### آموزش وب دزاین

فهرست

[آموزش وب دزاین 1](#_Toc209640780)

[مفهوم Frontend (فرانت‌اند) 11](#_Toc209640781)

[تفاوت Frontend با Backend (یک مقایسه کلیدی) 12](#_Toc209640782)

[مهارت‌های مورد نیاز یک توسعه‌دهنده فرانت‌اند 13](#_Toc209640783)

[بررسی مفهوم Backend 14](#_Toc209640784)

[Backend (بک‌اند) چیست؟ 14](#_Toc209640785)

[اجزای اصلی توسعه بک‌اند (Core Components) 14](#_Toc209640786)

[مهارت‌های مورد نیاز یک توسعه‌دهنده بک‌اند 16](#_Toc209640787)

[معرفی HTML: 16](#_Toc209640788)

[HTML چیست؟ 17](#_Toc209640789)

[HTML چگونه کار می‌کند؟ 17](#_Toc209640790)

[مثال یک تگ ساده: 17](#_Toc209640791)

[توضیح بخش‌های مختلف: 18](#_Toc209640792)

[معرفی مهم‌ترین تگ‌های HTML 18](#_Toc209640793)

[ویژگی‌های تگ‌ها (Attributes) 20](#_Toc209640794)

[مثال‌های رایج: 20](#_Toc209640795)

[معرفی Css: 21](#_Toc209640796)

[CSS چیست؟ 21](#_Toc209640797)

[CSS چگونه کار می‌کند؟ 21](#_Toc209640798)

[روش‌های اتصال CSS به HTML 22](#_Toc209640799)

[2. Box Model (مدل جعبه‌ای) 24](#_Toc209640800)

[ویژگی‌های پرکاربرد CSS 25](#_Toc209640801)

[معرفی JavaScript 26](#_Toc209640802)

[ویژگی‌های کلیدی JavaScript: 26](#_Toc209640803)

[بررسی مفهوم SEO 26](#_Toc209640804)

[تعریف SEO 26](#_Toc209640805)

[سه رکن اصلی SEO 27](#_Toc209640806)

[مراحل اجرای پروژه SEO 28](#_Toc209640807)

[بررسی مفهوم Progressive Enhancement بهبود تدریجی 30](#_Toc209640808)

[نکات کلیدی برای یادآوری 34](#_Toc209640809)

[جمع‌بندی برای مبتدیان 34](#_Toc209640810)

[Progressive Enhancement یعنی: 34](#_Toc209640811)

[بررسی مفهوم Responsive Web Design 34](#_Toc209640812)

[بررسی مفهوم Internet 40](#_Toc209640813)

[بررسی مفهوم Protocol 48](#_Toc209640814)

[تعریف ساده پروتکل 48](#_Toc209640815)

[چرا پروتکل لازم است؟ 49](#_Toc209640816)

[1. زبان مشترک 49](#_Toc209640817)

[2. قابل پیش‌بینی بودن 49](#_Toc209640818)

[3. خطایابی آسان 49](#_Toc209640819)

[انواع پروتکل‌های اینترنتی 49](#_Toc209640820)

[1. TCP/IP - پادشاه پروتکل‌ها 49](#_Toc209640821)

[2. HTTP/HTTPS - پروتکل وب 50](#_Toc209640822)

[3. FTP - پروتکل انتقال فایل 50](#_Toc209640823)

[4. SMTP/POP3 - پروتکل ایمیل 50](#_Toc209640824)

[ویژگی‌های یک پروتکل خوب 51](#_Toc209640825)

[1. قابل اعتماد (Reliable) 51](#_Toc209640826)

[2. کارا (Efficient) 51](#_Toc209640827)

[3. امن (Secure) 52](#_Toc209640828)

[4. قابل توسعه (Extensible) 52](#_Toc209640829)

[پروتکل‌های معروف در زندگی روزمره 52](#_Toc209640830)

[1. بلوتوث - پروتکل بی‌سیم کوتاه‌برد 52](#_Toc209640831)

[2. WiFi - پروتکل شبکه بی‌سیم 52](#_Toc209640832)

[3. NFC - پروتکل ارتباط نزدیک 52](#_Toc209640833)

[مثال ساده: پروتکل دست دادن 53](#_Toc209640834)

[پروتکل‌های امنیتی 53](#_Toc209640835)

[1. SSL/TLS - پروتکل امن 53](#_Toc209640836)

[2. VPN - پروتکل شبکه خصوصی 53](#_Toc209640837)

[3. SSH - پروتکل امن برای سرور 54](#_Toc209640838)

[آینده پروتکل‌ها 54](#_Toc209640839)

[پروتکل چیست: 54](#_Toc209640840)

[پروتکل چیست نیست: 55](#_Toc209640841)

[جمع‌بندی نهایی 55](#_Toc209640842)

[سه نکته طلایی: 55](#_Toc209640843)

[بررسی مفهوم WEB 55](#_Toc209640844)

[تعریف ساده وب 55](#_Toc209640845)

[تفاوت اینترنت و وب 56](#_Toc209640846)

[تاریخچه وب به زبان ساده 57](#_Toc209640847)

[انواع وب‌سایت‌ها 58](#_Toc209640848)

[1. وب‌سایت استاتیک (ایستا) 58](#_Toc209640849)

[2. وب‌سایت داینامیک (پویا) 58](#_Toc209640850)

[3. وب اپلیکیشن‌ها 59](#_Toc209640851)

[چگونه وب کار می‌کند؟ - یک داستان 59](#_Toc209640852)

[اجزای اصلی تجربه وب 60](#_Toc209640853)

[تکنولوژی‌های سمت سرور vs سمت کلاینت 60](#_Toc209640854)

[طراحی وب مدرن - اصول مهم 61](#_Toc209640855)

[امنیت در وب 62](#_Toc209640856)

[راه‌های محافظت: 62](#_Toc209640857)

[تاثیر وب بر زندگی ما 62](#_Toc209640858)

[آینده وب 63](#_Toc209640859)

[مفاهیم کلیدی برای یادگیری 63](#_Toc209640860)

[جمع‌بندی نهایی 64](#_Toc209640861)

[سه نکته طلایی: 64](#_Toc209640862)

[شروع یادگیری 64](#_Toc209640863)

[بررسی مفهوم HTTP 65](#_Toc209640864)

[متدهای اصلی HTTP (Verbs) 66](#_Toc209640865)

[تفاوت HTTP و HTTPS 68](#_Toc209640866)

[حالت‌های ارتباط HTTP 69](#_Toc209640867)

[نسخه‌های HTTP 69](#_Toc209640868)

[هدرهای مهم HTTP 71](#_Toc209640869)

[مشاهده HTTP در عمل 71](#_Toc209640870)

[ویژگی‌های پیشرفته HTTP 72](#_Toc209640871)

[امنیت در HTTP 72](#_Toc209640872)

[نکات کلیدی برای مبتدیان 73](#_Toc209640873)

[HTTP چیست: 73](#_Toc209640874)

[HTTP چیست نیست: 73](#_Toc209640875)

[جمع‌بندی نهایی 73](#_Toc209640876)

[شروع یادگیری عملی 74](#_Toc209640877)

[بررسی مفهوم Server 74](#_Toc209640878)

[انواع سرورها 75](#_Toc209640879)

[سرور چگونه کار می‌کند؟ - یک داستان 76](#_Toc209640880)

[اجزای اصلی یک سرور 76](#_Toc209640881)

[سرور در مقابل کلاینت 77](#_Toc209640882)

[کلاینت (خدمت‌گیرنده): 77](#_Toc209640883)

[نمونه‌های واقعی سرور 79](#_Toc209640884)

[امنیت سرور 79](#_Toc209640885)

[ویژگی‌های یک سرور خوب 80](#_Toc209640886)

[سرورها در زندگی روزمره 80](#_Toc209640887)

[هزینه‌های سرور 82](#_Toc209640888)

[چندپارادایمی: 84](#_Toc209640889)

[1- برنامه نویسی تابعی یا رویه‌ایی: 84](#_Toc209640890)

[2- برنامه نویسی شیئ گرا (ObjectOriented Programming) 84](#_Toc209640891)

[3- برنامه‌نویسی تابعی (Functional Programming) 86](#_Toc209640892)

[آموزش استفاده از Map در جاوا اسکریپت 89](#_Toc209640893)

[1- مقدمه‌ای بر Map 89](#_Toc209640894)

[نکات پیشرفته 99](#_Toc209640895)

[تمرینات اضافه 101](#_Toc209640896)

[آموزش کار با createDocumentFragment() 108](#_Toc209640897)

[آموزش استفاده از filter 108](#_Toc209640898)

[آموزش جامع استفاده از متد filter() در جاوااسکریپت 108](#_Toc209640899)

[1- مقدمه‌ای بر متد filter() 108](#_Toc209640900)

[2- سینتکس و پارامترها 109](#_Toc209640901)

[3- موارد استفاده و مثال‌های کاربردی 110](#_Toc209640902)

[4- تفاوت filter با سایر متدهای آرایه 111](#_Toc209640903)

[5- بهترین practices و نکات مهم 112](#_Toc209640904)

[خلاصه مزایای استفاده از filter(): 115](#_Toc209640905)

[آموزش استفاده از reduce 116](#_Toc209640906)

[مثال‌های کاربردی 116](#_Toc209640907)

[تمرین‌ : 118](#_Toc209640908)

[آموزش استفاده از Rest parameters آموزش استفاده از Spread Operator: 120](#_Toc209640909)

[1. Rest Parameters (پارامترهای rest) 120](#_Toc209640910)

[2. Spread Operator (عملگر spread) 121](#_Toc209640911)

[آموزش کار با تابع join() 124](#_Toc209640912)

[آموزش Destructuring Assignment 124](#_Toc209640913)

[Destructuring آبجکت‌ها (Object Destructuring) 126](#_Toc209640914)

[کاربردهای عملی Destructuring 128](#_Toc209640915)

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

# بررسی مفهوم IP

IP چیست؟ (یک تشبیه ساده)

فرض کنید شما می‌خواهید یک نامه پستی برای دوست خود بفرستید. برای اینکه نامه به دست او برسد، باید آدرس دقیق خانه او را روی پاکت بنویسید. در دنیای شبکه‌های کامپیوتری و اینترنت، آدرس IP دقیقاً همان نقش "آدرس" را بازی می‌کند.

IP مخفف Internet Protocol (پروتکل اینترنت) است. هر دستگاه متصل به یک شبکه (مثل اینترنت یا حتی شبکه داخلی خانه شما) یک آدرس IP منحصربه‌فرد دریافت می‌کند تا بتواند با دیگر دستگاه‌ها ارتباط برقرار کند.

دستگاه شما (کامپیوتر، موبایل، تلویزیون هوشمند و...) = خانه شما

آدرس IP = شماره پلاک و نام خیابان خانه شما

آدرس IP چگونه است؟ (دو نسخه اصلی)

دو استاندارد اصلی برای آدرس‌های IP وجود دارد:

1. IPv4 (نسخه ۴)

این نسخه کلاسیک و رایج‌ترین فرمت است. آدرس‌های IPv4 از چهار عدد که با نقطه از هم جدا می‌شوند تشکیل شده‌اند. هر عدد می‌تواند بین ۰ تا ۲۵۵ باشد.

مثال: `192.168.1.1` یا `8.8.8.8` (آدرس سرورهای DNS گوگل)

با گسترش اینترنت، تعداد آدرس‌های منحصربه‌فرد IPv4 (حدود ۴.۳ میلیارد) در حال اتمام است.

