript شما است."

5. مرورگر: صفحه را نمایش می‌دهد

"صفحه آماده نمایش است!"

## اجزای اصلی تجربه وب

1. مرورگر (Browser) - پنجره به دنیای وب

- گوگل کروم، فایرفاکس، سافاری

- مثل مترجم بین شما و وب‌سایت‌ها

2. موتور جستجو (Search Engine) - فهرست کتابخانه

- گوگل، بینگ، یاندکس

- کمک می‌کند محتوای مورد نظر را پیدا کنید

3. سرور (Server) - انبار اطلاعات

- کامپیوترهای همیشه روشن

- اطلاعات را ذخیره و ارائه می‌دهند

4. دامنه (Domain) - آدرس منحصر به فرد

- example.com

- مثل شماره پلاک خانه

## تکنولوژی‌های سمت سرور vs سمت کلاینت

سمت کلاینت (Front-end) - جلوی صحنه:

- HTML: ساختار

- CSS: ظاهر

- JavaScript: رفتار

سمت سرور (Back-end) - پشت صحنه:

- پایگاه داده: ذخیره اطلاعات

- سرور: پردازش درخواست‌ها

- زبان‌های برنامه‌نویسی: PHP, Python, Java

## طراحی وب مدرن - اصول مهم

1. Responsive Design - طراحی واکنش‌گرا

- صفحه در موبایل، تبلت، دسکتاپ خوب دیده می‌شود

- مثل آب که شکل ظرف را می‌گیرد

2. Accessibility - دسترسی‌پذیری

- افراد با معلولیت هم بتوانند استفاده کنند

- screen readerها بتوانند بخوانند

3. SEO - بهینه‌سازی برای موتورهای جستجو

- کمک به گوگل برای فهمیدن محتوا

- نمایش بهتر در نتایج جستجو

4. Performance - عملکرد

- سریع لود شود

- منابع کم مصرف کند

## امنیت در وب

خطرات رایج:

- هک: دسترسی غیرمجاز

- فیشینگ: کلاهبرداری آنلاین

- بدافزار: نرم‌افزارهای مخرب

## راه‌های محافظت:

- HTTPS: ارتباط امن

- پسورد قوی: مثل کلید محکم

- آپدیت: مثل تعمیر درب و پنجره

## تاثیر وب بر زندگی ما

ارتباطات:

- قبل: نامه، تلفن

- بعد: ایمیل، شبکه‌های اجتماعی، ویدیوکال

اطلاعات:

- قبل: کتابخانه، دایرةالمعارف

- بعد: گوگل، ویکیپدیا

کسب و کار:

- قبل: مغازه فیزیکی

- بعد: فروشگاه اینترنتی

## آینده وب

1. وب 3.0 - وب هوشمند

- هوش مصنوعی

- شخصی‌سازی پیشرفته

- واقعیت مجازی

2. Progressive Web Apps (PWA)

- ترکیب وب‌سایت و اپلیکیشن

- کار آفلاین

- نصب بدون استور

3. Voice Search - جستجوی صوتی

- صحبت با دستگاه‌ها

- دسترسی بدون صفحه‌کلید

4. WebAssembly

- اجرای برنامه‌های سنگین در مرورگر

- مثل نرم‌افزارهای دسکتاپ

## مفاهیم کلیدی برای یادگیری

برای شروع کدنویسی:

1. HTML: ساختار صفحه

2. CSS: زیبایی و layout

3. JavaScript: تعامل و منطق

ابزارهای مفید:

- VS Code: ویرایشگر کد

- Git: کنترل نسخه

- Chrome DevTools: دیباگ کردن

## جمع‌بندی نهایی

وب مثل یک جهان دیجیتال است:

- قاره‌ها: دسته‌های مختلف محتوا

- کشورها: وب‌سایت‌های مختلف

- جاده‌ها: لینک‌ها و ارتباطات

## سه نکته طلایی:

1. وب روی اینترنت ساخته شده است

2. از سه تکنولوژی اصلی تشکیل شده

3. در حال تکامل و هوشمندتر شدن است

وب یکی از دموکراتیک‌ترین اختراعات بشر است که اطلاعات را در دسترس همه قرار داده است!

## شروع یادگیری

اگر می‌خواهید وب را یاد بگیرید:

1. با HTML شروع کنید

2. سپس CSS را بیاموزید

3. در نهایت JavaScript را یاد بگیرید

یادگیری وب امروز = ساختن آینده فردا

# بررسی مفهوم HTTP

تعریف ساده HTTP

HTTP مثل زبان مشترک برای صحبت با وب‌سایت‌ها است. وقتی شما به یک وب‌سایت سر می‌زنید، مرورگر شما با سرور آن وب‌سایت به زبان HTTP صحبت می‌کند.

قیاس ساده: سفارش غذا در رستوران

تصور کنید HTTP مثل فرآیند سفارش غذا است:

|  |  |
| --- | --- |
| مرحله سفارش غذا | معادل در HTTP |
| مشتری منو می‌خواهد | GET /menu |
| مشتری غذا سفارش می‌دهد | POST /order |
| آشپز غذا را آماده می‌کند | Processing |
| گارسون غذا را می‌آورد | 200 OK |
| غذا تمام شده است | 404 Not Found |

HTTP چگونه کار می‌کند؟ - یک داستان

داستان بازدید از وب‌سایت:

کاربر (شما):

|  |
| --- |
| GET /index.html HTTP/1.1  Host: www.example.com |

سرور (وب‌سایت):

|  |
| --- |
| HTTP/1.1 200 OK  Content-Type: text/html  <html>  <head>  <title>  خوش آمدید  </title>  </head>  <body>  محتوا اینجاست  </body>  </html> |

ساختار پایه HTTP

1. درخواست (Request) - چیزی که می‌خواهیم

|  |
| --- |
| GET /about.html HTTP/1.1  Host: example.com  User-Agent: Chrome  Accept-Language: fa |

1. پاسخ (Response) - چیزی که می‌گیریم

|  |
| --- |
| HTTP/1.1 200 OK  Content-Type: text/html  Content-Length: 1256  <html>  ...محتوا...  </html> |

## متدهای اصلی HTTP (Verbs)

1. GET - درخواست اطلاعات

- مثل پرسیدن سوال

- فقط داده می‌گیرد، چیزی تغییر نمی‌دهد

- مثال: مشاهده صفحه، دانلود فایل

2. POST - ارسال اطلاعات

- مثل پر کردن فرم

- داده جدید ایجاد می‌کند

- مثال: ثبت نام، ارسال نظر

3. PUT - آپدیت اطلاعات

- مثل تصحیح اشتباه

- داده موجود را تغییر می‌دهد

- مثال: ویرایش پروفایل

4. DELETE - حذف اطلاعات

- مثل پاره کردن کاغذ

- داده را حذف می‌کند

- مثال: حذف پست

کدهای وضعیت HTTP (Status Codes)

کدهای 1xx: اطلاعی

- `100 Continue`: ادامه بده

کدهای 2xx: موفقیت

- `200 OK`: همه چیز خوب است

- `201 Created`: چیز جدید ساخته شد

- `204 No Content`: موفق اما بدون محتوا

کدهای 3xx: تغییر مسیر

- `301 Moved Permanently`: آدرس تغییر کرده

- `304 Not Modified`: از کش استفاده کن

کدهای 4xx: خطای کلاینت

- `400 Bad Request`: درخواست بد

- `401 Unauthorized`: نیاز به لاگین

- `403 Forbidden`: دسترسی ممنوع

- `404 Not Found`: صفحه پیدا نشد

کدهای 5xx: خطای سرور

- `500 Internal Server Error`: مشکل در سرور

- `503 Service Unavailable`: سرور در دسترس نیست

## تفاوت HTTP و HTTPS

HTTP - معمولی:

- مثل پستکار معمولی

- نامه‌ها قابل خواندن هستند

- مناسب برای اطلاعات عمومی

HTTPS - امن:

- مثل پستکار با خودروی زرهی

- نامه‌ها قفل شده‌اند

- مناسب برای اطلاعات محرمانه

HTTP: http://example.com

HTTPS: https://example.com

## حالت‌های ارتباط HTTP

1. Stateless - بدون حافظه

- هر درخواست مستقل است

- سرور شما را به خاطر نمی‌سپارد

- مثل فروشنده غریبه

2. با کوکی‌ها - حافظه دار

- کوکی هویت شما را ذخیره می‌کند

- سرور شما را می‌شناسد

- مثل فروشنده آشنا

## نسخه‌های HTTP

HTTP/1.1 (1999) - کلاسیک

- یک ارتباط برای هر درخواست

- کند اما سازگار با همه

HTTP/2 (2015) - مدرن

- چندین درخواست همزمان

- سریع‌تر و کارآمدتر

HTTP/3 (2022) - جدید

- از UDP вместо TCP

- بسیار سریع‌تر

HTTP در عمل - مثال‌های واقعی

مثال ۱: ورود به فیسبوک

1. `GET facebook.com` - صفحه لاگین

2. `POST /login` - ارسال نام کاربری و رمز

3. `200 OK` - ورود موفق

4. `GET /home` - صفحه اصلی

مثال ۲: جستجو در گوگل

1. `GET google.com` - صفحه اصلی

2. `GET /search?q=HTTP` - نتایج جستجو

3. `200 OK` - نمایش نتایج

مثال ۳: خرید آنلاین

1. `GET /products` - مشاهده محصولات

2. `POST /cart` - اضافه به سبد

3. `POST /checkout` - پرداخت

4. `201 Created` - سفارش ثبت شد

## هدرهای مهم HTTP

هدرهای درخواست:

- `User-Agent`: مرورگر شما چیست؟

- `Accept`: چه نوع محتوایی می‌خواهی؟

- `Authorization`: هویت شما چیست؟

هدرهای پاسخ:

- `Content-Type`: نوع محتوا (HTML, JSON, etc.)