2. IPv6 (نسخه ۶)

برای حل مشکل کمبود آدرس، نسخه جدیدی به نام IPv6 معرفی شد. این آدرس‌ها بسیار طولانی‌تر و از اعداد هگزادسیمال (۰-۹ و A-F) و دونقطه تشکیل شده‌اند.

\* مثال: `2001:0db8:85a3:0000:0000:8a2e:0370:7334`

این نسخه تعداد بسیار بیشتری آدرس (به میزان غیرقابل تصوری) در اختیار می‌گذارد.

انواع آدرس IP از نظر دسترسی

آدرس‌های IP به دو دسته اصلی تقسیم می‌شوند:

1. IP عمومی (Public IP)

این آدرس، شناسه اصلی شبکه شما در اینترنت است. وقتی به اینترنت متصل می‌شوید، شرکت ارائه‌دهنده اینترنت (ISP) مانند ایرانسل، شاتل یا پارس‌آنلاین یک IP عمومی به روتر (مودم) خانه شما اختصاص می‌دهد. تمام دستگاه‌های داخل خانه شما با یک IP عمومی به دنیای خارج دیده می‌شوند.

مثال: وقتی در گوگل "what is my ip" را جستجو می‌کنید، IP عمومی خود را می‌بینید.

2. IP خصوصی (Private IP)

در داخل شبکه محلی شما (مثلاً شبکه Wi-Fi خانه)، روتر به هر دستگاه یک IP خصوصی اختصاص می‌دهد. این آدرس‌ها فقط در داخل شبکه معتبر هستند و از بیرون قابل دسترسی نیستند. این کار شبیه دادن شماره اتاق‌های مختلف به ساکنان یک ساختمان بزرگ است.

مثال: آدرس‌هایی مانند `192.168.1.10` برای لپ‌تاپ شما و `192.168.1.11` برای موبایل شما.

نحوه ارتباط: وقتی لپ‌تاپ شما (`192.168.1.10`) یک درخواست به سایت گوگل می‌فرستد، روتر این درخواست را دریافت کرده، با استفاده از IP عمومی خود آن را به اینترنت ارسال می‌کند و وقتی پاسخ آمد، آن را به لپ‌تاپ شما برمی‌گرداند. این فرآیند ترجمه آدرس شبکه (NAT) نام دارد.

IP استاتیک (ثابت) در مقابل IP داینامیک (پویا)

IP داینامیک: بیشتر کاربران عادی اینترنت این نوع IP را دارند. IP عمومی شما پس از مدتی (مثلاً هر ۲۴ ساعت یا با هر بار restart کردن مودم) توسط ISP تغییر می‌کند. این روش برای ISPها مقرون‌به‌صرفه‌تر است.

IP استاتیک: این آدرس ثابت است و تغییر نمی‌کند. معمولاً سرورها، وب‌سایت‌ها یا دستگاه‌هایی که نیاز است همیشه از بیرون در دسترس باشند (مثل دوربین‌های مداربسته تحت شبکه) به IP استاتیک نیاز دارند و معمولاً هزینه اضافه دارند.

پروتکل اینترنت (Internet Protocol) چه می‌کند؟

همانطور که از نامش پیداست، IP یک پروتکل یا قوانین ارتباطی است. وظیفه اصلی آن مسیریابی (Routing) بسته‌های داده به مقصد صحیح است. IP مسئولیت تحویل بسته‌ها را بر عهده دارد، اما تضمینی برای رسیدن بدون خطای آن‌ها نمی‌دهد. تضمین تحویل مطمئن بر عهده پروتکل‌های دیگر مانند TCP است (که معمولاً together با آن به صورت TCP/IP شناخته می‌شوند).

چگونه IP خود را پیدا کنیم؟

IP عمومی: کافیست در مرورگر خود به سایتی مانند [whatismyipaddress.com](https://whatismyipaddress.com/) بروید.

IP خصوصی:

ویندوز: در Command Prompt دستور `ipconfig` را تایپ کنید.

مک: در Terminal دستور `ifconfig` را تایپ کنید.

اندروید و iOS: به تنظیمات WiFi رفته و روی شبکه متصل شده کلیک کنید تا جزئیات را ببینید.

جمع‌بندی نکات کلیدی

|  |  |
| --- | --- |
| ویژگی | توضیح |
| کاربرد اصلی | شناسایی و مسیریابی دستگاه‌ها در یک شبکه |
| فرمت‌های رایج | IPv4 (مثل ۱۹۲.۱۶۸.۱.۱) و IPv6 (مثل ۲۰۰۱:db8::۸a۲e) |
| انواع دسترسی | عمومی: برای اینترنت | خصوصی: برای شبکه داخلی |
| انواع تخصیص | داینامیک: تغییرکننده | استاتیک: ثابت |

# بررسی مفهوم Domain

دامنه چیست؟ (تشبیه ساده)

فرض کنید ما در دنیای اینترنت زندگی می‌کنیم. هر وب‌سایت یک خانه است که روی یک سرور (یک کامپیوتر قدرتمند همیشه روشن) قرار دارد. هر خانه یک آدرس IP منحصربه‌فرد دارد (مثل `۱۹۲.۰.۲.۴۴`).

مشکل اینجاست که به خاطر سپردن این اعداد برای انسان‌ها بسیار سخت است. آیا می‌توانید آدرس IP تمام وب‌سایت‌هایی که روزانه بازمی‌کنید را به خاطر بسپارید؟

اینجاست که نام دامنه وارد می‌شود. دامنه مانند یک پلاک خوانا و به یاد ماندنی برای آن آدرس IP است.

آدرس IP = `۱۹۲.۰.۲.۴۴` (مثل مختصات جغرافیایی یک خانه)

نام دامنه = `google.com` (مثل یک آدرس پستی ساده و قابل خواندن)

پس دامنه در واقع نامی است که شما در نوار آدرس مرورگر خود تایپ می‌کنید تا به یک وب‌سایت دسترسی پیدا کنید.

دامنه چگونه کار می‌کند؟ (سیستم DNS)

وقتی شما `www.google.com` را در مرورگر خود وارد می‌کنید، چه اتفاقی می‌افتد؟

1. مرورگر شما به سرورهای خاصی به نام DNS (سیستم نام دامنه) مراجعه می‌کند. DNS مانند دفترچه تلفن بسیار بزرگ اینترنت است.

2. مرورگر می‌پرسد: "آدرس IP مربوط به `google.com` چیست؟"

3. سرور DNS پاسخ می‌دهد: "آدرس IP آن `۱۷۲.۲۱۷.۱۶.۲۳۸` است."

4. حالا مرورگر با استفاده از آن آدرس IP، می‌تواند به سرور گوگل متصل شده و صفحه وب را برای شما بارگذاری کند.

این کل فرآیند در کسری از ثانیه اتفاق می‌افتد.

اجزای تشکیل‌دهنده یک نام دامنه

یک نام دامنه کامل از چند بخش تشکیل شده است. بیایید `https://www.example.ir` را بررسی کنیم:

1. زیردامنه (Subdomain):

بخش سمت چپ دامنه اصلی.

رایج‌ترین آن `www` است، اما می‌تواند چیزهای دیگری مثل `mail.google.com` یا `blog.example.com` باشد.

اختیاری است (امروزه بسیاری از سایت‌ها بدون `www` نیز کار می‌کنند).

2. نام دامنه (Second-Level Domain - SLD):

بخش اصلی و منحصربه‌فرد نامی که شما انتخاب می‌کنید.

در مثال ما `example` نام دامنه است.

این بخش هویت و برند وب‌سایت شما را می‌سازد.

3. پسوند دامنه (Top-Level Domain - TLD):

بخش سمت راست نقطه.

در مثال ما `.ir` پسوند است.

این پسوندها دسته‌بندی خاصی را نشان می‌دهند.

انواع پسوندهای دامنه (TLD)

پسوندها به دسته‌های مختلفی تقسیم می‌شوند:

دامنه‌های ملی (ccTLD - Country Code TLD):

مخصوص کشورها هستند.

مثال: `.ir` (ایران)، `.de` (آلمان)، `.us` (آمریکا)، `.tr` (ترکیه).

دامنه‌های عمومی (gTLD - Generic TLD):

معروف‌ترین و پرکاربردترین پسوندها.

مثال: `.com` (Commercial - تجاری)، `.org` (Organization - سازمان‌ها)، `.net` (Network - شبکه‌ها)، `.info` (Information - اطلاعات).

دامنه‌های جدید (New gTLD):

پسوندهای جدیدی که برای تنوع بیشتر ایجاد شده‌اند.

مثال: `.shop`، `.app`، `.blog`، `.tech`، `.bank`.

چرا به دامنه نیاز داریم؟ (اهداف و مزایا)

1. قابل حفظ کردن: به خاطر سپردن `google.com` بسیار ساده‌تر از `۱۷۲.۲۱۷.۱۶.۲۳۸` است.

2. حرفه‌ای بودن: داشتن یک دامنه اختصاصی (مثل `brandname.com`) بسیار معتبرتر از استفاده از یک آدرس رایگان (مثل `brandname.freehost.com`) است.

3. برندسازی: دامنه بخشی از هویت و برند کسب‌وکار شما در اینترنت می‌شود.

4. ثبات: حتی اگر سرور یا آدرس IP وب‌سایت شما تغییر کند، می‌توانید دامنه خود را به آدرس جدید اشاره کنید و کاربران همچنان با همان نام قدیمی به سایت شما دسترسی خواهند داشت.

چگونه یک دامنه ثبت کنیم؟

دامنه‌ها را نمی‌توان خرید، بلکه آن‌ها را برای یک دوره مشخص (معمولاً ۱ تا ۱۰ سال) ثبت (Register) می‌کنید. برای این کار باید به شرکت‌های ثبت‌کننده دامنه (Domain Registrar) مانند شرکت‌های داخلی (مثلاً `پارس‌پک`، `ایران‌سرور`) یا بین‌المللی (مثلاً `GoDaddy`، `Namecheap`) مراجعه کنید.

تفاوت دامنه و هاست (مهم!)

این دو مفهوم اغلب با هم اشتباه گرفته می‌شوند، اما کاملاً متفاوت هستند:

|  |  |  |
| --- | --- | --- |
| ویژگی | دامنه (Domain) | هاست (Hosting) |
| تعریف | آدرس وب‌سایت شما (مثل آدرس پستی) | فضای ذخیره‌سازی وب‌سایت شما (مثل زمین و ساختمان) |
| کاربرد | کاربران با تایپ آن به سایت شما می‌رسند | فایل‌ها، عکس‌ها و کدهای سایت شما روی آن قرار می‌گیرد |
| مثال | example.com | یک سرور فیزیکی در یک مرکز داده |

برای راه‌اندازی یک وب‌سایت، شما به هر دو نیاز دارید: هم یک دامنه و هم یک سرویس هاستینگ.

جمع‌بندی نکات کلیدی

|  |  |
| --- | --- |
| مفهوم | توضیح خلاصه |
| کاربرد اصلی | تبدیل آدرس‌های عددی سخت (IP) به نام‌های خوانا و به یاد ماندنی |
| اجزای اصلی | زیردامنه (مثل www) + نام (مثل example) + پسوند (مثل .com) |
| سیستم پشتیبان | DNS که مانند دفترچه تلفن، دامنه را به IP تبدیل می‌کند |
| ثبت دامنه | اجاره نام برای یک دوره مشخص (مثلاً ۱ سال) از طریق Registrarها |

# بررسی مفهوم DNS

DNS چیست؟ (تشبیه ساده)

فرض کنید شما می‌خواهید با یک دوست تماس بگیرید. شما نام او را در حافظه خود دارید (مثلاً "احمد")، اما تلفن برای برقراری ارتباط نیاز به شماره تلفن او دارد. در این حالت، شما از دفترچه تلفن استفاده می‌کنید تا نام "احمد" را به شماره تلفن او تبدیل کنید.

DNS (Domain Name System) دقیقاً همان نقش دفترچه تلفن جهانی اینترنت را بازی می‌کند.

نام دامنه (مثل `google.com`) = نام شخص (مثل "احمد")

آدرس IP (مثل `142.251.16.100`) = شماره تلفن

DNS = دفترچه تلفن

کار اصلی DNS ترجمه نام‌های قابل خواندن توسط انسان (دامنه) به آدرس‌های عددی ماشین‌ها (IP) است.

مشکل بزرگی که DNS حل می‌کند

بدون DNS، شما برای بازدید از هر وب‌سایتی مجبور بودید آدرس IP عددی و خشک آن را به خاطر بسپارید.

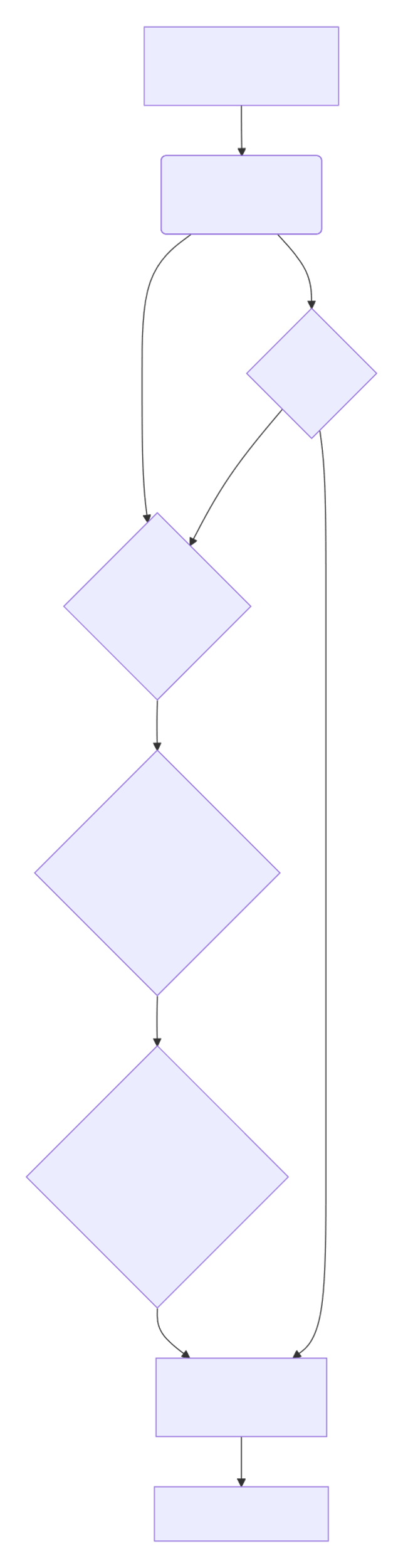
با DNS: شما `digikala.com` را تایپ می‌کنید.

بدون DNS: شما مجبور بودید چیزی شبیه `۵.۱۴۴.۱۳۱.۲۳` را به خاطر بسپارید و وارد کنید.

DNS زندگی را در اینترنت برای انسان‌ها ممکن ساخته است.

DNS چگونه کار می‌کند؟ (یک جستجوی معمولی)

وقتی شما `www.youtube.com` را در مرورگر خود وارد کرده و Enter می‌زنید، چه اتفاقی پشت صحنه می‌افتد؟ این فرآیند که جستجوی DNS (DNS Lookup) نام دارد، در چندین مرحله و در کسری از ثانیه انجام می‌شود. نمودار زیر این جریان را به وضوح نشان می‌دهد:



Website Loaded

IP Address : 142.251.16.100

Authoritative Server

IP for www.youtube.com?

yes

TLD Server

Where is youtube.com?

Root Server

Where is .com?

no

Cache Hit?

Cache Check

Resolver  
ISP/Recursive DNS

User enters  
www.youtube.com

در نمودار بالا دیدیم که چگونه چهار سرویس اصلی DNS برای تبدیل نام دامنه به آدرس IP با یکدیگر همکاری می‌کنند. حالا اجازه دهید نقش هرکدام را دقیق‌تر بررسی کنیم:

سرورهای درگیر در فرآیند DNS

1. DNS Recursor (Resolve کننده بازگشتی):

نقش: همان "تحقیق‌کننده" است. این سرور معمولاً توسط ISP (شرکت اینترنتی شما) ارائه می‌شود. درخواست شما را دریافت می‌کند و برای پیدا کردن پاسخ، به دیگر سرورها "سرک" می‌کشد.

مثال: سرور DNS شرکت‌هایی مانند ایرانسل، شاتل یا پارس‌آنلاین.

2. Root Name Server (سرور ریشه):

نقش: مانند فهرست بخش‌های مختلف دفترچه تلفن عمل می‌کند. این سرور آدرس سرورهای مربوط به پسوندها (TLD) مانند `.com`، `.org`، `.ir` را می‌داند.

کاربرد: به Resolver می‌گوید: "برای پیدا کردن `youtube.com`، باید به سرورهای مخصوص پسوند `.com` مراجعه کنی."

3. TLD Name Server (سرور پسوند دامنه):

نقش: اطلاعات مربوط به یک پسوند خاص را نگهداری می‌کند.

کاربرد: سرور مخصوص `.com` به Resolver می‌گوید: "برای اطلاعات دامنه `youtube.com`، باید به سرورهای معتبر (Authoritative) که مسئول این دامنه هستند، مراجعه کنی. آدرس آن سرورها این است..."

4. Authoritative Name Server (سرور معتبر):

نقش: این سرور منبع اصلی حقیقت برای یک دامنه خاص است. اطلاعات دقیق دامنه (مانند آدرس IP اصلی) در اینجا نگهداری می‌شود.

مالکیت: این سرورها توسط صاحب دامنه یا شرکت هاستینگ آن مدیریت می‌شوند.

نتیجه نهایی: این سرور به Resolver پاسخ نهایی را می‌دهد: "آدرس IP `www.youtube.com` برابر است با `۱۴۲.۲۵۱.۱۶.۱۰۰`."

کش DNS (DNS Cache): میان‌بری برای سرعت

اگر نمودار را دوباره نگاه کنید، یک مسیر جایگزین از `Cache Hit?` می‌بینید. اینجاست که کش وارد عمل می‌شود.

کش چیست؟ یک حافظه موقت سریع.

کاربرد: پس از اولین بار پیدا کردن آدرس IP یک دامنه، Resolver نتیجه را برای مدت مشخصی (TTL) در کش خود ذخیره می‌کند.

فایده: اگر شما یا کاربر دیگری در مدت زمان معینی دوباره به همان دامنه مراجعه کند، Resolver به جای طی کردن کل مسیر طولانی، بلافاصله پاسخ را از کش خود برمی‌گرداند. این کار سرعت را به شدت افزایش می‌دهد و از ترافیک شبکه می‌کاهد.

کش در سیستم عامل (ویندوز، مک) و حتی مرورگر شما نیز وجود دارد.

انواع رکوردهای DNS (Records)

سرورهای معتبر، اطلاعات یک دامنه را در قالب رکوردهای مختلفی ذخیره می‌کنند. هر رکورد یک وظیفه خاص دارد:

A Record: اصلی‌ترین رکورد که دامنه را به آدرس IPv4 تبدیل می‌کند.

AAAA Record: دامنه را به آدرس IPv6 تبدیل می‌کند.

CNAME Record: یک دامنه را به نام دامنه دیگری "هدایت می‌کند" (مثلاً `digikala.com` را به `www.digikala.com`).

MX Record: مشخص می‌کند ایمیل‌های مربوط به دامنه شما به کدام سرور ایمیل ارسال شوند.

TXT Record: برای ذخیره اطلاعات متنی مانند تأیید مالکیت دامنه یا تنظیمات امنیتی (SPF, DKIM) استفاده می‌شود.

اهمیت DNS: چرا DNS حیاتی است؟

1. کاربرپسند کردن اینترنت: پایه و اساس استفاده آسان از اینترنت است.

2. قابلیت اطمینان و افزونگی: سیستم DNS به گونه‌ای طراحی شده که بسیار مقاوم است و اگر یک سرور از کار بیفتد، سرورهای دیگر پاسخگو هستند.

3. انعطاف‌پذیری: به صاحبان وب‌سایت اجازه می‌دهد بدون تغییر دامنه، سرور یا هاستینگ خود را عوض کنند (فقط رکوردهای DNS را به روز می‌کنند).

4. توزیع بار (Load Balancing): DNS می‌تواند ترافیک را بین چندین سرور توزیع کند تا از overload شدن یک سرور جلوگیری شود.

جمع‌بندی نکات کلیدی

|  |  |
| --- | --- |
| مفهوم | توضیح خلاصه |
| کاربرد اصلی | تبدیل نام دامنه (مثل google.com) به آدرس IP (مثل ۱۴۲.۲۵۱.۱۶.۱۰۰) |
| تشبیه | دفترچه تلفن اینترنت |
| فرآیند اصلی | جستجوی DNS: درخواست کاربر -> Resolver -> Root -> TLD -> Authoritative -> IP |
| عنصر سرعت‌دهنده | کش (Cache): ذخیره موقت پاسخ‌ها برای دسترسی سریع‌تر در آینده |
| اجزای کلیدی | Recursive Resolver, Root Server, TLD Server, Authoritative Server |

# بررسی مفهوم Interanet

اینترانت چیست؟ (تشبیه ساده)

فرض کنید یک شرکت یا سازمان، یک شهر کوچک خصوصی است که دیوارهایی دور آن کشیده شده است. فقط ساکنان این شهر (کارکنان آن سازمان) اجازه ورود دارند و می‌توانند از امکانات داخل شهر مانند کتابخانه، تابلوهای اعلانات، ادارات و سیستم پستی داخلی استفاده کنند.

اینترانت دقیقاً همان شهر خصوصی است.

\* اینترانت = یک شبکه داخلی خصوصی که از فناوری‌های مشابه اینترنت (مانند وب‌سرورها و مرورگرها) استفاده می‌کند، اما فقط برای اعضای یک سازمان قابل دسترسی است.

تعریف فنی

اینترانت (Intranet) یک شبکه رایانه‌ای خصوصی است که از پروتکل‌های اینترنتی (مثل TCP/IP و HTTP) برای اشتراک‌گذاری اطلاعات، ابزارهای collaboration و سیستم‌های عملیاتی در داخل یک سازمان استفاده می‌کند. دسترسی به آن معمولاً به کارمندان سازمان محدود می‌شود و از طریق فایروال از دسترسی عمومی محافظت می‌گردد.

هدف اصلی اینترانت چیست؟

هدف اصلی، ایجاد یک پلتفرم مرکزی و امن برای موارد زیر است:

اشتراک‌گذاری اطلاعات: انتشار اخبار شرکت، دستورالعلها، راهنماهای کاری، فرم‌ها.

همکاری (Collaboration): فضای کاری مشترک، انجمن‌های گفتگو، ویکی‌های داخلی.

اتوماسیون فرآیندها: سیستم درخواست مرخصی، ثبت سفارش خرید، مدیریت پروژه.

دسترسی به برنامه‌های کاربردی: برنامه‌های نرم‌افزاری داخلی شرکت که فقط از طریق شبکه داخلی در دسترس هستند.