- `Set-Cookie`: ذخیره اطلاعات شما

- `Cache-Control`: چگونه کش شود؟

## مشاهده HTTP در عمل

با DevTools مرورگر:

1. F12 را بزنید

2. به تب Network بروید

3. یک صفحه را باز کنید

4. درخواست‌های HTTP را ببینید

مثال واقعی:

|  |
| --- |
| Request URL: https://www.google.com/  Request Method: GET  Status Code: 200 OK  Content-Type: text/html |

## ویژگی‌های پیشرفته HTTP

1. Caching - ذخیره موقت

- صفحاتی که تغییر نمی‌کنند را ذخیره می‌کند

- سرعت را افزایش می‌دهد

2. Compression - فشرده‌سازی

- داده‌ها را کوچک می‌کند

- مصرف اینترنت کاهش می‌یابد

3. Pipelining - ارسال چندتایی

- چند درخواست همزمان

- مانند چند مسافر در یک تاکسی

## امنیت در HTTP

مشکلات امنیتی HTTP:

- اطلاعات قابل مشاهده توسط دیگران

- جعل هویت امکان‌پذیر است

- داده‌ها قابل تغییر هستند

راه حل: استفاده از HTTPS

- رمزگذاری end-to-end

- هویت سرور تأیید می‌شود

- داده‌ها محافظت می‌شوند

## نکات کلیدی برای مبتدیان

### HTTP چیست:

- پروتکل ارتباط با وب‌سایت‌ها

- زبان مشترک مرورگر و سرور

- پایه و اساس وب

### HTTP چیست نیست:

- اینترنت نیست

- یک برنامه نیست

- فقط برای وب‌سایت نیست (APIها هم استفاده می‌کنند)

## جمع‌بندی نهایی

HTTP مثل زبان دیپلماتیک بین مرورگرها و سرورهاست:

- دستور زبان: ساختار درخواست و پاسخ

- واژگان: متدها و کدهای وضعیت

- لهجه: هدرها و کوکی‌ها

سه نکته طلایی:

1. HTTP زبان ارتباطی وب است

2. هر عمل در وب یک درخواست HTTP است

3. HTTPS نسخه امن HTTP است

HTTP ستون فقرات ارتباطات وب است و بدون آن، وب امروزی وجود نداشت!

## شروع یادگیری عملی

برای دیدن HTTP در عمل:

1. مرورگر خود را باز کنید

2. F12 بزنید → Network tab

3. به یک وب‌سایت سر بزنید

4. مکالمات HTTP را مشاهده کنید

# بررسی مفهوم Server

بررسی کامل مفهوم سرور (Server) برای مبتدیان

تعریف ساده سرور

سرور یک کامپیوتر همیشه روشن و متصل به اینترنت است که خدمات مختلفی به دیگر کامپیوترها (کلاینت‌ها) ارائه می‌دهد.

قیاس ساده: رستوران

تصور کنید سرور مثل یک رستوران است:

|  |  |
| --- | --- |
| بخش‌های رستوران | معادل در سرور |
| آشپزخانه | پردازش اطلاعات |
| منو | خدمات قابل ارائه |
| گارسون | نرم‌افزار سرور |
| مشتریان | کاربران (کلاینت‌ها) |

## انواع سرورها

1. وب سرور (Web Server) - فروشگاه آنلاین

- وب‌سایت‌ها را میزبانی می‌کند

- مثال: Apache, Nginx

- مثل: یک مغازه که 24 ساعته باز است

2. فایل سرور (File Server) - کتابخانه

- فایل‌ها را ذخیره و به اشتراک می‌گذارد

- مثال: Google Drive, Dropbox

- مثل: کتابخانه عمومی

3. ایمیل سرور (Mail Server) - اداره پست

- ایمیل‌ها را مدیریت می‌کند

- مثال: Gmail, Outlook

- مثل: اداره پست الکترونیکی

4. گیم سرور (Game Server) - زمین بازی

- بازی‌های آنلاین را میزبانی می‌کند

- مثال: سرورهای PUBG, Minecraft

- مثل: زمین ورزشی مجازی

## سرور چگونه کار می‌کند؟ - یک داستان

داستان درخواست یک صفحه وب:

کاربر (کلاینت):

"سلام سرور، لطفاً صفحه اصلی Google را بفرست"

سرور:

"بفرما! این صفحه HTML است...

همینطور فایل‌های CSS و JavaScript"

کاربر:

"ممنون! حالا یک عکس هم نیاز دارم"

سرور:

"عکس هم برات می‌فرستم"

## اجزای اصلی یک سرور

1. سخت‌افزار (Hardware) - جسم سرور

- پردازنده (CPU): مغز سرور

- حافظه (RAM): حافظه موقت

- هارد دیسک (Storage): انبار اطلاعات

- شبکه (Network Card): ارتباط با اینترنت

2. نرم‌افزار (Software) - روح سرور

- سیستم عامل: Windows Server, Linux

- نرم‌افزار سرویس: Apache, MySQL

- امنیت: فایروال، آنتی‌ویروس

3. شبکه (Network) - راه‌های ارتباطی

- پهنای باند: عرض جاده اطلاعات

- آیپی ثابت: آدرس همیشگی

- Uptime: زمان فعالیت

## سرور در مقابل کلاینت

سرور (خدمت‌دهنده):

- ✅ همیشه روشن

- ✅ قدرت پردازش بالا

- ✅ اتصال دائمی به اینترنت

- ✅ خدمات ارائه می‌دهد

## کلاینت (خدمت‌گیرنده):

- ✅ گاهی خاموش/روشن

- ✅ قدرت پردازش معمولی

- ✅ اتصال موقت به اینترنت

- ✅ خدمات دریافت می‌کند

انواع میزبانی سرور

1. سرور اختصاصی (Dedicated Server) - ویلا

- فقط برای شما

- قدرت کامل

- هزینه بالا

- کنترل کامل

2. سرور مجازی (VPS) - آپارتمان

- بخشی از یک سرور بزرگ

- قدرت اشتراکی

- هزینه متوسط

- کنترل نسبی

3. میزبانی اشتراکی (Shared Hosting) - خوابگاه

- یک سرور با چندین کاربر

- قدرت بسیار محدود

- هزینه کم

- کنترل کم

4. ابری (Cloud) - هتل

- منابع قابل تغییر

- پرداخت به میزان استفاده

- انعطاف‌پذیر

- مقیاس‌پذیر

## نمونه‌های واقعی سرور

سرورهای معروف:

- Google Server: جستجو، یوتیوب، جیمیل

- Facebook Server: فیسبوک، اینستاگرام، واتس‌اپ

- Amazon AWS: سرورهای ابری آمازون

- Microsoft Azure: سرورهای مایکروسافت

## امنیت سرور

خطرات رایج:

- حمله DDoS: ترافیک مصنوعی زیاد

- هک: نفوذ به سیستم

- بدافزار: آلوده شدن سرور

راه‌های محافظت:

- فایروال: دیوار محافظ

- آنتی‌ویروس: سیستم ایمنی

- بک‌آپ: نسخه پشتیبان

- آپدیت: به‌روزرسانی منظم

## ویژگی‌های یک سرور خوب

1. قابلیت اطمینان (Reliability)

- همیشه در دسترس باشد

- مثل: برق شهر

2. مقیاس‌پذیری (Scalability)

- بتواند بزرگ شود

- مثل: ساختمان قابل توسعه

3. امنیت (Security)

- در برابر threats مقاوم باشد

- مثل: بانک امن

4. کارایی (Performance)

- سریع پاسخ دهد

- مثل: آمبولانس سریع

## سرورها در زندگی روزمره

وقتی که از سرور استفاده می‌کنید:

- وب‌گردی: سرورهای وب

- ارسال ایمیل: سرورهای ایمیل

- تماشای فیلم: سرورهای استریمینگ

- بازی آنلاین: سرورهای بازی

- ذخیره عکس: سرورهای ابری

چگونه یک سرور راه‌اندازی کنیم؟

مراحل ساده:

1. خرید سرور: سخت‌افزار یا سرویس ابری

2. نصب سیستم عامل: Windows Server یا Linux

3. نصب نرم‌افزار: وب سرور، پایگاه داده

4. پیکربندی شبکه: آیپی، دامنه

5. راه‌اندازی سرویس: وب‌سایت، اپلیکیشن

مفاهیم پیشرفته

1. Load Balancing - تقسیم بار

- چندین سرور با هم کار می‌کنند

- ترافیک بین آنها تقسیم می‌شود

- مثل: چندین گارسون در رستوران شلوغ

2. Clustering - خوشه‌سازی

- چند سرور مثل یک سرور عمل می‌کنند

- اگر یکی خراب شد، بقیه کار می‌کنند

- مثل: تیم پزشکی در اتاق عمل

3. Virtualization - مجازی‌سازی

- یک سرور فیزیکی به چند سرور مجازی تقسیم می‌شود

- صرفه‌جویی در منابع

- مثل: تقسیم یک خانه به چند آپارتمان

تفاوت سرور فیزیکی و مجازی

سرور فیزیکی:

کامپیوتر واقعی

- قدرت کامل

- هزینه بالا

- نگهداری سخت

سرور مجازی (VPS):

- ️ بخشی از یک سرور بزرگ

- ️ قدرت اشتراکی

- هزینه مناسب

- نگهداری آسان

## هزینه‌های سرور

هزینه‌های معمول:

- خرید/اجاره سخت‌افزار

- برق و cooling

- پهنای باند اینترنت

- نگهداری و پشتیبانی

- امنیت و بک‌آپ

جمع‌بندی نهایی

سرور مثل یک خدمتکار وفادار است که:

- همیشه بیدار است

- قدرت زیادی دارد

- به همه خدمات می‌دهد

- در سایه کار می‌کند

سه نکته طلایی:

1. سرورها پایه و اساس اینترنت هستند

2. هر سرویس آنلاین روی سرورها اجرا می‌شود

3. سرورها همیشه در حال کار هستند

بدون سرورها، اینترنتی وجود نداشت و همه سرویس‌های آنلاین از کار می‌افتادند!

🔗 شروع یادگیری عملی

برای کار با سرورها می‌توانید:

1. یک VPS ارزان خریداری کنید

2. با Linux آشنا شوید

3. یک وب سرور نصب کنید

4. یک وب‌سایت ساده راه‌اندازی کنید

یادگیری سرورها = فهمیدن پایه‌های اینترنت

## چندپارادایمی:

پشتیبانی از برنامه‌نویسی شیءگرا، تابعی و رویه‌ای: توضیح چندپارادایمی بودن جاوااسکریپت

جاوااسکریپت یک زبان برنامه‌نویسی چندپارادایمی است که از سبک‌های مختلف برنامه‌نویسی پشتیبانی می‌کند. در این مثال، هر سه پارادایم اصلی را نمایش می‌دهم.