ویژگی‌های کلیدی اینترانت

1. دسترسی محدود و امن: مهمترین ویژگی. فقط افراد مجاز (کارکنان) می‌توانند به آن وارد شوند. این دسترسی اغلب با نام کاربری و رمز عبور کنترل می‌شود.

2. محرمانگی: اطلاعات حساس شرکت (مثل استراتژی‌ها، مالی، منابع انسانی) در محیطی امن قرار می‌گیرد.

3. متمرکز بودن: همه چیز در یک جای واحد قرار دارد و employees برای پیدا کردن اطلاعات لازم به چندین منبع مراجعه نمی‌کنند.

4. استفاده از مرورگر وب: کاربران با مرورگر آشنا (مثل Chrome, Firefox) به آن دسترسی دارند، بنابراین نیاز به آموزش کمی وجود دارد.

اینترانت چگونه کار می‌کند؟

اینترانت از همان زیرساخت یک شبکه محلی (LAN) استفاده می‌کند:

1. یک یا چند وب‌سرور در داخل شبکه سازمان نصب می‌شوند که صفحات اینترانت را میزبانی می‌کنند.

2. یک فایروال قوی بین شبکه داخلی و اینترنت عمومی قرار می‌گیرد تا از ورود غیرمجاز جلوگیری کند.

3. کارمندان از طریق کامپیوترهای خود در شبکه سازمان، با تایپ یک آدرس (مثل `http://intranet`) به آن متصل می‌شوند.

4. اگر کارمندی از راه دور بخواهد به اینترانت دسترسی داشته باشد، معمولاً باید از یک VPN (شبکه خصوصی مجازی) استفاده کند تا connection او امن شده و به عنوان بخشی از شبکه داخلی شناسایی شود.

مثال‌های عینی از اینترانت

سایت داخلی یک شرکت بزرگ: صفحه‌ای که با ورود به شرکت، همه کارمندان می‌بینند و شامل اخبار شرکت، لیست تلفن داخلی، لینک به نرم‌افزارهای داخلی و فرم درخواست مرخصی است.

سیستم مدیریت آموزش (LMS) داخلی: پورتالی که کارمندان برای گذراندن دوره‌های آموزشی اجباری از آن استفاده می‌کنند.

ویکی (Wiki) داخلی شرکت: دانشنامه‌ای که در آن روش‌های انجام کارها، troubleshooting و بهترین تجربیات به اشتراک گذاشته می‌شود.

مقایسه اینترانت با مفاهیم مشابه (مهم!)

برای درک بهتر، اینترانت را با اینترنت و اکسترانت مقایسه می‌کنیم:

|  |  |  |  |
| --- | --- | --- | --- |
| ویژگی | اینترنت (Internet) | اینترانت (Intranet) | اکسترانت (Extranet) |
| محدوده دسترسی | عمومی و جهانی - همه می‌توانند به بخش‌های عمومی دسترسی داشته باشند. | داخلی و خصوصی - فقط اعضای یک سازمان. | | نیمه خصوصی - اعضای سازمان + شرکای خارجی محدود (مانند تامین کنندگان، مشتریان خاص). |
| هدف | اشتراک‌گذاری اطلاعات در سطح جهانی. | اشتراک‌گذاری اطلاعات و collaboration در داخل سازمان. | اشتراک‌گذاری اطلاعات selected با شرکای خارج از سازمان. |
| میزان امنیت | امنیت نسبتاً پایین (فضای عمومی). | امنیت بسیار بالا (پشت فایروال). | امنیت بالا، با کنترل دسترسی دقیق برای کاربران خارجی. |
| مثال | `google.com`, `digikala.com` | آدرس داخلی شرکت برای اخبار و منابع انسانی | پورتالی که یک شرکت برای تامین کنندگان خود ایجاد کرده تا موجودی انبار را ببینند. |

مزایا و معایب اینترانت

مزایا:

افزایش بهره‌وری: دسترسی سریع و متمرکز به اطلاعات.

کاهش هزینه‌ها: کاهش مصرف کاغذ، بهبود ارتباطات.

بهبود فرهنگ سازمانی: ایجاد حس تعلق و یکپارچگی.

امنیت اطلاعات: حفاظت از داده‌های حساس شرکت.

معایب:

هزینه راه‌اندازی و نگهداری: نیاز به سرور، نرم‌افزار و پشتیبانی فنی دارد.

مقاومت در برابر تغییر: ممکن است کارمندان به استفاده از آن عادت نکنند.

نیاز به بروزرسانی: اطلاعات باید همیشه به روز باشند تا مفید واقع شوند.

جمع‌بندی نکات کلیدی

|  |  |
| --- | --- |
| مفهوم | توضیح خلاصه |
| تعریف | شبکه خصوصی یک سازمان که از تکنولوژی وب استفاده می‌کند |
| تشبیه | شهر خصوصی یا شبکه داخلی یک شرکت |
| کاربران | فقط کارکنان و اعضای سازمان |
| هدف اصلی | اشتراک‌گذاری اطلاعات و بهبود همکاری داخلی به صورت امن و متمرکز |
| تفاوت با اینترنت | اینترنت عمومی است، اینترانت خصوصی است. |

# بررسی مفهوم Extranet

فرض کنید یک شرکت، یک محله امن و خصوصی دارد (اینترانت آن). حالا این شرکت می‌خواهد برای شرکای تجاری خاصی (مثل تأمین‌کنندگان یا مشتریان ویژه) دسترسی کنترل‌شده‌ای به بخشی از این محله بدهد. اما نمی‌خواهد این افراد به تمام ساختمان‌ها دسترسی داشته باشند.

راه‌حل چیست؟ یک دریچه امن یا اتاق مهمان مخصوص در حصار این محله ایجاد می‌کند که دسترسی به چند بخش مشخص را فراهم می‌کند.

اکسترانت دقیقاً همان اتاق مهمان یا دریچه امن است.

اکسترانت = یک شبکه خصوصی گسترده که دسترسی محدود و امنی را به بخشی از اطلاعات یا سرویس‌های یک سازمان برای کاربران خارجی (مانند شرکا، تأمین‌کنندگان، مشتریان منتخب) فراهم می‌کند.

تعریف فنی

اکسترانت (Extranet) بخشی از شبکه خصوصی یک سازمان (اینترانت) است که به صورت کنترل‌شده و از طریق اینترنت، در اختیار کاربران مجاز خارج از سازمان قرار می‌گیرد. دسترسی معمولاً با نام کاربری، رمز عبور و سایر روش‌های احراز هویت شدیداً کنترل می‌شود.

هدف اصلی اکسترانت چیست؟

هدف اصلی، ایجاد همکاری امن و کارآمد با entities خارجی بدون به خطر انداختن امنیت شبکه داخلی است.

کاربردهای رایج اکسترانت

1. برای تأمین‌کنندگان (Suppliers):

مشاهده سطح موجودی انبار شرکت

دیدن برنامه‌ریزی تولید برای هماهنگی بهتر تحویل کالا

ثبت قیمت‌ها و شرکت در مناقصات داخلی

2. برای مشتریان بزرگ (Key Customers):

پیگیری سفارشات (مثلاً دیدن وضعیت تولید و حمل سفارش خود)

دسترسی به پایگاه دانش فنی و پشتیبانی ویژه

ثبت مستقیم سفارشات جدید

3. برای شعب یا نمایندگی‌های دیگر:

اشتراک‌گذاری اطلاعات فروش، بازاریابی و منابع آموزشی بین شعب مختلف یک شرکت بین‌المللی

4. برای پیمانکاران و شرکای پروژه:

دسترسی به اسناد، نقشه‌ها و زمان‌بندی‌های یک پروژه خاص برای هماهنگی بهتر

ویژگی‌های کلیدی اکسترانت

1. دسترسی بسیار کنترل‌شده: کاربران خارجی فقط به بخش‌های خاصی که برایشان تعریف شده دسترسی دارند ("نیاز به دانستن").

2. امنیت فوق‌العاده بالا: از فایروال‌های پیشرفته، رمزگذاری (Encryption) و روش‌های قوی احراز هویت (مانند کلیدهای امنیتی) استفاده می‌کند.

3. مبتنی بر اینترنت: دسترسی از طریق اینترنت عمومی انجام می‌شود، اما connection کاملاً امن است.

4. هدف تجاری و همکاری: ماهیت آن همکاری با entities خارجی برای بهبود زنجیره تأمین و روابط تجاری است.

اکسترانت چگونه کار می‌کند؟

1. یک سازمان بخشی از سرویس‌های اینترانت خود را (مثلاً یک برنامه یا پایگاه داده) برای دسترسی از بیرون آماده می‌کند.

2. یک درگاه امن (Secure Gateway) روی فایروال ایجاد می‌کند.

3. برای هر کاربر خارجی یک حساب کاربری (Account) با سطوح دسترسی مشخص ایجاد می‌شود.

4. کاربران خارجی از طریق اینترنت، به آدرس خاص اکسترانت متصل می‌شوند.

5. آن‌ها باید هویت خود را احراز کنند (با نام کاربری و رمز عبور قوی یا روش‌های دو مرحله‌ای).

6. پس از ورود، فقط به منابعی که مجاز هستند دسترسی پیدا می‌کنند.

یک مثال عینی

شرکت خودروسازی "الف" را در نظر بگیرید:

اینترانت شرکت الف: حاوی اطلاعات محرمانه مانند استراتژی‌های آینده، مالیات، حقوق کارکنان و طرح‌های جدید است. فقط کارکنان به آن دسترسی دارند.

اکسترانت شرکت الف: این شرکت یک اکسترانت برای تأمین‌کنندگان قطعات خود ایجاد کرده است.

تأمین‌کننده با ورود به اکسترانت می‌تواند:

موجودی انبار قطعات شرکت الف را ببیند تا بداند چه زمانی باید قطعات جدید ارسال کند.

برنامه تولید هفته آینده شرکت الف را مشاهده کند.

صورتحساب‌های خود را بارگذاری و وضعیت پرداخت آن‌ها را پیگیری کند.

اما تأمین‌کننده هرگز نمی‌تواند به اطلاعات حقوق کارکنان یا طرح‌های محرمانه جدید دسترسی پیدا کند.

مزایا و معایب اکسترانت

مزایا:

بهبود کارایی زنجیره تأمین: هماهنگی سریع‌تر و دقیق‌تر با تأمین‌کنندگان و مشتریان.

کاهش هزینه‌ها: کاهش خطاهای ارتباطی، کاغذبازی و زمان پردازش سفارشات.

تقویت روابط با شرکا: شفافیت و اعتماد را افزایش می‌دهد.

امنیت بالا: اطلاعات حساس شرکت در عین به اشتراک‌گذاری محدود، محافظت می‌شوند.

معایب:

هزینه پیاده‌سازی و نگهداری بالا: نیاز به زیرساخت امنیتی قوی دارد.

پیچیدگی مدیریت: مدیریت کاربران خارجی و سطوح دسترسی آن‌ها چالش‌برانگیز است.

ریسک امنیتی بالقوه: با وجود امنیت بالا، باز کردن یک دریچه به بیرون همیشه یک ریسک محسوب می‌شود و نیاز به نظارت دائمی دارد.

# بررسی موتورهای رندرگیری مرورگرها

موتور رندرگیری مرورگر، هسته اصلی هر مرورگر وب است که مسئول تفسیر کدهای HTML و CSS و نمایش بصری صفحات وب برای شماست. در جدول زیر میتوانید مهمترین موتورهای رندر و مرورگرهای مرتبط با آنها را ببینید.

|  |  |  |
| --- | --- | --- |
| موتور رندر | مرورگرهای شاخص | توضیحات |
| Blink | گوگل کروم، مایکروسافت اج (نسخه جدید)، اپرا، ادج | موتور اصلی و پیشرو در دنیای مرورگرها که توسط گوگل توسعه یافته است. |
| WebKit | سافاری (اپل) | موتور اصلی مورد استفاده در محصولات اپل. همه مرورگرهای iOS موظفند از این موتور استفاده کنند. |
| Gecko | موزیلا فایرفاکس | موتور متن باز توسعه یافته توسط موزیلا که به عنوان جایگزینی مستقل برای Blink شناخته می‌شود. |
| Trident | اینترنت اکسپلورر (IE) | موتور قدیمی مایکروسافت که امروزه منسوخ شده است. |
| EdgeHTML | مایکروسافت اج (نسخه قدیمی) | جانشین Trident که بعداً جای خود را به Blink داد. |

چرا شناخت موتور رندر مهم است؟

تفاوت در موتورهای رندر می‌تواند دلیل اصلی تفاوت در نمایش یک صفحه وب در مرورگرهای مختلف باشد. این موضوع به ویژه برای طراحان وب بسیار حیاتی است، زیرا آن‌ها باید اطمینان حاصل کنند که وب‌سایت طراحی شده، در تمامی موتورهای رندر اصلی به درستی و یکسان نمایش داده می‌شود.

# بررسی URL

URL چیست؟ (تشبیه ساده)

فرض کنید شما در یک شهر بزرگ زندگی می‌کنید. هر خانه در این شهر یک آدرس پستی منحصربه‌فرد دارد که شامل نام خیابان، شماره پلاک، نام شهر و کدپستی است. با این آدرس، پستچی می‌داند دقیقاً باید نامه را به کجا برساند.

URL دقیقاً همان "آدرس پستی" برای اطلاعات در دنیای وب است.

\* URL = نشانی یکتای منابع در وب که به مرورگر می‌گوید دقیقاً چه چیزی را و از کجا باید بگیرد.

URL مخفف Uniform Resource Locator به معنای "نشانی‌یاب یکنواخت منبع" است.

URL چگونه است؟ (اجزای تشکیل‌دهنده)

یک URL کامل از چند بخش تشکیل شده است. بیایید آدرس زیر را بررسی کنیم:

`https://www.example.com:8080/path/to/myfile.html?key1=value1&key2=value2section1`

1. پروتکل (Protocol):

مثال: `https://`

نقش: قوانین ارتباطی بین مرورگر و سرور را مشخص می‌کند. مشخص می‌کند که داده‌ها چگونه منتقل شوند.

رایج‌ترین پروتکل: `HTTP` (ناآمنه) و `HTTPS` (امن و رمزگذاری شده). پروتکل‌های دیگر عبارتند از `ftp://` (برای انتقال فایل) و `mailto://` (برای ایمیل).

2. نام میزبان (Hostname):

مثال: `www.example.com`

نقش: آدرس سروری که منبع (مثل وب‌سایت) روی آن قرار دارد را مشخص می‌کند. این بخش معمولاً یک نام دامنه است.

3. پورت (Port):

مثال: `:8080`

نقش: یک "درگاه" خاص روی سرور را برای ارتباط مشخص می‌کند. مانند شماره داخلی یک ساختمان بزرگ.

پورت‌های پیش‌فرض: اگر پورت مشخص نشود، مرورگر به طور خودکار از پورت استاندارد استفاده می‌کند (پورت ۸۰ برای HTTP و پورت ۴۴۳ برای HTTPS). بنابراین معمولاً در URLها دیده نمی‌شوند.

4. مسیر (Path):

مثال: `/path/to/myfile.html`

نقش: محل دقیق منبع درخواستی (مثل یک صفحه HTML، یک عکس یا یک فایل) را در ساختار پوشه‌های سرور مشخص می‌کند. مانند مسیر یک فایل در کامپیوتر شما.

5. پارامترهای Query (کوئری):

مثال: `?key1=value1&key2=value2`

نقش: اطلاعات اضافی را به سرور ارسال می‌کند. این بخش معمولاً پس از علامت سؤال (`?`) می‌آید و شامل جفت‌های `key=value` است که با `&` از هم جدا می‌شوند.

کاربرد: برای جستجو در سایت، فیلتر کردن نتایج یا ارسال داده‌های فرم استفاده می‌شود. مثلاً وقتی در یوتیوب جستجو می‌کنید، URL به شکل `youtube.com/results?search\_query=query` درمی‌آید.

6. لنگر (Fragment یا Anchor):

مثال: `section1`

نقش: به مرورگر دستور می‌دهد که پس از بارگذاری صفحه، به یک قسمت خاص از همان صفحه پرش کند (مثل یک تیتر یا بخش مشخص).

نکته: این بخش به سرور ارسال نمی‌شود و فقط توسط مرورگر پردازش می‌گردد.

یک مثال واقعی از URL

آدرس `https://digikala.com/search/?q=mobile` را در نظر بگیرید:

پروتکل: `https://` (ارتباط امن)

نام میزبان: `digikala.com`

مسیر: `/search/` (به صفحه جستجو برو)

پارامترهای Query: `?q=mobile` (عبارت جستجو شده "mobile" است)

تفاوت URL و Domain (مهم!)

این دو مفهوم مرتبط اما متفاوت هستند:

دامنه (Domain): فقط بخش نام و آدرس اصلی وب‌سایت است (مثلاً `google.com`).

URL (نشانی وب): یک آدرس کامل و دقیق است که شامل پروتکل، دامنه، مسیر و سایر پارامترها می‌شود تا یک منبع خاص را پیدا کند (مثلاً `https://google.com/maps`).

به زبان ساده: دامنه مانند نام شهر است، اما URL مانند آدرس کامل یک ساختمان خاص در آن شهر است.

انواع URL

1. URL مطلق (Absolute): شامل تمام بخش‌های لازم (پروتکل، میزبان و مسیر) برای پیدا کردن منبع است.

مثال: `https://example.com/images/photo.jpg`

2. URL نسبی (Relative): فقط مسیر را نسبت به آدرس فعلی مشخص می‌کند. این نوع URL در طراحی صفحات وب بسیار کاربرد دارد.

اگر در صفحه `https://example.com/about/` باشیم، یک URL نسبی مانند `../images/photo.jpg` به آدرس `https://example.com/images/photo.jpg` اشاره می‌کند.

چرا URL مهم است؟

1. مسیریابی: بدون URL، مرورگر نمی‌داند چگونه اطلاعات را از سرور درخواست کند.

2. ساختاردهی به وب: URLها به سازماندهی منطقی اطلاعات در وب کمک می‌کنند.

3. سئو (SEO): URLهای خوانا و توصیفی (مثل `example.com/blog/seo-tips`) برای موتورهای جستجو بهتر هستند و رتبه بهتری دارند.

4. قابلیت اشتراک‌گذاری: کاربران می‌توانند آدرس دقیق یک صفحه را با دیگران به اشتراک بگذارند.

جمع‌بندی نکات کلیدی

|  |  |
| --- | --- |
| مفهوم | توضیح خلاصه |
| تعریف | آدرس کامل و دقیق یک منبع (صفحه، عکس، فایل) در وب |
| تشبیه | آدرس پستی کامل یک ساختمان |
| اجزای اصلی | پروتکل (قوانین) + میزبان (آدرس سرور) + مسیر (محل فایل) + پارامترها (داده اضافی) |
| تفاوت با دامنه | دامنه بخشی از URL است. URL یک آدرس کامل‌تر است |
| مثال | <https://www.google.com/maps> |

# بررسی مفهوم HTTP:

HTTP چیست؟

فرض کنید شما در یک رستوران هستید. شما (مرورگر) یک درخواست (Request) به پیشخدمت (سرور) می‌دهید: "یک ساندویچ مرغ می‌خواهم."

پیشخدمت درخواست شما را می‌شنود، به آشپزخانه می‌رود و سپس یک پاسخ (Response) برای شما می‌آورد: "بسیار خوب، این هم ساندویچ مرغ شما."

HTTP دقیقاً نقش قوانین و زبان مشترک برای این تبادل درخواست و پاسخ بین مرورگر و سرور را بازی می‌کند.

مرورگر (Client) = مشتری رستوران

سرور (Server) = پیشخدمت رستوران

HTTP = پروتکل و قوانین ارتباطی بین مشتری و پیشخدمت

HTTP مخفف Hypertext Transfer Protocol به معنای "پروتکل انتقال ابرمتن" است.

تعریف فنی

HTTP یک پروتکل لایه کاربرد (Application Layer) است که برای انتقال ابرمتن‌ها (مانند صفحات وب) و دیگر محتوای چندرسانه‌ای در شبکه جهانی وب استفاده می‌شود. HTTP پایه و اساس هرگونه تبادل داده در وب است.

HTTP چگونه کار می‌کند؟ (درخواست-پاسخ)

کاربر در مرورگر خود آدرس سایتی مانند `http://example.com` را وارد می‌کند.

1. درخواست (Request): مرورگر یک درخواست HTTP به سرور می‌فرستد. این درخواست شامل اطلاعاتی مانند:

روش (Method): نوع عمل درخواستی (مثلاً GET برای دریافت داده).

مسیر (URL): آدرس منبع درخواستی.

هدرها (Headers): اطلاعات اضافی درباره درخواست.

2. پاسخ (Response): سرور درخواست را پردازش کرده و یک پاسخ HTTP برمی‌گرداند. این پاسخ شامل:

کد وضعیت (Status Code): عددی که نشان می‌دهد درخواست موفق بود یا خطایی رخ داده.

هدرهای پاسخ (Response Headers): اطلاعاتی درباره پاسخ.

بدنه (Body): محتوای اصلی درخواست شده (مثلاً کد HTML صفحه).

ویژگی‌های کلیدی HTTP

1. بدون حالت (Stateless):

مفهوم: سرور، وضعیت قبلی مرورگر (مثلاً اینکه درخواست قبلی چه بوده) را به خاطر نمی‌سپارد. هر درخواست مستقل از درخواست‌های قبل است.

مشکل: چگونه وب‌سایت‌ها شما را به خاطر می‌سپارند (مثلاً لاگین می‌مانید)؟

راه‌حل: از کوکی‌ها (Cookies) استفاده می‌شود. کوکی‌ها اطلاعات را در سمت مرورگر ذخیره کرده و با هر درخواست به سرور ارسال می‌کنند تا سرور "شما" را بشناسد.

2. مبتنی بر متن (Text-based):

دستورات و هدرهای HTTP به صورت متن ساده و قابل خواندن برای انسان هستند.