### برنامه نویسی تابعی یا رویه‌ایی:

|  |
| --- |
| // برنامه‌نویسی رویه‌ای تمرکز بر توابع و رویه‌ها  // محاسبه مساحت دایره  function calculateCircleArea(radius) {  return Math.PI radius radius;  }  // محاسبه محیط دایره  function calculateCircleCircumference(radius) {  return 2 Math.PI radius;  }  // استفاده از توابع  const radius = 5;  const area = calculateCircleArea(radius);  const circumference = calculateCircleCircumference(radius);  console.log(`مساحت دایره: ${area}`);  console.log(`محیط دایره: ${circumference}`); |

### برنامه نویسی شیئ گرا (ObjectOriented Programming)

|  |
| --- |
| // برنامه‌نویسی شیءگرا استفاده از کلاس‌ها و اشیا  // تعریف کلاس  class Person {      constructor(name, age) {          this.name = name;          this.age = age;      }        // متد      greet() {          return `سلام، من ${this.name} هستم و ${this.age} سال دارم.`;      }        // متد دیگر      haveBirthday() {          this.age++;          return `تولدت مبارک! حالا ${this.age} ساله شدی.`;      }  }  // ارث‌بری  class Student extends Person {      constructor(name, age, studentId) {          super(name, age);          this.studentId = studentId;      }        study() {          return `${this.name} در حال مطالعه است.`;      }  }  // ایجاد نمونه‌ها  const person1 = new Person("امین", 30);  const student1 = new Student("سارا", 22, "12345");  console.log(person1.greet());  console.log(student1.greet());  console.log(student1.study());  console.log(person1.haveBirthday()); |

[مثال بالا در دنیای وب](webdesign/js/Object/js/ObjectOrientedProgramming.js)

### برنامه‌نویسی تابعی (Functional Programming)

|  |
| --- |
| // برنامه‌نویسی تابعی استفاده از توابع خالص و توابع مرتبه بالا  // توابع خالص (همان ورودی → همیشه همان خروجی)  const add = (a, b) => a + b;  const multiply = (a, b) => a b;  // تابعی که تابع برمی‌گرداند (تابع ساز)  const createMultiplier = (factor) => (number) => number factor;  // توابع مرتبه بالا (HigherOrder Functions)  const numbers = [1, 2, 3, 4, 5];  // map تبدیل آرایه  const doubled = numbers.map(n => n 2);  // filter فیلتر کردن آرایه  const evens = numbers.filter(n => n % 2 === 0);  // reduce کاهش آرایه به یک مقدار  const sum = numbers.reduce((acc, n) => acc + n, 0);  // ترکیب توابع  const pipe = (...fns) => (x) => fns.reduce((v, f) => f(v), x);  const double = x => x 2;  const increment = x => x + 1;  const square = x => x x;  const transform = pipe(double, increment, square);  const result = transform(5); // (52 + 1)^2 = 121  console.log("اعداد دوبرابر:", doubled);  console.log("اعداد زوج:", evens);  console.log("مجموع اعداد:", sum);  console.log("نتیجه ترکیب توابع:", result); |

[مثال بالا در دنیای وب](webdesign/js/Object/js/FunctionalProgramming.js)

ترکیب پارادایم‌ها

|  |
| --- |
| // ترکیب پارادایم‌های مختلف  // رویه‌ای + تابعی  function processUserData(users, filterFunc, mapFunc) {      return users          .filter(filterFunc)          .map(mapFunc);  }  // شیءگرا + تابعی  class Calculator {      constructor() {          this.operations = [];      }        // ذخیره عملیات به صورت تابع      addOperation(name, operationFunc) {          this.operations.push({ name, func: operationFunc });      }        executeOperation(name, ...args) {          const operation = this.operations.find(op => op.name === name);          if (operation) {              return operation.func(...args);          }          throw new Error(`Operation ${name} not found`);      }  }  // استفاده از ترکیب پارادایم‌ها  const users = [      { name: "Ali", age: 25, active: true },      { name: "Sara", age: 30, active: false },      { name: "Reza", age: 22, active: true }  ];  const activeUsers = processUserData(      users,      user => user.active,  // تابع فیلتر (تابعی)  (user) => {return user.active;}      user => user.name      // تابع تبدیل (تابعی) (user) => {return user.name;}  ); // call processUserData() function with specefic arugomans  console.log("کاربران فعال:", activeUsers);  // ایجاد ماشین حساب و اضافه کردن عملیات  const calc = new Calculator();  // شیءگرا  calc.addOperation("add", (a, b) => a + b);  // تابعی  calc.addOperation("multiply", (a, b) => a b);  console.log("جمع:", calc.executeOperation("add", 5, 3));  console.log("ضرب:", calc.executeOperation("multiply", 5, 3)); |

[ترکیب پارادایم‌ها](webdesign/js/Object/js/paradim.js)

این مثال‌ها نشان می‌دهند که چگونه جاوااسکریپت از چندین پارادایم برنامه‌نویسی پشتیبانی می‌کند و چگونه می‌توان این پارادایم‌ها را با هم ترکیب کرد تا کدهای انعطاف‌پذیر و قدرتمندی ایجاد شود.

[آموزش استفاده از map](#_Map_در_جاوا)   
[آموزش استفاده از filter](#_آموزش_استفاده_از)

[آموزش استفاده از reduce](#_آموزش_استفاده_از_1)

[آموزش استفاده از Rest parameters](#_آموزش_استفاده_از_2) و [آموزش استفاده از Spread Operator](#_آموزش_استفاده_از_2)

جاوااسکریپت به شما امکان می‌دهد از هر یک از این پارادایم‌ها یا ترکیبی از آن‌ها استفاده کنید که این انعطاف‌پذیری باعث می‌شود بتوانید بهترین روش را برای حل هر مسئله انتخاب نمایید.

کاربردهای اصلی:

افزودن تعامل به صفحات وب

ایجاد برنامه‌های تحت وب (Web Applications)

توسعه سمت سرور با Node.js

ساخت اپلیکیشن‌های موبایل

توسعه بازی‌های تحت وب

|  |
| --- |
| // نمایش پیام در کنسول  console.log("Hello, World!");  // تعریف متغیر  let name = "John";  const age = 25;  // تابع ساده  function greet(person) {      return "Hello, " + person + "!";  }  // استفاده از تابع  console.log(greet(name)); |

[مثال ساده](webdesign/js/Object/js/simple.js)

محیط‌های اجرا:

مرورگرهای وب (Chrome, Firefox, Safari)

Node.js برای اجرای سمت سرور

JavaScript امروزه به یکی از محبوب‌ترین زبان‌های برنامه‌نویسی جهان تبدیل شده و تقریباً در تمام وبسایت‌های مدرن استفاده می‌شود.

## آموزش استفاده از Map در جاوا اسکریپت

### مقدمه‌ای بر Map

`Map` یک ساختار داده‌ای در جاوااسکریپت است که به شما امکان ذخیره‌سازی جفت‌های کلید-مقدار (key-value) را می‌دهد. برخلاف Object، کلیدهای Map می‌توانند از هر نوع داده‌ای باشند.

|  |
| --- |
| // ایجاد یک Map جدید  const myMap = new Map();  // یا مقداردهی اولیه با آرایه‌ای از جفت‌های [key, value]  const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  console.log(initializedMap.get("name")); |

[Map()](webdesign/js/Object/js/map.js)

#### نکته برای دسترسی به عناصر Map از متد `get()` استفاده می‌شود. در مورد مثال شما:

##### روش‌های دسترسی به مقادیر Map

1. استفاده از متد `get()`

|  |
| --- |
| const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  // دسترسی به مقدار 'name'  const nameValue = initializedMap.get('name');  console.log(nameValue); // خروجی: 'Alice'  // دسترسی به سایر مقادیر  const numberValue = initializedMap.get(1);  console.log(numberValue); // خروجی: 'number one'  const booleanValue = initializedMap.get(true);  console.log(booleanValue); // خروجی: 'boolean true' |

[get()](webdesign/js/Object/js/get.js)

* 1. بررسی وجود کلید قبل از دسترسی

|  |
| --- |
| // بررسی وجود کلید قبل از دسترسی  const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  if (initializedMap.has('name')) {  const name = initializedMap.get('name');  console.log('Name found:', name); // خروجی: 'Name found: Alice'  } else {  console.log('Key not found');  } |

[get()  بررسی وجود کلید قبل از دسترسی](webdesign/js/Object/js/getExist.js)

1. دسترسی به همه مقادیر با تکرار (Iteration)

|  |
| --- |
| // getIteration.js  const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  // روش ۱: استفاده از for...of  for (const [key, value] of initializedMap) {  console.log(key, ':', value);  }  // خروجی:  // name : Alice  // 1 : number one  // true : boolean true  // روش ۲: استفاده از forEach  initializedMap.forEach((value, key) => {  console.log(`${key} → ${value}`);  }); |

[getIteration](webdesign/js/Object/js/getIteration.js)

1. دریافت همه کلیدها و مقادیر

|  |
| --- |
| // getAll.js  const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  // دریافت همه کلیدها  const keys = Array.from(initializedMap.keys());  console.log(keys); // خروجی: ['name', 1, true]  // دریافت همه مقادیر  const values = Array.from(initializedMap.values());  console.log(values); // خروجی: ['Alice', 'number one', 'boolean true']  // دریافت همه entries به صورت آرایه  const entries = Array.from(initializedMap.entries());  console.log(entries);  // خروجی: [['name', 'Alice'], [1, 'number one'], [true, 'boolean true']] |

[getAll.js](webdesign/js/Object/js/getAll.js)

##### مثال‌های کاربردی بیشتر

مثال ۱: مدیریت خطا برای کلیدهای ناموجود

|  |
| --- |
| const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  function safeGet(map, key, defaultValue = null) {  return map.has(key) ? map.get(key) : defaultValue;  }  const name = safeGet(initializedMap, 'name', 'Unknown');  const age = safeGet(initializedMap, 'age', 25); // کلید وجود ندارد  console.log(name); // 'Alice'  console.log(age); // 25 (مقدار پیش‌فرض) |

[getDefault](webdesign/js/Object/js/getDefault.js)

مثال ۲: استفاده در تابع

|  |
| --- |
| const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  function getUserInfo(map) {  return {  name: map.get('name') || 'N/A',  id: map.get('id') || 0,  isActive: map.get('isActive') || false  };  }  const userInfo = getUserInfo(initializedMap);  console.log(userInfo);  // خروجی: { name: 'Alice', id: 0, isActive: false } |

[get() استفاده در تابع](webdesign/js/Object/js/getFunction.js)

مثال ۳: تبدیل Map به Object

|  |
| --- |
| function mapToObject(map) {  const obj = {};  for (const [key, value] of map) {  obj[key] = value;  }  return obj;  }  const mapObject = mapToObject(initializedMap);  console.log(mapObject.name); // 'Alice'  console.log(mapObject[1]); // 'number one'  console.log(mapObject[true]); // 'boolean true' |

[getToObject](webdesign/js/Object/js/getToObject.js)

##### نکات مهم

1. کلیدها case-sensitive هستند:

|  |
| --- |
| const myMap = new Map([['NAME', 'Bob']]);  console.log(myMap.get('name')); // undefined  console.log(myMap.get('NAME')); // 'Bob' |