روش‌های رایج HTTP (HTTP Methods)

این روش‌ها عمل مورد نظر مرورگر را مشخص می‌کنند (مثل فعل در جمله):

|  |  |  |
| --- | --- | --- |
| روش | کاربرد | مثال |
| GET | دریافت داده از سرور | بارگذاری یک صفحه وب |
| POST | ارسال داده به سرور | ارسال اطلاعات فرم لاگین |
| PUT | به‌روزرسانی داده موجود در سرور | ویرایش پروفایل کاربر |
| DELETE | حذف داده از سرور | پاک کردن یک پست |

کدهای وضعیت HTTP (HTTP Status Codes)

این کدهای سه‌رقمی نتیجه درخواست را به صورت عددی نشان می‌دهند (مثلاً آیا موفق بود یا خطایی رخ داد؟):

|  |  |  |
| --- | --- | --- |
| دسته | معنی | مثال‌های معروف |
| 1xx (اطلاعاتی) | درخواست دریافت شده و پردازش ادامه دارد | - |
| 2xx (موفقیت) | درخواست با موفقیت دریافت و پردازش شد | 200 OK (موفقیت‌آمیز) |
| 3xx (تغییر مسیر) | برای تکمیل درخواست، عمل اضافی لازم است | 301 Moved Permanently (صفحه به آدرس جدید منتقل شده) |
| 4xx (خطای سمت کلاینت) | درخواست حاوی خطا یا غیرمعتبر است | 404 Not Found (صفحه پیدا نشد) |
| 5xx (خطای سمت سرور) | سرور در پردازش درخواست موفق نبود | 500 Internal Server Error (خطای داخلی سرور) |

معایب HTTP (چرا HTTPS ایجاد شد؟)

اطلاعات در HTTP به صورت متن ساده (Plain Text) منتقل می‌شوند. این یعنی:

اگر کسی ارتباط شما را استراق سمع کند، می‌تواند تمام اطلاعاتی که رد و بدل می‌شود (مانند رمز عبور، اطلاعات کارت بانکی) را به راحتی بخواند.

این یک مشکل بزرگ امنیتی است.

تفاوت HTTP با HTTPS

HTTPS (HTTP Secure) نسخه امن HTTP است.

مکانیزم: HTTPS از پروتکل SSL/TLS برای رمزگذاری (Encrypt) کردن تمام ارتباط بین مرورگر و سرور استفاده می‌کند.

نتیجه: حتی اگر داده‌ها intercepted شوند، برای فرد مهاجم قابل خواندن نیستند.

شناخت: وجود یک قفل در نوار آدرس مرورگر، نشان‌دهنده استفاده از HTTPS است.

امروزه استفاده از HTTPS برای تمام وب‌سایت‌ها به یک استاندارد ضروری تبدیل شده است.

# بررسی مفهوم HTTPS

حتماً! در ادامه به بررسی مفهوم HTTPS (پروتکل امن انتقال ابرمتن) به زبان ساده و جامع می‌پردازیم.

HTTPS چیست؟ (تشبیه ساده)

فرض کنید می‌خواهید یک محموله بسیار باارزش را برای دوست خود بفرستید. اگر آن را در یک جعبه معمولی بگذارید (مثل HTTP)، هر کسی در مسیر می‌تواند جعبه را باز کند، محتویات را ببیند یا حتی تغییر دهد.

اما اگر محموله را داخل یک جعبه امنیتی ضدگلوله قرار دهید که فقط شما و دوستتان کلید باز کردن آن را دارید (مثل HTTPS)، حتی اگر دزدان راهی هم جعبه را بدزدند، نمی‌توانند آن را باز کنند یا محتویات را بفهمند.

HTTPS دقیقاً همان جعبه امنیتی برای ارتباطات اینترنتی است.

HTTPS = نسخه امن شده HTTP که ارتباط بین مرورگر و سرور را رمزگذاری (Encrypt) می‌کند.

HTTPS مخفف Hypertext Transfer Protocol Secure به معنای "پروتکل امن انتقال ابرمتن" است.

تعریف فنی

HTTPS یک پروتکل ارتباطی امن برای انتقال داده‌ها در شبکه اینترنت است که با استفاده از پروتکل SSL/TLS، یک لایه امنیتی به پروتکل استاندارد HTTP اضافه می‌کند. این پروتکل از سه جنبه مهم محافظت می‌کند:

1. رمزگذاری (Encryption)

2. اعتبارسنجی (Authentication)

3. یکپارچگی داده (Data Integrity)

HTTPS چگونه کار می‌کند؟ (SSL/TLS Handshake)

وقتی شما به یک سایت HTTPS متصل می‌شوید (مثلاً `https://bank.com`)، قبل از شروع تبادل داده‌های اصلی، یک فرآیند امنیتی به نام SSL/TLS Handshake بین مرورگر و سرور انجام می‌شود.

مزایای کلیدی HTTPS

1. رمزگذاری داده (Encryption):

مهمترین مزیت. تمام داده‌هایی که بین شما و سایت رد و بدل می‌شود (رمز عبور، اطلاعات کارت بانکی، پیام‌های خصوصی) به صورت رمزگذاری شده انتقال می‌یابند.

حتی اگر کسی ارتباط را استراق سمع کند، نمی‌تواند محتوای آن را بخواند.

2. اعتبارسنجی (Authentication):

HTTPS با استفاده از گواهی SSL تأیید می‌کند که شما در حال ارتباط با وب‌سایت واقعی و قانونی هستید، نه یک سایت جعلی که خود را به جای آن جا زده است.

این گواهی را یک مرجع معتبر (Certificate Authority) صادر می‌کند.

3. حفظ یکپارچگی داده (Data Integrity):

HTTPS تضمین می‌کند که داده‌ها در مسیر انتقال دستکاری یا تغییر نکرده‌اند.

چگونه بفهمیم سایتی از HTTPS استفاده می‌کند؟

در نوار آدرس مرورگر خود به دنبال این نشانه‌ها باشید:

1. قفل (🔒) بسته: اصلی‌ترین نشانه. معمولاً کنار آدرس سایت نمایش داده می‌شود.

2. شروع آدرس با `https://`: به جای `http://`.

3. علامت "امن" (Secure): در برخی مرورگرها کلمه "امن" یا "Secure" نوشته می‌شود.

تفاوت HTTP با HTTPS (مقایسه کلی)

|  |  |  |
| --- | --- | --- |
| ویژگی | HTTPS | HTTP |
| امنیت | ندارد - داده‌ها به صورت متن ساده منتقل می‌شوند | دارد - داده‌ها کاملاً رمزگذاری می‌شوند |
| پورت پیش‌فرض | پورت ۸۰ | پورت ۴۴۳ |
| الگوریتم رمزنگاری | استفاده نمی‌کند | از SSL/TLS استفاده می‌کند |
| سرعت | کمی سریع‌تر (به دلیل عدم overhead رمزگذاری) | کمی کندتر (به دلیل پردازش‌های رمزگذاری) |
| امنیت برای | مناسب اطلاعات عمومی و غیرحساس | ضروری برای اطلاعات حساس (لاگین، بانک، پرداخت) |
| نماد در مرورگر | ⓘ یا "ناامن" | 🔒 یا "امن" |

چرا HTTPS امروزه ضروری است؟

1. حفاظت از اطلاعات کاربران: اولین و مهمترین دلیل.

2. اعتماد و اعتبار (Credibility): کاربران به سایت‌هایی که قفل سبز رنگ دارند بیشتر اعتماد می‌کنند.

3. سئو (SEO): گوگل به سایت‌های دارای HTTPS رتبه بهتری می‌دهد.

4. امکان استفاده از قابلیت‌های جدید وب: بسیاری از APIهای مدرن مرورگر (مانند دسترسی به مکان) فقط برای سایت‌های HTTPS فعال هستند.

5. افزایش امنیت کلی اینترنت.

گواهی SSL (SSL Certificate) چیست؟

برای راه‌اندازی HTTPS، صاحب وب‌سایت باید یک گواهی SSL از یک مرجع معتبر خریداری یا دریافت کند. این گواهی مانند یک کارت شناسایی دیجیتال برای وب‌سایت است که:

هویت صاحب سایت را تأیید می‌کند.

حاوی یک کلید عمومی برای برقراری ارتباط امن است.

# بررسی سایت‌های Static و Dynamic

سایت‌های استاتیک و داینامیک دو نوع اصلی از وبسایت‌ها هستند که از نظر فنی، کاربرد و مدیریت با یکدیگر تفاوت‌های بنیادینی دارند. انتخاب بین این دو به طور کامل به اهداف و نیازهای کسب‌وکار شما بستگی دارد.

در جدول زیر می‌توانید خلاصه‌ای از مهم‌ترین تفاوت‌های این دو نوع سایت را مشاهده کنید:

|  |  |  |
| --- | --- | --- |
| ویژگی | سایت استاتیک (ایستا) | سایت داینامیک (پویا) |
| محتوا و تعامل | محتوای ثابت و از پیش نوشته شده. فاقد تعامل کاربر (مانند ثبت‌نام یا نظر دادن) . | محتوای پویا و تغییرپذیر. تعامل کامل کاربر (ثبت‌نام، خرید، سیستم کاربری و غیره) . |
| فناوری‌های مورد استفاده | عمدتاً HTML و CSS، گاهی جاوااسکریپت . | زبان‌های سمت سرور (مانند PHP، ASP.NET) و یک پایگاه داده (Database) |
| مدیریت و به‌روزرسانی | برای هر تغییر کوچک نیاز به دانش فنی و تغییر کدها توسط برنامه‌نویس است . | از طریق پنل مدیریت (CMS)؛ به راحتی و بدون نیاز به کدنویسی قابل مدیریت است . |
| هزینه | هزینه طراحی اولیه کمتر، اما هزینه به‌روزرسانی‌های مکرر می‌تواند بالا باشد . | هزینه طراحی اولیه بالاتر، اما به‌صرفه‌تر در بلندمدت به دلیل مدیریت آسان . |
| سرعت بارگذاری | سرعت بسیار بالا به دلیل سادگی ساختار و نبود درخواست به پایگاه داده . | سرعت نسبتاً کمتر (به دلیل پردازش‌های سمت سرور)، اما قابل بهینه‌سازی |
| امنیت | امنیت بالاتر به دلیل سادگی ساختار و نداشتن پایگاه داده و نقطه ورود پیچیده . | نیاز به تمهیدات امنیتی بیشتر به دلیل پیچیدگی و وجود پایگاه داده . |
| مثال‌های کاربردی | سایت‌های معرفی (پورتفولیو)، رزومه آنلاین، صفحات لندینگ، بروشورهای الکترونیک | فروشگاه‌های اینترنتی، سایت‌های خبری، شبکه‌های اجتماعی، وبلاگ‌ها، پرتال‌های سازمانی |

چگونه بین سایت استاتیک و داینامیک انتخاب کنیم؟

انتخاب صحیح به طور مستقیم به پاسخ این سوال بستگی دارد: هدف اصلی شما از راه‌اندازی وبسایت چیست؟

- یک سایت استاتیک مناسب شماست اگر:

- نیاز به یک بروشور آنلاین ساده و ثابت دارید که اطلاعات پایه شرکت یا خدمات شما را نمایش دهد.

- محتوای سایت شما به ندرت نیاز به تغییر دارد (مثلاً یک یا دو بار در سال).

- بودجه اولیه کمی دارید و اولویت شما سرعت بسیار بالا و امنیت است.

- نیازی به ویژگی‌های پیچیده‌ای مانند عضویت کاربران، سبد خرید یا درگاه پرداخت ندارید .

- یک سایت داینامیک انتخاب بهتری است اگر:

- قصد دارید به طور منظم محتوا منتشر کنید (مثلاً مقاله، خبر یا محصول جدید).

- نیازمند تعامل با کاربران هستید (مانند نظردهی، ثبت‌نام، سیستم سفارش‌دهی).

- می‌خواهید خودتان به راحتی و بدون وابستگی به برنامه‌نویس، سایت را مدیریت کنید.

- کسب‌وکار شما در حال رشد است و به قابلیت‌های گسترش‌پذیر نیاز دارید .