1. اشیا به عنوان کلید:

|  |
| --- |
| const objKey = { id: 1 };  const mapWithObject = new Map([[objKey, 'object value']]);  console.log(mapWithObject.get(objKey)); // 'object value'  console.log(mapWithObject.get({ id: 1 })); // undefined (اشیا مختلف) |

1. مقدار پیش‌فرض برای کلیدهای ناموجود:

|  |
| --- |
| // استفاده از || برای مقدار پیش‌فرض  const value = initializedMap.get('nonexistent') || 'Default Value';  console.log(value); // 'Default Value'  // استفاده از Nullish Coalescing (??)  const value2 = initializedMap.get('nonexistent') ?? 'Default';  console.log(value2); // 'Default' |

جمع‌بندی

برای دسترسی به مقادیر Map از `map.get(key)` استفاده کنید. همیشه بهتر است ابتدا با `map.has(key)` وجود کلید را بررسی کنید تا از خطاهای `undefined` جلوگیری شود.

|  |
| --- |
| const initializedMap = new Map([  ['name', 'Alice'],  [1, 'number one'],  [true, 'boolean true']  ]);  // بهترین روش دسترسی  if (initializedMap.has('name')) {  const name = initializedMap.get('name');  console.log('Name:', name); // خروجی: 'Name: Alice'  }  // یا با مقدار پیش‌فرض  const name = initializedMap.get('name') || 'Unknown';  console.log(name); // 'Alice' |

تفاوت Map با Object

#### مقایسه Map و Object

|  |  |  |
| --- | --- | --- |
| ویژگی | Map | Map |
| نوع کلیدها | هر نوع داده‌ای | فقط String یا Symbol |
| ترتیب عناصر | حفظ ترتیب درج | ترتیب تضمین شده نیست |
| سایز | size` property | باید manually محاسبه شود |
| پرفورمنس | بهتر برای افزودن/حذف مکرر | بهتر برای موارد ساده |
| تکرار | built-in | نیاز به Object.keys() دارد |

مثال مقایسه‌ای:

|  |
| --- |
| // Object  const obj = {};  obj[1] = 'number'; // کلید به string تبدیل می‌شود: '1'  obj['1'] = 'string'; // مقدار قبلی overwrite می‌شود  // Map  const map = new Map();  map.set(1, 'number'); // کلید number می‌ماند  map.set('1', 'string'); // کلید جداگانه برای string  console.log(obj); // { '1': 'string' }  console.log(map); // Map(2) { 1 → 'number', '1' → 'string' } |

#### متدهای اصلی Map

|  |
| --- |
| const map = new Map();  // 1. set() -  map.set('name', 'Alice');  map.set('age', 30);  map.set('name', 'Bob'); // Update مقدار  // 2. get() - دریافت مقدار  console.log(map.get('name')); // 'Bob'  console.log(map.get('age')); // 30  // 3. has() - بررسی وجود کلید  console.log(map.has('name')); // true  console.log(map.has('city')); // false  // 4. delete() - حذف یک کلید  map.delete('age');  console.log(map.has('age')); // false  // 5. clear() - حذف همه عناصر  map.clear();  console.log(map.size); // 0  // 6. size - تعداد عناصر  console.log(map.size); // 0 |

[getMethod](webdesign/js/Object/js/getMethod.js)

متدهای تکرار (Iteration):

|  |
| --- |
| const userMap = new Map([  ['name', 'Alice'],  ['age', 30],  ['city', 'New York']  ]);  // 1. keys() - کلیدها  for (const key of userMap.keys()) {  console.log(key); // 'name', 'age', 'city'  }  // 2. values() - مقادیر  for (const value of userMap.values()) {  console.log(value); // 'Alice', 30, 'New York'  }  // 3. entries() - جفت‌های [key, value] (پیش‌فرض)  for (const [key, value] of userMap.entries()) {  console.log(key, value);  }  // 4. forEach() - مانند آرایه  userMap.forEach((value, key) => {  console.log(key, value);  }); |

[mapIteration2](webdesign/js/Object/js/mapIteration2.js)

#### مثال‌های کاربردی

مثال 1: مدیریت کاربران

|  |
| --- |
| class UserManager {  constructor() {  this.users = new Map();  }  addUser(id, userData) {  this.users.set(id, { ...userData, createdAt: new Date() });  }  getUser(id) {  return this.users.get(id);  }  updateUser(id, updates) {  if (this.users.has(id)) {  const user = this.users.get(id);  this.users.set(id, { ...user, ...updates });  }  }  deleteUser(id) {  this.users.delete(id);  }  getAllUsers() {  return Array.from(this.users.entries());  }  }  // استفاده  const manager = new UserManager();  manager.addUser(1, { name: 'Alice', email: 'alice@example.com' });  manager.addUser(2, { name: 'Bob', email: 'bob@example.com' });  console.log(manager.getUser(1)); |

[mapUserManagement](webdesign/js/Object/js/mapUserManagement.js)

مثال 2: کش داده‌ها (Caching)

|  |
| --- |
| class DataCache {  constructor(maxSize = 100) {  this.cache = new Map();  this.maxSize = maxSize;  }  set(key, value) {  if (this.cache.size >= this.maxSize) {  // حذف قدیمی‌ترین آیتم (اولین آیتم)  const firstKey = this.cache.keys().next().value;  this.cache.delete(firstKey);  }  this.cache.set(key, value);  }  get(key) {  if (this.cache.has(key)) {  const value = this.cache.get(key);  // بروزرسانی به عنوان جدیدترین آیتم  this.cache.delete(key);  this.cache.set(key, value);  return value;  }  return null;  }  clear() {  this.cache.clear();  }  } |

[mapCaching](webdesign/js/Object/js/mapCaching.js)

مثال 3: شمارش تکرار عناصر

|  |
| --- |
| // mapCounting.js  function countOccurrences(array) {  const countMap = new Map();    for (const item of array) {  console.log("item",item);  console.log("countMap.get(item):",countMap.get(item));  const currentCount = countMap.get(item) || 0;  console.log("currentCount:",currentCount);  countMap.set(item, currentCount + 1);  }    return countMap;  }  const fruits = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple'];  const fruitCounts = countOccurrences(fruits);  console.log(fruitCounts);  // Map(3) { 'apple' → 3, 'banana' → 2, 'orange' → 1 } |

[mapCounting](webdesign/js/Object/js/mapCounting.js)

مثال 4: تبدیل بین ساختارها

|  |  |
| --- | --- |
| // تبدیل Object به Map  // const obj = { a: 1, b: 2, c: 3 };  // const mapFromObj = new Map(Object.entries(obj));  // // تبدیل Map به Object  // const objFromMap = Object.fromEntries(mapFromObj);  // console.log(objFromMap);  // // تبدیل Map به Array  // const arrayFromMap = Array.from(mapFromObj);  // console.log(arrayFromMap);  // // یا  // const arrayFromMap2 = [...mapFromObj];  // console.log(arrayFromMap2);  // // تبدیل Array به Map  // const array = [['a', 1], ['b', 2], ['c', 3]];  // const mapFromArray = new Map(array);  // console.log(mapFromArray); |  |

[mapToObject2](webdesign/js/Object/js/mapToObject2.js)

### نکات پیشرفته

استفاده از کلیدهای Object

|  |
| --- |
| const user1 = { id: 1, name: 'Alice' };  const user2 = { id: 2, name: 'Bob' };  const userSettings = new Map();  userSettings.set(user1, { theme: 'dark', language: 'en' });  userSettings.set(user2, { theme: 'light', language: 'fa' });  console.log(userSettings.get(user1)); // { theme: 'dark', language: 'en' } |

[mapObjectKeys](webdesign/js/Object/js/mapObjectKeys.js)

#### کلیدهای NaN

|  |
| --- |
| const map = new Map();  map.set(NaN, 'This is NaN');  console.log(map.get(NaN)); // 'This is NaN'  console.log(map.get(Number('abc'))); // 'This is NaN' - چون NaN === NaN |

[mapNaN](webdesign/js/Object/js/mapNaN.js)

#### زنجیره‌ای کردن متد set

|  |
| --- |
| const map = new Map();  // زنجیره‌ای کردن  map.set('a', 1)  .set('b', 2)  .set('c', 3);  console.log(map); // Map(3) { 'a' → 1, 'b' → 2, 'c' → 3 } |

[mapChainSets](webdesign/js/Object/js/mapChainSets.js)

استفاده با توابع

|  |
| --- |
| function createCounter() {  const counts = new Map();    return {  increment(key) {  counts.set(key, (counts.get(key) || 0) + 1);  },  getCount(key) {  return counts.get(key) || 0;  },  getAllCounts() {  return Array.from(counts.entries());  }  };  }  const counter = createCounter();  counter.increment('apple');  counter.increment('banana');  counter.increment('apple');  console.log(counter.getCount('apple')); // 2  console.log(counter.getAllCounts()); // [['apple', 2], ['banana', 1]] |

[mapFunctions2](webdesign/js/Object/js/mapFunctions2.js)

##### جمع‌بندی

چه زمانی از Map استفاده کنیم:

1. کلیدهای غیر-string نیاز دارید

2. ترتیب عناصر مهم است

3. تعداد عناصر را زیاد نیاز دارید

4. عملکرد افزودن/حذف مکرر مهم است

5. نیاز به تکرار ساده دارید

##### مزایای Map:

- کلیدهای با انواع مختلف

- حفظ ترتیب درج

- سایز built-in

- تکرار آسان

- پرفورمنس بهتر برای عملیات مکرر

محدودیت‌ها:

- تبدیل به JSON نیاز به کار اضافه دارد

- سینتکس ساده‌تر Object برای موارد ساده

|  |
| --- |
| // تبدیل Map به JSON و برعکس  const map = new Map([['name', 'Alice'], ['age', 30]]);  // به JSON  const json = JSON.stringify(Array.from(map.entries()));  // از JSON  const reconstructedMap = new Map(JSON.parse(json)); |

[map()   JSON](webdesign/js/Object/js/mapJSON.js)

Map یک ابزار قدرتمند در جاوااسکریپت است که در بسیاری از سناریوها می‌تواند جایگزین بهتری برای Object باشد، به خصوص وقتی که با داده‌های پویا و پیچیده کار می‌کنید.

### تمرینات اضافه

1. مپ کردن آرایه:

|  |
| --- |
| const array = [1, 4, 9, 16];  // Pass a function to map  const mapped = array.map((x) => x 2);  console.log(mapped);  // Expected output: Array [2, 8, 18, 32] |

[mapArray](webdesign/js/Object/js/mapArray.js)

1. مپ کردن آرایه با مربع عناصر:

|  |
| --- |
| //js/mapSqrt.js  const numbers = [1, 4, 9];  const roots = numbers.map((num) => Math.sqrt(num));  // roots is now [1, 2, 3]  // numbers is still [1, 4, 9]  console.log(numbers);  console.log(roots); |

[mapSqrt](webdesign/js/Object/js/mapSqrt.js)

1. تبدیل آرایه رشته‌ها به عناصر li با استفاده از Map در جاوااسکریپت

روش‌های مختلف برای تبدیل آرایه به عناصر li

استفاده از `map()` و `join()`

|  |
| --- |
| const fruits = ['سیب', 'موز', 'پرتقال', 'انگور'];  // روش 1: استفاده از map و join  const htmlList = fruits.map(fruit => `<li>${fruit}</li>`).join('\n');  console.log(htmlList);  // خروجی:  // <li>سیب</li>  // <li>موز</li>  // <li>پرتقال</li>  // <li>انگور</li>  // قرار دادن در ul  const fullHtml = `<ul>\n${htmlList}\n</ul>`;  console.log(fullHtml); |

[Map join()](webdesign/js/Object/js/mapJoin.js)

2. ایجاد عناصر DOM واقعی

|  |
| --- |
| const fruits = ['سیب', 'موز', 'پرتقال', 'انگور'];  // روش 2: ایجاد عناصر DOM واقعی  const listItems = fruits.map(fruit => {  const li = document.createElement('li');  li.textContent = fruit;  li.className = 'fruit-item';  return li;  });  // اضافه کردن به DOM  const ulElement = document.createElement('ul');  ulElement.append(...listItems);  document.body.appendChild(ulElement); |

[mapDOM](webdesign/js/Object/js/mapDOM.js)

1. استفاده با template literals

|  |
| --- |
| const fruits = ['سیب', 'موز', 'پرتقال', 'انگور'];  // روش 3: با قالب‌بزی پیشرفته  const htmlList = fruits.map(fruit => `  <li class="fruit-item" data-fruit="${fruit}">  <span>${fruit.toUpperCase()}</span>  </li>  `).join('');  const fullHtml = `  <ul class="fruits-list">  ${htmlList}  </ul>  `;  console.log(fullHtml); |

[mapLiterals](webdesign/js/Object/js/mapLiterals.js)

مثال‌های کاربردی

مثال 1: لیست کاربران

|  |
| --- |
| const users = ['علی محمدی', 'فاطمه احمدی', 'محمد رضایی', 'زهرا حسینی'];  const userList = users.map(user => `  <li class="user-item">  <i class="icon-user"></i>  <span>${user}</span>  </li>  `).join('');  const usersHtml = `  <div class="users-container">  <h3>لیست کاربران</h3>  <ul class="users-list">  ${userList}  </ul>  </div>  `;  // قرار دادن در DOM  document.getElementById('app').innerHTML = usersHtml; |

[mapUserList](webdesign/js/Object/js/mapUserList.js)

مثال 2: لیست با شماره

|  |
| --- |
| const tasks = ['تمرین جاوااسکریپت', 'پروژه وب', 'مطالعه ریاضی', 'ورزش'];  const numberedList = tasks.map((task, index) => `  <li class="task-item">  <span class="task-number">${index + 1}.</span>  <span class="task-text">${task}</span>  <button onclick="completeTask(${index})">انجام شد</button>  </li>  `).join('');  const tasksHtml = `  <div class="tasks-wrapper">  <h2>لیست کارهای روزانه</h2>  <ol class="tasks-list">  ${numberedList}  </ol>  </div>  `; |

[mapUserOlList](webdesign/js/Object/js/mapUserOlList.js)

مثال 3: لیست با آیکون

|  |
| --- |
| const features = [  'پشتیبانی 24/7',  'امنیت بالا',  'رابط کاربری آسان',  'پشتیبانی از تمام دستگاه‌ها'  ];  const icons = ['🕒', '🔒', '💻', '📱'];  const featureList = features.map((feature, index) => `  <li class="feature-item">  <span class="feature-icon">${icons[index]}</span>  <span class="feature-text">${feature}</span>  </li>  `).join('');  const featuresHtml = `  <section class="features">  <h2>ویژگی‌های محصول</h2>  <ul class="features-list">  ${featureList}  </ul>  </section>  `; |

[mapListIcon](webdesign/js/Object/js/mapListIcon.js)

بهینه‌سازی عملکرد

استفاده از [DocumentFragment](#_آموزش_کار_با) برای عملکرد بهتر

|  |
| --- |
| const fruits = ['سیب', 'موز', 'پرتقال', 'انگور'];  function createListItems(items) {  const fragment = document.createDocumentFragment();    items.map(item => {  const li = document.createElement('li');  li.textContent = item;  li.className = 'fruit-item';  fragment.appendChild(li);  return li;  });    return fragment;  }  const ulElement = document.createElement('ul');  ulElement.appendChild(createListItems(fruits));  // فقط یک بار DOM را آپدیت می‌کنیم  document.getElementById('app').appendChild(ulElement); |

[mapDocumentFragment](webdesign/js/Object/js/mapDocumentFragment.js)

استفاده از innerHTML برای کارایی بالا

|  |
| --- |
| const fruits = ['سیب', 'موز', 'پرتقال', 'انگور'];  function renderList(items) {  const html = items.map(item => `<li>${item}</li>`).join('');  return `<ul>${html}</ul>`;  }  document.getElementById('app').innerHTML = renderList(fruits); |

[mapinnerHTML](webdesign/js/Object/js/mapinnerHTML.js)

تابع قابل استفاده مجدد

|  |
| --- |
| function arrayToHtmlList(items, options = {}) {  const {  listType = 'ul',  itemClass = '',  listClass = '',  transform = item => item  } = options;    const listItems = items.map(item => `  <li class="${itemClass}">${transform(item)}</li>  `).join('');    return `  <${listType} class="${listClass}">  ${listItems}  </${listType}>  `;  }  // استفاده  const fruits = ['سیب', 'موز', 'پرتقال', 'انگور'];  const html = arrayToHtmlList(fruits, {  listType: 'ul',  listClass: 'fruits-list',  itemClass: 'fruit-item',  transform: fruit => `<strong>${fruit}</strong>`  });  let app = document.getElementById("app");  app.innerHTML = html; |

[mapFunction](webdesign/js/Object/js/mapFunction.js)

جمع‌بندی

استفاده از `map()` برای تبدیل آرایه رشته‌ها به عناصر li بهترین روش است زیرا:

1. کد تمیز و خوانا

2. قابل استفاده مجدد

3. عملکرد عالی ⚡

4. قابل ترکیب با سایر متدها 🧩

|  |
| --- |
| // mapSimple2.js  // ساده‌ترین روش  const items = ['آیتم ۱', 'آیتم ۲', 'آیتم ۳'];  const html = items.map(item => `<li>${item}</li>`).join('');  // یا برای عناصر واقعی  const listElements = items.map(item => {      const li = document.createElement('li');      li.textContent = item;      return li;  });  let app = document.getElementById("app");  app.innerHTML = html;  listElements.forEach(li => app.appendChild(li)); |

[mapSimple2](webdesign/js/Object/js/mapSimple2.js)

این روش برای ایجاد لیست‌های پویا، منوها، نتایج جستجو و هر نوع لیستی در وب اپلیکیشن‌ها عالی کار می‌کند.

## آموزش کار با createDocumentFragment()

یک نود خالی برای درج عناصر Html درون آن. DocumentFragment جزئی از درخت نود صفحه ما نیست. این عنصر بعداً در Html درج خواهد شد و هنگام درج تنها فرزندانش را در نود صفحه مشاهده خواهید کرد. استفاده از DocumentFragment باعث بهبود عملکرد در برخی مرورگرهای قدیمی‌تر می‌شود.

|  |
| --- |
| const fragment = new DocumentFragment(); |

|  |
| --- |
| const element = document.getElementById("app"); // assuming ul exists  const fragment = document.createDocumentFragment();  const browsers = ["Firefox", "Chrome", "Opera", "Safari"];  browsers.forEach((browser) => { const li = document.createElement("li"); li.textContent = browser; fragment.appendChild(li); });  element.appendChild(fragment); |

[createDocumentFragment](webdesign/js/Object/js/createDocumentFragment.js)

## آموزش استفاده از filter

جاوااسکریپت یکی از قدرتمندترین زبان‌های برنامه‌نویسی برای توسعه وب است و متد `filter()` یکی از ابزارهای کاربردی برای کار با آرایه‌ها محسوب می‌شود. در ادامه به طور کامل با نحوه استفاده از این متد آشنا می‌شویم.

### آموزش جامع استفاده از متد filter() در جاوااسکریپت

### مقدمه‌ای بر متد filter()

متد `filter()` یک متد داخلی برای آرایه‌ها در جاوااسکریپت است که برای فیلتر کردن عناصر آرایه بر اساس یک شرط خاص به کار می‌رود. این متد یک آرایه جدید شامل عناصری که شرط مشخص شده را دارند برمی‌گرداند و آرایه اصلی را تغییر نمی‌دهد.

#### ویژگی‌های کلیدی:

تغییر ندادن آرایه اصلی (immutable)

بازگرداندن آرایه جدید با عناصر فیلتر شده

پشتیبانی از تمام مرورگرهای مدرن

این متد بخشی از ویژگی‌های استاندارد ECMAScript 5 (ES5) است و در همه محیط‌های اجرایی جاوااسکریپت از جمله مرورگرها و Node.js در دسترس می‌باشد .