یک نکته مهم: راه حل‌های مدرن (Static Site Generators)

امروزه با فناوری‌هایی مانند Static Site Generators (مثل Gatsby, Next.js) می‌توان تا حدی مرز بین این دو نوع سایت را کمرنگ کرد. این فناوری‌ها امکان ایجاد سایت‌هایی با سرعت و امنیت سایت‌های استاتیک را فراهم می‌کنند، در حالی که محتوای آن‌ها می‌تواند از یک سیستم مدیریت محتوا (CMS) به صورت پویا استخراج شود. این گزینه می‌تواند برای پروژه‌های خاصی بسیار ایده‌آل باشد.

# بررسی request و Response های یک وب سایت

Request و Response چیست؟

فرض کنید شما به یک رستوران رفتید:

1. شما (مرورگر) به پیشخدمت (سرور) می‌گویید: "یک پیتزا پپرونی سفارش می‌دهم."

→ این یک Request (درخواست) است.

2. پیشخدمت (سرور) به آشپزخانه می‌رود، پیتزا را آماده می‌کند و برای شما (مرورگر) می‌آورد: "بفرمایید پیتزای شما."

→ این یک Response (پاسخ) است.

این مکالمه پایه و اساس تمام ارتباطات در دنیای وب است.

تعریف فنی

Request (درخواست): پیامی است که مرورگر (Client) به سرور می‌فرستد تا اطلاعات یا سرویس خاصی را درخواست کند.

Response (پاسخ): پیامی است که سرور پس از پردازش Request، برای مرورگر می‌فرستد و حاوی نتیجه درخواست (داده‌های مورد نظر یا خطا) است.

Request و Response چگونه کار می‌کنند؟

وقتی شما آدرس یک سایت را در مرورگر وارد می‌کنید یا روی لینکی کلیک می‌کنید، این فرآیند به طور مداوم تکرار می‌شود.

اجزای تشکیل‌دهنده یک Request (درخواست)

هر Request معمولاً شامل سه بخش اصلی است:

۱. خط شروع (Request Line)

- متد (Method): نوع عمل درخواستی را مشخص می‌کند.

- آدرس (URL): منبع درخواستی را مشخص می‌کند.

- ورژن پروتکل: مانند `HTTP/1.1`

مثال: `GET /products/index.html HTTP/1.1`

۲. هدرها (Headers)

اطلاعات اضافی درباره درخواست را شامل می‌شوند:

- `Host: example.com` (دامنه سایت مقصد)

- `User-Agent: Mozilla/5.0...` (اطلاعات مرورگر و سیستم عامل)

- `Cookie: session\_id=abc123` (اطلاعات ذخیره شده در مرورگر)

- `Accept: text/html,application/xhtml+xml` (نوع داده‌هایی که مرورگر می‌تواند دریافت کند)

۳. بدنه (Body) - اختیاری

حاوی داده‌ای است که به سرور ارسال می‌شود (مثلاً اطلاعات فرم ثبت نام).

انواع متدهای رایج Request (HTTP Methods)

|  |  |  |
| --- | --- | --- |
| متد | کاربرد | مثال |
| GET | دریافت اطلاعات از سرور | بارگذاری یک صفحه وب |
| POST | ارسال اطلاعات به سرور | ثبت نام کاربر، ارسال نظر |
| PUT | به‌روزرسانی اطلاعات موجود | ویرایش پروفایل کاربر |
| DELETE | حذف اطلاعات | پاک کردن یک پست |

اجزای تشکیل‌دهنده یک Response (پاسخ)

هر Response نیز شامل سه بخش اصلی است:

۱. خط وضعیت (Status Line)

- ورژن پروتکل: مانند `HTTP/1.1`

- کد وضعیت (Status Code): عدد سه رقمی که نتیجه درخواست را نشان می‌دهد.

- توضیح وضعیت: شرح متنی کد وضعیت.

مثال: `HTTP/1.1 200 OK`

۲. هدرهای پاسخ (Response Headers)

اطلاعات اضافی درباره پاسخ:

- `Content-Type: text/html` (نوع محتوای بازگشتی)

- `Content-Length: 1024` (حجم داده بازگشتی)

- `Set-Cookie: session\_id=xyz456` (ذخیره اطلاعات در مرورگر)

۳. بدنه (Body)

محتوای اصلی پاسخ (مثلاً کد HTML صفحه، داده JSON، یا یک فایل).

یک مثال واقعی از Request و Response

موقعیت: کاربر روی لینک "تماس با ما" کلیک می‌کند.

Request ارسالی توسط مرورگر:

```

GET /contact-us.html HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36

Accept: text/html,application/xhtml+xml

Cookie: user\_lang=fa

```

Response دریافتی از سرور:

```

HTTP/1.1 200 OK

Content-Type: text/html; charset=utf-8

Content-Length: 1542

Date: Sat, 01 Jan 2024 12:00:00 GMT

HTML Response come here:

```

ابزارهای مشاهده Request و Response (برای توسعه‌دهندگان)

- Developer Tools مرورگر: در Chrome/Firefox با F12 باز می‌شود. تب Network تمام Request/Responseها را نشان می‌دهد.

- Postman: ابزاری برای تست و بررسی APIها با ساخت Requestهای دستی.

- cURL: یک ابزار خط فرمان برای ارسال Request.

# بررسی نحوه رندرگیری کدها توسط مرورگر

مرورگرها برای تبدیل کدهای HTML، CSS و JavaScript به صفحات وب قابل مشاهده، مراحل پیچیده و به هم پیوستهای را طی می کنند. درک این فرآیند به شما کمک میکند تا وبسایتهای سریعتر و کاراتری توسعه دهید.

در جدول زیر مراحل اصلی این فرآیند بهطور خلاصه آمده است:

|  |  |  |
| --- | --- | --- |
| مرحله | توضیح | خروجی |
| ۱. پارس کردن HTML | مرورگر کد HTML را خط به خط میخواند و به یک ساختار درختی از اشیاء تبدیل میکند. | DOM (Document Object Model) |
| ۲. پارس کردن CSS | استایلهای CSS نیز به یک ساختار درختی قابل فهم برای مرورگر تبدیل میشوند | CSSOM (CSS Object Model |
| ۳. ساخت درخت رندر | DOM و CSSOM با هم ترکیب میشوند تا درختی ساخته شود که فقط شامل عناصر قابل مشاهده صفحه است (عناصری مانند `display: none` در این درخت قرار نمیگیرند). | Render Tree (درخت رندر) |
| ۴. لایه بندی (Layout) | مرورگر موقعیت و اندازه دقیق هر عنصر در صفحه را محاسبه میکند. این مرحله گاهی Reflow نیز نامیده میشود. | موقعیت و ابعاد هندسی |
| ۵. رنگآمیزی (Painting) | مرورگر پیکسلهای هر عنصر (مانند رنگها، تصاویر و حاشیهها) را رسم میکند. این مرحله Repaint نیز نام دارد. | لایه های پیکسلی |
| ۶. ترکیب (Compositing) | در نهایت، لایههای مختلفی که رنگآمیزی شدهاند، به ترتیب صحیح روی هم قرار میگیرند تا صفحه نهایی تشکیل شود. | صفحه وب نهایی |

نقش JavaScript و بهینه سازی عملکرد

JavaScript به طور خاص بر روند رندر تأثیر میگذارد، زیرا معمولاً تجزیه و اجرای آن باعث توقف ساخت DOM میشود. برای کاهش این تأثیر منفی:

- جاوااسکریپت را در انتهای صفحه (قبل از تگ بسته `body`) قرار دهید تا ابتدا محتوای HTML بارگذاری و پارس شود.

- از ویژگیهای async` یا defer` برای اسکریپت های خارجی استفاده کنید تا بارگیری آنها به صورت غیرمسدودکننده انجام شود.

- مسیر رندر بحرانی را بهینه کنید: تعداد منابع (فایلهای CSS و JS) را کاهش دهید، حجم آنها را با فشردهسازی (Minify) کم کنید و با تکنیکهایی مانند Lazy Loading بارگذاری منابع غیرضروری برای نمایش اولیه را به تأخیر بیندازید.

تفاوت Repaint و Reflow

دو مفهوم مهم در بهروزرسانی صفحات هستند:

- Repaint (رنگآمیزی مجدد): هنگامی رخ میدهد که ظاهر یک عنصر تغییر کند، اما بر Layout صفحه تأثیر نگذارد (مانند تغییر رنگ پسزمینه). این عمل نسبتاً سبک است.

- Reflow (بازچینی): هنگامی رخ میدهد که تغییری در صفحه ایجاد شود که بر Layout و موقعیت عناصر تأثیر بگذارد (مانند تغییر اندازه پنجره مرورگر یا اضافه کردن یک عنصر DOM). این عمل پرهزینه است و ممکن است باعث شود مرورگر بخشی یا تمام صفحه را مجدداً محاسبه کند.

نتیجه گیری

درک دقیق فرآیند رندر مرورگر یک مزیت بزرگ برای توسعهدهندگان وب است. با بهینهسازی هر یک از این مراحل، به ویژه مسیر رندر بحرانی، میتوانید تجربه کاربری بسیار بهتری از نظر سرعت بارگذاری و تعامل روان ایجاد کنید.

اگر درباره تأثیر یک المان یا تکنیک خاص (مثلاً استفاده از فونتهای وب یا تصاویر بزرگ) بر رندر صفحه سؤال خاصی دارید، خوشحال میشوم در مورد آن نیز صحبت کنیم.

# آموزش HTML

## [کامنت ها در HTML](webdesign/html/comments)

کامنت‌ها در HTML ابزار بسیار مهمی برای توسعه‌دهندگان هستند که توسط مرورگر نمایش داده نمی‌شوند اما برای اهداف مختلفی استفاده می‌شوند.

نحوه نوشتن کامنت در HTML

|  |
| --- |
| <!-- این یک کامنت تک خطی است -->  <!--  این یک کامنت  چند خطی است  -->  <div class="container">  <!-- کامنت داخل عناصر -->  <p>محتوا</p>  </div> |

[کامنت ها مثال اول](webdesign/html/comments/first.html)

کاربردهای اصلی کامنت‌ها

1. توضیح کد (Documentation)

|  |
| --- |
| <!-- نوار ناوبری اصلی سایت -->  <nav>  <ul>  <li><a href="/">خانه</a></li>  <li><a href="/about">درباره ما</a></li>  </ul>  </nav>  <!-- بخش مقالات -->  <section id="articles">  <!-- هر مقاله در یک کارت نمایش داده می‌شود -->  <article class="card">  <h3>عنوان مقاله</h3>  <p>متن مقاله...</p>  </article>  </section> |

[Documentation](webdesign/html/comments/Documentation.html)

1. غیرفعال کردن موقت کد (Debugging)

|  |
| --- |
| <!-- این بخش در حال توسعه است -->  <!--  <div class="feature">  <p>این ویژگی به زودی اضافه می‌شود</p>  </div>  -->  <!-- کد قدیمی که ممکن است بعداً نیاز شود -->  <!--  <button class="old-button">دکمه قدیمی</button>  --> |

[Debugging](webdesign/html/comments/Debugging.html)

1. علامت‌گذاری و سازماندهی

|  |
| --- |
| <!-- ===== HEADER START ===== -->  <header>  <!-- لوگو -->  <div class="logo">سایت من</div>    <!-- منوی اصلی -->  <nav>...</nav>  </header>  <!-- ===== HEADER END ===== -->  <!-- ===== MAIN CONTENT START ===== -->  <main>...</main>  <!-- ===== MAIN CONTENT END ===== --> |

[categorized](webdesign/html/comments/cat.html)

نکات مهم درباره کامنت‌ها

نحوه پردازش توسط مرورگر:

- کامنت‌ها جزئی از DOM نمی‌شوند

- در Render Tree لحاظ نمی‌شوند

- بر عملکرد مرورگر تأثیر منفی نمی‌گذارند

- در سورس کد صفحه قابل مشاهده هستند

بهترین تمرین :

|  |
| --- |
| <!-- خوب: کامنت واضح و مختصر -->  <!-- منوی کاربر -->  <div class="user-menu">...</div>  <!-- بد: کامنت بی‌معنی -->  <!-- div -->  <div>...</div>  <!-- خوب: کامنت برای کد پیچیده -->  <!-- محاسبه قیمت با درنظرگیری تخفیف و مالیات -->  <div class="price-calculator">  <script>  // کدهای پیچیده JavaScript  </script>  </div> |

[practices](webdesign/html/comments/practices.html)

کامنت‌های شرطی برای مرورگرهای قدیمی

|  |
| --- |
| <!-- نمایش پیام برای کاربران IE قدیمی -->  <!--[if lt IE 9]>  <div class="browser-warning">  لطفاً مرورگر خود را آپدیت کنید  </div>  <![endif]--> |

[if](webdesign/html/comments/if.html)

نکات پیشرفته

کامنت‌ها در ابزارهای توسعه:

کامنت‌ها در DevTools مرورگر قابل مشاهده هستند و به دیباگ کمک می‌کنند.