### سینتکس و پارامترها

سینتکس اصلی متد filter() به صورت زیر است:

|  |
| --- |
| let newArray = array.filter(function(element, index, array) {  // شرط انتخاب عنصر  }, thisArg); |

#### پارامترها:

|  |  |  |
| --- | --- | --- |
| پارامتر | توضیح | اختیاری |
| element | عنصر فعلی که پردازش می‌شود | اجباری |
| index | ایندکس عنصر فعلی در آرایه |  |
| array | آرایه اصلی که filter روی آن فراخوانی شده |  |
| thisArg | مقداری که به عنوان this برای تابع استفاده می‌شود |  |

#### مقدار بازگشتی:

یک آرایه جدید شامل تمام عناصری که تابع callback شرط true را برگرداند

اگر هیچ عنصری شرط را برآورده نکند، یک آرایه خالی بازگردانده می‌شود

### موارد استفاده و مثال‌های کاربردی

مثال 1: فیلتر کردن اعداد زوج

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  const evenNumbers = numbers.filter(number => number % 2 === 0);  console.log(evenNumbers); // خروجی: [2, 4, 6, 8, 10] |

[filterEven](webdesign/js/Object/js/filterEven.js)

مثال 2: فیلتر کردن بر اساس طول رشته

|  |
| --- |
| const words = ['hello', 'world', 'javascript', 'filter', 'code', 'web'];  const longWords = words.filter(word => word.length > 5);  console.log(longWords); // خروجی: ['javascript', 'filter'] |

[filterLength](webdesign/js/Object/js/filterLength.js)

مثال 3: فیلتر کردن اشیاء در آرایه

|  |
| --- |
| const products = [    { name: 'Laptop', price: 1000, category: 'electronics' },    { name: 'Book', price: 15, category: 'education' },    { name: 'Phone', price: 500, category: 'electronics' },    { name: 'Pen', price: 2, category: 'office' },    { name: 'eboard', price: 400, category: 'education' }  ];  const expensiveProducts = products.filter(product => product.price > 100);  console.log(expensiveProducts);  // خروجی: [{ name: 'Laptop', price: 1000, category: 'electronics' },  //         { name: 'Phone', price: 500, category: 'electronics' },  //         {name: 'eboard', price: 400, category: 'education'}]  const electronicProducts = products.filter(product => product.category === 'education');  console.log(electronicProducts);  // خروجی: [{name: 'Book', price: 15, category: 'education'},  //         {name: 'eboard', price: 400, category: 'education'}] |

[filterObject](webdesign/js/Object/js/filterObject.js)

مثال 4: استفاده از index در فیلتر کردن

|  |
| --- |
| const numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];  // انتخاب عناصر با ایندکس زوج  const evenIndexNumbers = numbers.filter((number, index) => index % 2 === 0);  console.log(evenIndexNumbers); // خروجی: [0, 2, 4, 6, 8] |

[filterIndex](webdesign/js/Object/js/filterIndex.js)

مثال 5: حذف مقادیر falsy از آرایه

|  |
| --- |
| const mixedValues = [0, 1, false, 2, '', 3, null, undefined, 4, NaN];  const truthyValues = mixedValues.filter(value => value);  // یا به صورت explicit: mixedValues.filter(value => Boolean(value))  console.log(truthyValues); // خروجی: [1, 2, 3, 4] |

[filterfalsy](webdesign/js/Object/js/filterfalsy.js)

### تفاوت filter با سایر متدهای آرایه

مقایسه متدهای رایج آرایه در جاوااسکریپت

|  |  |  |  |
| --- | --- | --- | --- |
| متد | هدف اصلی | مقدار بازگشتی | تغییر آرایه اصلی |
| Filter() | فیلتر کردن عناصر بر اساس شرط | آرایه جدید با عناصر فیلتر شده | خیر |
| Map() | تبدیل هر عنصر به یک عنصر جدید | آرایه جدید با نتایج تابع callback | خیر |
| forEach() | اجرای تابع برای هر عنصر | توسط کاربر تعیین میشود | خیر |
| Find() | یافتن اولین عنصر که شرط را دارد | عنصر یافت شده یا undefined | خیر |
| [Reduce()](#_آموزش_استفاده_از_1) | کاهش آرایه به یک مقدار | مقدار نهایی حاصل از کاهش | خیر |

مثال مقایسه‌ای:

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5];  // filter: فقط اعداد زوج  const filtered = numbers.filter(n => n % 2 === 0); // [2, 4]  // map: تبدیل هر عدد به مربع آن  const mapped = numbers.map(n => n n); // [1, 4, 9, 16, 25]  // find: یافتن اولین عدد بزرگتر از 3  const found = numbers.find(n => n > 3); // 4  // reduce: جمع همه اعداد  const reduced = numbers.reduce((sum, n) => sum + n, 0); // 15 |

[filterVs](webdesign/js/Object/js/filterVs.js)

### بهترین practices و نکات مهم

نکات کلیدی برای استفاده بهینه از filter():

1. عدم تغییر آرایه اصلی: همیشه به یاد داشته باشید که filter() یک آرایه جدید برمی‌گرداند و آرایه اصلی را تغییر نمی‌دهد.

2. استفاده از توابع arrow: برای کدهای مختصر و خوانا، از توابع arrow استفاده کنید.

3. بهینه‌سازی عملکرد: برای آرایه‌های بسیار بزرگ، از پیچیدگی محاسباتی تابع callback آگاه باشید.

4. قابلیت ترکیب: filter() را می‌توان با سایر متدهای آرایه مانند map()و reduce() ترکیب کرد.

5. خوانایی کد: برای شرایط پیچیده، از نام‌های توصیفی برای متغیرها استفاده کنید.

مواردی که باید اجتناب کنید:

1. استفاده از filter() برای side effects: از filter() فقط برای فیلتر کردن استفاده کنید، نه برای اثرات جانبی.

عدم استفاده از filter() برای Side Effects با مثال

مشکل استفاده نادرست از filter() برای Effects جانبی

متد `filter()` در جاوااسکریپت تنها برای فیلتر کردن عناصر آرایه بر اساس یک شرط طراحی شده است. استفاده از آن برای انجام عملیات جانبی (side effects) یک الگوی ضد طراحی (antipattern) محسوب می‌شود.

#### مقایسه استفاده صحیح و نادرست

مثال نادرست استفاده از filter() برای side effects:

|  |
| --- |
| const users = [  { id: 1, name: 'Alice', active: true },  { id: 2, name: 'Bob', active: false },  { id: 3, name: 'Charlie', active: true }  ];  // استفاده نادرست: انجام عملیات جانبی در filter  const activeUsers = users.filter(user => {  console.log(`Checking user: ${user.name}`); // side effect  sendLogToServer(user.id); // side effect خطرناک  return user.active;  });  function sendLogToServer(userId) {  // شبیه‌سازی ارسال به سرور  console.log(`Logging user ${userId} to server...`);  }  // خروجی:  // Checking user: Alice  // Logging user 1 to server...  // Checking user: Bob  // Logging user 2 to server...  // Checking user: Charlie  // Logging user 3 to server...  console.log(activeUsers);  // [{ id: 1, name: 'Alice', active: true }, { id: 3, name: 'Charlie', active: true }] |

[filterSideEffects](webdesign/js/Object/js/filterSideEffects.js)

مثال صحیح جداسازی concerns

|  |
| --- |
| const users = [  { id: 1, name: 'Alice', active: true },  { id: 2, name: 'Bob', active: false },  { id: 3, name: 'Charlie', active: true }  ];  const sendLogToServer = (userId)=>{  console.log(`this user id is ${userId}`);  }  // ابتدا عملیات جانبی را انجام دهید (اگر لازم است)  users.forEach(user => {  console.log(`Checking user: ${user.name}`); // ✅ مناسب برای side effects  sendLogToServer(user.id); // ✅ در جای مناسب  });  // سپس فیلتر کردن را انجام دهید  const activeUsers = users.filter(user => user.active); // ✅ فقط فیلتر کردن  console.log(activeUsers);  // [{ id: 1, name: 'Alice', active: true }, { id: 3, name: 'Charlie', active: true }] |

[filterConcerns](webdesign/js/Object/js/filterConcerns.js)

خطرات استفاده نادرست از filter() برای side effects

1. اجرای مضاعف عملیات

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5];  let count = 0;  // خطرناک: ممکن است چندین بار اجرا شود  const evenNumbers = numbers.filter(num => {  count++; // side effect  console.log(`Processing number: ${num}`);  return num % 2 === 0;  });  console.log(`Total operations: ${count}`); // 5 - اما ممکن است غیرمنتظره باشد |

[filterOver](webdesign/js/Object/js/filterOver.js)

1. مشکلات عملکردی در زنجیره متدها

|  |
| --- |
| // تشخیص منبع side effect سخت می‌شود  const processedData = sourceData.filter(item => {  updateCache(item); // side effect پنهان  validateItem(item); // side effect دیگر  return item.isValid;  }); |

[filterChain](webdesign/js/Object/js/filterChain.js)

1. مشکلات در دیباگ کردن

|  |
| --- |
| const users = [  { id: 1, name: 'Alice', premium: true },  { id: 2, name: 'Bob', premium: false },  { id: 3, name: 'Charlie', premium: true }  ];  const premiumUsers = [];  const logs = [];  // انجام همزمان فیلتر و logging با کنترل کامل  for (const user of users) {  console.log(`Processing user: ${user.name}`);  logs.push(`Processed user ${user.id}`);  if (user.premium) {  premiumUsers.push(user);  }  }  console.log(premiumUsers);  console.log(logs); |

[filterDebug](webdesign/js/Object/js/filterDebug.js)

2. بازگرداندن مقادیر غیرboolean: تابع callback باید مقدار boolean بازگرداند.

3. نادیده گرفتن مقدار بازگشتی: اگر به آرایه جدید نیاز ندارید، شاید forEach() یا for...of مناسب‌تر باشد.

### خلاصه مزایای استفاده از filter():

خوانایی کد: کد شما را تمیزتر و خواناتر می‌کند

عدم تغییرپذیری: آرایه اصلی را تغییر نمی‌دهد

قابلیت ترکیب: با سایر متدهای آرایه ترکیب می‌شود

پشتیبانی گسترده: در تمام محیط‌های اجرایی جاوااسکریپت پشتیبانی می‌شود

یادگیری و تسلط بر متد `filter()` و سایر متدهای آرایه مانند `map()` و `reduce` برای تبدیل شدن به یک توسعه‌دهنده ماهر جاوااسکریپت ضروری است . با تمرین و به کارگیری این مفاهیم در پروژه‌های واقعی، می‌توانید کدهای کارآمدتر و تمیزتری بنویسید.

## آموزش استفاده از reduce

آموزش کامل متد `reduce` در جاوااسکریپت

متد `reduce` یکی از قدرتمندترین متدهای آرایه در جاوااسکریپت است که برای محاسبه یک مقدار بر اساس تمام عناصر آرایه استفاده می‌شود.

ساختار پایه

|  |
| --- |
| array.reduce(callback(accumulator, currentValue, index, array), initialValue) |

accumulator: مقدار انباشته شده از فراخوانی‌های قبلی

currentValue: عنصر فعلی آرایه که در حال پردازش است

index: اندیس عنصر فعلی (اختیاری)

array: آرایه اصلی (اختیاری)

initialValue: مقدار اولیه برای accumulator (اختیاری)

### مثال‌های کاربردی

1. جمع کردن اعداد آرایه

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5];  // بدون مقدار اولیه  const sum = numbers.reduce((acc, curr) => acc + curr);  console.log(sum); // 15  // با مقدار اولیه  const sumWithInitial = numbers.reduce((acc, curr) => acc + curr, 10);  console.log(sumWithInitial); // 25 |

[reduceSum](webdesign/js/Object/js/reduceSum.js)

1. محاسبه حاصل ضرب

|  |
| --- |
| const numbers = [2, 3, 4];  const product = numbers.reduce((acc, curr) => acc curr, 1);  console.log(product); // 24 |

[reduceMultiply](webdesign/js/Object/js/reduceMultiply.js)

1. پیدا کردن بیشترین مقدار

|  |
| --- |
| const numbers = [12, 45, 7, 32, 89, 3];  const max = numbers.reduce((acc, curr) => Math.max(acc, curr));  console.log(max); // 89 |

[reduceMax](webdesign/js/Object/js/reduceMax.js)

1. تبدیل آرایه به شیء

|  |
| --- |
| const fruits = ['apple', 'banana', 'orange'];  const fruitObject = fruits.reduce((acc, curr, index) => {  acc[index] = curr;  return acc;  }, {});  console.log(fruitObject);  // {0: "apple", 1: "banana", 2: "orange"} |

[reduceArrayToObject](webdesign/js/Object/js/reduceArrayToObject.js)

1. شمارش تکرار عناصر

|  |
| --- |
| const words = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple'];  const wordCount = words.reduce((acc, curr) => {  acc[curr] = (acc[curr] || 0) + 1;  return acc;  }, {});  console.log(wordCount);  // {apple: 3, banana: 2, orange: 1} |

[reduceCounter](webdesign/js/Object/js/reduceCounter.js)

1. مسطح کردن آرایه‌های تو در تو

|  |
| --- |
| const nestedArray = [[1, 2], [3, 4], [5, 6]];  const flatArray = nestedArray.reduce((acc, curr) => acc.concat(curr), []);  console.log(flatArray); // [1, 2, 3, 4, 5, 6] |

[reduceFlat](webdesign/js/Object/js/reduceFlat.js)

1. گروه‌بندی اشیا بر اساس ویژگی

|  |
| --- |
| const people = [  { name: 'Alice', age: 25 },  { name: 'Bob', age: 30 },  { name: 'Charlie', age: 25 },  { name: 'David', age: 30 }  ];  const groupedByAge = people.reduce((acc, curr) => {  const age = curr.age;  if (!acc[age]) {  acc[age] = [];  }  acc[age].push(curr);  return acc;  }, {});  console.log(groupedByAge);  /  {  25: [{ name: 'Alice', age: 25 }, { name: 'Charlie', age: 25 }],  30: [{ name: 'Bob', age: 30 }, { name: 'David', age: 30 }]  }  / |

[reduceCat](webdesign/js/Object/js/reduceCat.js)

### تمرین‌ :

#### تمرین 1: محاسبه میانگین

|  |
| --- |
| const scores = [85, 90, 78, 92, 88];  const average = scores.reduce((acc, curr, index, array) => {  acc += curr;  if (index === array.length-1) {  return acc / array.length;  }  return acc;  }, 0);  console.log(average); // 86.6 |

[reduceAverage](webdesign/js/Object/js/reduceAverage.js)

تمرین 2: حذف مقادیر تکراری

|  |
| --- |
| const numbers = [1, 2, 2, 3, 4, 4, 5, 5, 5];  const uniqueNumbers = numbers.reduce((acc, curr) => {  if (!acc.includes(curr)) {  acc.push(curr);  }  return acc;  }, []);  console.log(uniqueNumbers); // [1, 2, 3, 4, 5] |

[reduceRepeat](webdesign/js/Object/js/reduceRepeat.js)

تمرین 3: ترکیب چندین آرایه

|  |
| --- |
| const arrays = [[1, 2], [3, 4], [5, 6]];  const combined = arrays.reduce((acc, curr) => [...acc, ...curr], []);  console.log(combined); // [1, 2, 3, 4, 5, 6] |

[reduceArrayConcat](webdesign/js/Object/js/reduceArrayConcat.js)

تمرین 4: تبدیل آرایه به رشته با فرمت خاص

|  |
| --- |
| // reduceArrayToString  const items = ['apple', 'banana', 'orange'];  const sentence = items.reduce((acc, curr, index) => {  if (index === 0) {  return curr;  } else if (index === items.length-1) {  return `${acc} and ${curr}`;  } else {  return `${acc}, ${curr}`;  }  }, '');  console.log(sentence); // "apple, banana and orange" |

[reduceArrayToString](webdesign/js/Object/js/reduceArrayToString.js)

نکات مهم

1. همیشه مقدار اولیه (`initialValue`) ارائه دهید تا از خطاها جلوگیری شود.

2. بدون مقدار اولیه، reduce از اولین عنصر آرایه به عنوان accumulator اولیه استفاده می‌کند.

3. reduce می‌تواند برای پیاده سازی بسیاری از متدهای دیگر مانند map، filter و find استفاده شود.

جمع‌بندی :

متد `reduce` یک ابزار بسیار قدرتمند و انعطاف‌پذیر برای پردازش آرایه‌ها در جاوااسکریپت است. با تمرین و استفاده از مثال‌های فوق، می‌توانید به خوبی با این متد آشنا شده و از آن در پروژه‌های خود استفاده کنید.

## آموزش استفاده از Rest parameters آموزش استفاده از Spread Operator:

عملگر Spread و Rest در جاوااسکریپت

عملگر `...` که در پارامترهای تابع به صورت `(…fns)` استفاده می‌شود، در جاوااسکریپت به دو صورت مختلف استفاده می‌شود:

### 1. Rest Parameters (پارامترهای rest)

وقتی `...` در تعریف پارامترهای تابع استفاده می‌شود، به آن Rest Parameters می‌گویند. این عملگر تمام آرگومان‌های باقی‌مانده را در یک آرایه جمع می‌کند.

مثال‌هایی از Rest Parameters:

|  |
| --- |
| // جمع کردن تمام آرگومان‌ها  function sum(...numbers) {  return numbers.reduce((total, num) => total + num, 0);  }  console.log(sum(1, 2, 3, 4, 5)); // 15  // ترکیب با پارامترهای عادی  function greet(greeting, ...names) {  return names.map(name => `${greeting}, ${name}!`);  }  console.log(greet('Hello', 'Alice', 'Bob', 'Charlie'));  // ['Hello, Alice!', 'Hello, Bob!', 'Hello, Charlie!']  // در arrow functions  const multiply = (multiplier, ...numbers) => {  return numbers.map(n => n multiplier);  };  console.log(multiply(2, 1, 2, 3)); // [2, 4, 6] |

[RestParameters](webdesign/js/Object/js/RestParameters.js)

### 2. Spread Operator (عملگر spread)

وقتی `...` در فراخوانی تابع یا با آرایه‌ها و آبجکت‌ها استفاده می‌شود، به آن Spread Operator می‌گویند. این عملگر عناصر یک آرایه یا ویژگی‌های یک آبجکت را گسترش می‌دهد.

مثال‌هایی از Spread Operator:

با آرایه‌ها:

|  |
| --- |
| // ترکیب آرایه‌ها  const arr1 = [1, 2, 3];  const arr2 = [4, 5, 6];  const combined = [...arr1, ...arr2];  console.log(combined); // [1, 2, 3, 4, 5, 6]  // کپی آرایه  const original = [1, 2, 3];  const copy = [...original];  console.log(copy); // [1, 2, 3]  // استفاده در فراخوانی تابع  const numbers = [1, 2, 3, 4, 5];  console.log(Math.max(...numbers)); // 5 |

[SpreadOperator](webdesign/js/Object/js/SpreadOperator.js)

با آبجکت‌ها:

|  |
| --- |
| // ترکیب آبجکت‌ها  const obj1 = { a: 1, b: 2 };  const obj2 = { c: 3, d: 4 };  const merged = { ...obj1, ...obj2 };  console.log(merged); // { a: 1, b: 2, c: 3, d: 4 }  // کپی آبجکت  const originalObj = { x: 10, y: 20 };  const copyObj = { ...originalObj };  console.log(copyObj); // { x: 10, y: 20 }  // overwrite properties  const base = { a: 1, b: 2, c: 3 };  const updated = { ...base, b: 20, d: 4 };  console.log(updated); // { a: 1, b: 20, c: 3, d: 4 } |

[SpreadOperatorObject](webdesign/js/Object/js/SpreadOperatorObject.js)

تفاوت‌های کلیدی:

|  |  |  |
| --- | --- | --- |
| ویژگی | Spread Operator | Rest Parameters |
| موقعیت استفاده | در تعریف پارامترهای تابع | در فراخوانی تابع، آرایه‌ها و آبجکت‌ها |
| هدف | جمع‌آوری چندین مقدار در یک آرایه | | گسترش یک آرایه/آبجکت به عناصر/ویژگی‌های جداگانه |
| نوع داده | همیشه یک آرایه ایجاد می‌کند | روی آرایه‌ها، آبجکت‌ها و iterableها کار می‌کند |

مثال‌های ترکیبی:

|  |
| --- |
| // ترکیب Rest و Spread  function processData(first, second, ...rest) {  console.log(`First: ${first}`);  console.log(`Second: ${second}`);  console.log(`Rest: ${rest.join(', ')}`);    // استفاده از Spread برای پردازش بیشتر  const allData = [first, second, ...rest];  return allData.map(item => item 2);  }  const result = processData(1, 2, 3, 4, 5);  console.log(result); // [2, 4, 6, 8, 10]  // در Destructuring  const [first, ...others] = [1, 2, 3, 4, 5];  console.log(first); // 1  console.log(others); // [2, 3, 4, 5]  const { a, ...rest } = { a: 1, b: 2, c: 3, d: 4 };  console.log(a); // 1  console.log(rest); // { b: 2, c: 3, d: 4 } |

[SpreadOperatorRestParameters](webdesign/js/Object/js/SpreadOperatorRestParameters.js)

کاربردهای پیشرفته:

|  |
| --- |
| // کلون عمیق ساده (فقط برای آبجکت‌های ساده)  const original = { a: 1, b: { c: 2 } };  const clone = { ...original, b: { ...original.b } };  // ادغام آبجکت‌ها با ویژگی‌های تکراری  const defaults = { theme: 'light', fontSize: 16 };  const userPreferences = { theme: 'dark' };  const finalConfig = { ...defaults, ...userPreferences };  console.log(finalConfig); // { theme: 'dark', fontSize: 16 }  // ایجاد آرایه بدون مقادیر تکراری  const withDuplicates = [1, 2, 2, 3, 4, 4, 5];  const unique = [...new Set(withDuplicates)];  console.log(unique); // [1, 2, 3, 4, 5] |

[SpreadOperatorPros](webdesign/js/Object/js/SpreadOperatorPros.js)

این عملگر یکی از قدرتمندترین ویژگی‌های ES6+ است که کد را خوانا‌تر و مختصرتر می‌کند.

### آموزش کار با تابع join()

تابع join() از توابع کار با آرایه‌هاست. این متد آرایه را به یک رشته متنی تبدیل میکند و برای اتصال عناصر از کاراکتر خاصی میتواند استفاده کند.

مثال:

|  |
| --- |
| const elements = ["Fire", "Air", "Water"];  console.log(elements.join());  // Expected output: "Fire,Air,Water"  console.log(elements.join(""));  // Expected output: "FireAirWater"  console.log(elements.join("-"));  // Expected output: "Fire-Air-Water" |

[join](webdesign/js/Object/js/join.js)

## آموزش Destructuring Assignment

معرفی Destructuring

Destructuring Assignment یک قابلیت قدرتمند در ES6 است که به شما امکان می‌دهد مقادیر را از آرایه‌ها یا ویژگی‌ها را از آبجکت‌ها استخراج کرده و در متغیرهای جداگانه قرار دهید.

Destructuring آرایه‌ها (Array Destructuring):

مبانی پایه:

|  |
| --- |
| // آرایه سنتی  const numbers = [1, 2, 3];  const a = numbers[0];  const b = numbers[1];  const c = numbers[2];  // با Destructuring  const [x, y, z] = [1, 2, 3];  console.log(x); // 1  console.log(y); // 2  console.log(z); // 3 |

[Destructuring](webdesign/js/Object/js/Destructuring.js)

مقداردهی پیشفرض

|  |
| --- |
| const [a = 10, b = 20, c = 30] = [1, 2];  console.log(a); // 1  console.log(b); // 2  console.log(c); // 30 (پیش‌فرض) |

[DestructuringPreValue](webdesign/js/Object/js/DestructuringPreValue.js)

رد کردن عناصر

|  |
| --- |
| const [first, , third] = [1, 2, 3, 4];  console.log(first); // 1  console.log(third); // 3 |

[DestructuringOmitted](webdesign/js/Object/js/DestructuringOmitted.js)

جمع‌آوری باقی‌مانده‌ها

|  |
| --- |
| const [first, second, ...rest] = [1, 2, 3, 4, 5];  console.log(first); // 1  console.log(second); // 2  console.log(rest); // [3, 4, 5] |

[DestructuringSum](webdesign/js/Object/js/DestructuringSum.js)

تعویض مقادیر (Swap)

|  |
| --- |
| let a = 1;  let b = 2;  // روش سنتی  let temp = a;  a = b;  b = temp;  // با Destructuring  [a, b] = [b, a];  console.log(a); // 2  console.log(b); // 1 |

[DestructuringSwap](webdesign/js/Object/js/DestructuringSwap.js)

### Destructuring آبجکت‌ها (Object Destructuring)

|  |
| --- |
| const person = {  name: 'علی',  age: 30,  city: 'تهران'  };  // روش سنتی  const name = person.name;  const age = person.age;  // با Destructuring  const { name, age, city } = person;  console.log(name); // 'علی'  console.log(age); // 30  console.log(city); // 'تهران' |

[DestructuringObject](webdesign/js/Object/js/DestructuringObject.js)

نام‌گذاری مجدد متغیرها

|  |
| --- |
| const person = {  firstName: 'فاطمه',  lastName: 'احمدی'  };  const { firstName: fname, lastName: lname } = person;  console.log(fname); // 'فاطمه'  console.log(lname); // 'احمدی' |

[DestructuringRename](webdesign/js/Object/js/DestructuringRename.js)

مقادیر پیش‌فرض

|  |
| --- |
| const { name = 'ناشناس', age = 0, city = 'نامشخص' } = { name: 'محمد' };  console.log(name); // 'محمد'  console.log(age); // 0 (پیش‌فرض)  console.log(city); // 'نامشخص' (پیش‌فرض) |

[DestructuringPreValue2](webdesign/js/Object/js/DestructuringPreValue2.js)

ترکیب نام‌گذاری مجدد و پیش‌فرض

|  |
| --- |
| const user = {  username: 'user123'  };  const {  username: uname = 'مهمان',  role: userRole = 'user'  } = user;  console.log(uname); // 'user123'  console.log(userRole); // 'user' (پیش‌فرض) |

[DestructuringPreValue3](webdesign/js/Object/js/DestructuringPreValue3.js)

Destructuring تو در تو

|  |
| --- |
| const company = {  name: 'فناوری اطلاعات',  address: {  city: 'مشهد',  street: 'آزادی',  zip: '12345'  }  };  const {  name,  address: {  city,  street: mainStreet,  zip: postalCode = '00000'  }  } = company;  console.log(name); // 'فناوری اطلاعات'  console.log(city); // 'مشهد'  console.log(mainStreet); // 'آزادی'  console.log(postalCode); // '12345' |

[DestructuringComplex](webdesign/js/Object/js/DestructuringComplex.js)

### کاربردهای عملی Destructuring

#### پارامترهای تابع

|  |
| --- |
| // روش سنتی  function printPerson(person) {  console.log(person.name, person.age);  }  // با Destructuring  function printPerson({ name, age, city = 'نامشخص' }) {  console.log(`${name} - ${age} سال - ${city}`);  }  printPerson({ name: 'علی', age: 30 }); // علی - 30 سال - نامشخص |

[DestructuringParamiter](webdesign/js/Object/js/DestructuringParamiter.js)

1. پاسخ‌های API

|  |
| --- |
| // شبیه‌سازی پاسخ API  const apiResponse = {  status: 'success',  data: {  users: [  { id: 1, name: 'User1' },  { id: 2, name: 'User2' }  ],  total: 2  }  };  // Destructuring برای دسترسی به داده‌ها  const {  status,  data: {  users,  total  }  } = apiResponse;  console.log(status); // 'success'  console.log(users); // [{...}, {...}]  console.log(total); // 2 |

[DestructuringAPi](webdesign/js/Object/js/DestructuringAPi.js)

1. پیکربندی و تنظیمات

|  |
| --- |
| const defaultConfig = {  theme: 'light',  language: 'fa',  notifications: true  };  function initApp(userConfig = {}) {  const {  theme = 'light',  language = 'fa',  notifications = true,  fontSize = 16  } = userConfig;  console.log(`پیکربندی: ${theme}, ${language}, ${fontSize}`);  }  initApp({ theme: 'dark', fontSize: 18 }); |

[DestructuringSetting](webdesign/js/Object/js/DestructuringSetting.js)

Destructuring با آرایه و آبجکت ترکیبی

|  |
| --- |
| const users = [  { name: 'علی', age: 30 },  { name: 'فاطمه', age: 25 },  { name: 'محمد', age: 35 }  ];  // Destructuring در map  const names = users.map(({ name }) => name);  console.log(names); // ['علی', 'فاطمه', 'محمد']  // Destructuring در for-of  for (const { name, age } of users) {  console.log(`${name}: ${age} سال`);  } |

[DestructuringArrayObj](webdesign/js/Object/js/DestructuringArrayObj.js)

آبجکت‌های پیچیده

|  |
| --- |
| const project = {  title: 'وبسایت شرکت',  team: [  { name: 'علی', role: 'developer' },  { name: 'فاطمه', role: 'designer' }  ],  deadline: '2024-12-31'  };  const {  title,  team: [firstMember, secondMember],  deadline: dueDate  } = project;  console.log(title); // 'وبسایت شرکت'  console.log(firstMember); // { name: 'علی', role: 'developer' }  console.log(dueDate); // '2024-12-31' |

[DestructuringComplex2](webdesign/js/Object/js/DestructuringComplex2.js)

نکات مهم و خطاهای رایج

Destructuring آرایه‌های کوتاه:

|  |
| --- |
| const [a, b, c] = [1, 2];  console.log(a); // 1  console.log(b); // 2  console.log(c); // undefined  // با پیش‌فرض  const [x, y, z = 0] = [1, 2];  console.log(z); // 0 |

[DestructuringArray3](webdesign/js/Object/js/DestructuringArray3.js)

نام ویژگی‌های آبجکت

|  |
| --- |
| const obj = { 'first-name': 'علی', 'last-name': 'محمدی' };  // باید از نام واقعی ویژگی استفاده کرد  const { 'first-name': firstName, 'last-name': lastName } = obj;  console.log(firstName); // 'علی' |

[DestructuringObject3](webdesign/js/Object/js/DestructuringObject3.js)

تمرین‌های عملی

تمرین 1: تبدیل مختصات

|  |
| --- |
| const point = [10, 20, 30];  // تبدیل به آبجکت با Destructuring  const [x, y, z] = point;  const pointObj = { x, y, z };  console.log(pointObj); // { x: 10, y: 20, z: 30 } |

[DestructuringRemap](webdesign/js/Object/js/DestructuringRemap.js)

تمرین 2: پردازش کاربر

|  |
| --- |
| const userData = {      personalInfo: {          name: 'زهرا',          family: 'رضایی',          birthDate: '1995-05-15'      },      contact: {          email: 'zahra@example.com',          phone: '09123456789'      }  };  // Destructuring برای استخراج اطلاعات  const {      personalInfo: { name, family },      contact: { email, phone }  } = userData;  console.log(`نام: ${name} ${family}`);  console.log(`تماس: ${email} - ${phone}`); |

[DestructuringUsers](webdesign/js/Object/js/DestructuringUsers.js)

تمرین 3: مدیریت خطا

|  |
| --- |
| function safeDestructuring(obj, path, defaultValue) {      const keys = path.split('.');      let current = obj;        for (const key of keys) {          if (current === null || current === undefined) {              return defaultValue;          }          current = current[key];      }        return current !== undefined ? current : defaultValue;  }  // استفاده  const data = { user: { profile: { name: 'علی' } } };  const userName = safeDestructuring(data, 'user.profile.name', 'ناشناس');  console.log(userName); // 'علی' |

[DestructuringErrorHandling](webdesign/js/Object/js/DestructuringErrorHandling.js)

جمع‌بندی

✅ مزایای Destructuring:

- کد تمیزتر و خواناتر

- کاهش خطوط کد تکراری

- مدیریت آسان پارامترهای اختیاری

- استخراج سریع داده‌های پیچیده

موارد استفاده:

- پارامترهای تابع

- پاسخ‌های API

- پیکربندی‌ها

- کار با آرایه‌ها و آبجکت‌ها

- ایونت هندلینگ

نکات امنیتی:

- همیشه از پیش‌فرض استفاده کنید

- در برابر null/undefined محافظت کنید

- برای ویژگی‌های تودرتو احتیاط کنید

Destructuring یکی از مفیدترین ویژگی‌های مدرن جاوااسکریپت است که به شما کمک می‌کد کدهای تمیزتر و قابل نگهداری‌تری بنویسید!