تأثیر بر سئو:

کامنت‌ها معمولاً بر سئو تأثیر مستقیم ندارند، اما کامنت‌های زیاد ممکن است نسبت کد به محتوا را کاهش دهند.

امنیت:

هرگز اطلاعات حساس در کامنت نگذارید:

|  |
| --- |
| <!-- بد: اطلاعات حساس -->  <!-- ادمین: admin, رمز: 123456 -->  <!-- بد: کد مخفی -->  <!-- API Key: ak\_123456789 --> |

[sensetive](webdesign/html/comments/sensetive.html)

جمع‌بندی

کامنت‌ها ابزار قدرتمندی برای:

- بهبود خوانایی کد

- دیباگ راحت‌تر

- مستندسازی پروژه

- همکاری تیمی بهتر

اما به یاد داشته باشید:

- زیاده‌روی نکنید

- اطلاعات حساس نگذارید

- کدهای قدیمی را حذف کنید

## [معرفی tag ها در Html و انواع آن (singel & pairs)](webdesign/html/singel-pairs)

HTML از تگ‌ها (Tags) برای ساختاردهی و نمایش محتوای وب استفاده می‌کند. تگ‌ها به دو دسته اصلی تقسیم می‌شوند:

انواع تگ‌ها از نظر ساختاری

۱. تگ‌های جفتی (Paired Tags)

این تگ‌ها دارای تگ آغازین و پایانی هستند.

|  |
| --- |
| <tag>محتوای داخل تگ</tag> |

[PairedTags](webdesign/html/singel-pairs/PairedTags.html)

مثال‌ها:

|  |
| --- |
| <p>این یک پاراگراف است</p>  <div>این یک بخش است</div>  <h1>این یک عنوان است</h1>  <a href="https://example.com">این یک لینک است</a> |

[Example](webdesign/html/singel-pairs/sample.html)

۲. تگ‌های تکی (Single Tags)

این تگ‌ها فقط تگ آغازین دارند و محتوایی بین آنها قرار نمی‌گیرد.

|  |
| --- |
| <tag> |

[Singletag](webdesign/html/singel-pairs/Singletag.html)

مثال‌ها:

|  |
| --- |
| <br> <!-- خط شکست -->  <hr> <!-- خط افقی -->  <img> <!-- تصویر -->  <input> <!-- فیلد ورودی -->  <meta> <!-- متادیتا -->  <link> <!-- لینک به فایل خارجی --> |

[Samples2](webdesign/html/singel-pairs/samples2.html)

جدول مقایسه تگ‌های جفتی و تکی

|  |  |  |
| --- | --- | --- |
| ویژگی | تگ‌های جفتی | تگ‌های تکی |
| ساختار | `<tag>  محتوا  </tag>` | <tag /> |
| محتوای داخلی | دارد | ندارد |
| تگ پایانی | الزامی | ندارد |
| نحوه بستن | `</tag>` | <tag /> اختیاری |
| مثال | <p>  متن پاراگراف  </p> | <br> |

دسته‌بندی تگ‌ها بر اساس کاربرد

## [ساختار تگ ها در HTML](webdesign/html/tagsStructure)

درک پایه: تگ + صفت = عنصر کامل

فرض کنید هر تگ HTML مانند یک آدمک خام است و صفت‌ها مانند لباس‌ها و ابزار هستند که به آن ویژگی می‌دهند!

ساختار کلی:

|  |
| --- |
| <!-- attr => صفت است -->  <!-- value => مقدار صفت است که درون کوتیشن اجباری درج میشود-->  <tag attr="value">      محتوای تگ  </tag> |

[base](webdesign/html/tagsStructure/base.html)

انواع ساختار با صفت‌ها

مثال‌های ملموس:

|  |
| --- |
| <a href="https://google.com">برو به گوگل</a> |

[Sample](webdesign/html/tagsStructure/sample.html)

مثل نوشتن آدرس روی یک بلیط

|  |
| --- |
| <p dir="rtl">      این متن راست به چپ است  </p>  <p dir="ltr">      This content is left to right.  </p> |

[dir](webdesign/html/tagsStructure/dir.html)

مثل دادن یک اسم به جعبه

|  |
| --- |
| <div class="container">      محتوای داخل کانتینر  </div> |

[container](webdesign/html/tagsStructure/container.html)

۲. تگ‌های تکی با صفت

|  |
| --- |
| <tag attr="value"> |

## معرفی VS Code و پلاگین های آن

(material icons, material themes , open in browser , prettier ,rainbow brackets , path intellisense)

ویژوال استودیو کد (VS Code) یک ویرایشگر کد سبک‌وزن، قدرتمند و رایگان از مایکروسافت است که محبوبیت بسیار زیادی در بین توسعه‌دهندگان دارد. این ویرایشگر به لطف معماری مبتنی بر افزونه، امکان شخصی‌سازی و گسترش امکانات را تا حد زیادی فراهم می‌کند.

در ادامه، افزونه‌هایی که شما نام بردید را به طور کامل معرفی می‌کنم.

پلاگین‌های ظاهری و بصری

این دسته از افزونه‌ها به زیباتر شدن و کاربردی‌تر شدن محیط کدنویسی کمک می‌کنند.

|  |  |
| --- | --- |
| افزونه | توضیح |
| Material Icon Theme | آیکن‌های زیبا و مدرن بر اساس اصول Material Design برای فایل‌ها و پوشه‌های شما درون VS Code. |
| Material Theme | یک پوسته رنگی کامل که تمام محیط VS Code را بر اساس طراحی متریال زیبا می‌کند. |

پلاگین‌های افزایش بهره‌وری و عملکرد

این افزونه‌ها مستقیماً به شما در کدنویسی سریع‌تر، تمیزتر و با خطای کمتر کمک می‌کنند.

|  |  |
| --- | --- |
| افزونه | توضیح |
| Prettier | یک قالب‌بند (Formatter) کد که به صورت خودکار کدهای شما را مرتب و زیبا می‌کند. |
| Path Intellisense | هنگام نوشتن آدرس فایل‌ها (مثلاً در تگ `img` یا دستور `import`)، به صورت خودکار نام فایل‌ها و پوشه‌ها را پیشنهاد می‌دهد. |
| Rainbow Brackets | پرانتزها، کروشه‌ها و آکولادها را به صورت رنگین‌کمانی رنگ می‌کند |
| Open in Browser | این افزونه امکان می‌دهد فایل HTMLای که در حال ویرایش آن هستید را مستقیماً و با یک کلیک در مرورگر پیش‌فرض سیستم یا مرورگر دلخواهتان باز کنید. |

💡 نکات نصب و استفاده

- برای نصب، به تب Extensions در VS Code رفته (آیکون چهارگوش در نوار کناری) و نام افزونه را جستجو کنید.

- پس از نصب Prettier، بهتر است آن را به عنوان فرمت‌کننده پیش‌فرض برای زبان‌های مورد نظرتان در تنظیمات VS Code تنظیم کنید تا به طور خودکار upon save کدهایتان را مرتب کند.

- برای Open in Browser معمولاً از کلیدهای `Alt + B` برای بازکردن در مرورگر پیش‌فرض و `Shift + Alt + B` برای انتخاب از بین مرورگرها استفاده می‌شود.

## [بررسی ساختار یک سند Html](webdesign/html/HtmlStructure)

ساختار پایه HTML5

|  |
| --- |
| <!DOCTYPE html>  <html lang="fa" dir="rtl">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>عنوان صفحه</title>  </head>  <body>  <!-- محتوای اصلی صفحه -->  </body>  </html> |

تگ `<!DOCTYPE html>`

نقش و اهمیت:

- اولین خط در هر سند HTML5

- تعیین نوع سند برای مرورگر

- اطمینان از رندر صحیح صفحه

سیر تکامل:

|  |
| --- |
| <!-- HTML 4.01 Strict -->  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  "http://www.w3.org/TR/html4/strict.dtd">  <!-- XHTML 1.0 -->  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <!-- HTML5 (ساده شده) -->  <!DOCTYPE html> |

تگ `<html>`

ویژگی‌های اصلی:

|  |
| --- |
| <html lang="fa" dir="rtl"> |

|  |  |  |
| --- | --- | --- |
| ویژگی | مقدار | توضیح |
| lang | fa`, `en`, `ar | زبان اصلی سند |
| dir | rtl`, `ltr | جهت نوشتار |
|  |  |  |

اهمیت `lang`:

- سئو (بهینه‌سازی برای موتورهای جستجو)

- دسترسی‌پذیری (برای نرم‌افزارهای صفحه‌خوان)

- ترجمه خودکار مرورگر

تگ `<head>` - مغز صفحه

محتوای تگ head:

|  |
| --- |
| <head>  <!-- اطلاعات متا -->  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">    <!-- عنوان و لینک‌ها -->  <title>فروشگاه اینترنتی</title>  <link rel="stylesheet" href="styles.css">    <!-- اسکریپت‌ها -->  <script src="script.js"></script>  </head> |

متا تگ‌های ضروری

1. تعیین encoding:

|  |
| --- |
| <meta charset="UTF-8"> |

- پشتیبانی از زبان فارسی

- نمایش صحیح کاراکترها

۲. تنظیم viewport:

|  |
| --- |
| <meta name="viewport" content="width=device-width, initial-scale=1.0"> |

- ریسپانسیو کردن سایت

- تنظیم اندازه برای موبایل

۳. توضیحات صفحه:

|  |
| --- |
| <meta name="viewport" content="width=device-width, initial-scale=1.0"> |

- تاثیر مستقیم بر سئو

- نمایش در نتایج جستجو

۴. کلمات کلیدی:

|  |
| --- |
| <meta name="viewport" content="width=device-width, initial-scale=1.0"> |

1. نویسنده صفحه:

|  |
| --- |
| <meta name="author" content="نام شما"> |

تگ `<title>`

اهمیت:

- نمایش در تب مرورگر

- عنوان در نتایج جستجو

- ذخیره در بوکمارک‌ها

مثال‌های خوب و بد:

|  |
| --- |
| <!-- بد: بدون عنوان -->  <title></title>  <!-- بد: عنوان عمومی -->  <title>صفحه جدید</title>  <!-- خوب: توصیفی و خاص -->  <title>فروشگاه اینترنتی دیجی‌کالا - خرید آنلاین</title> |

تگ `<body>` - بدنه اصلی

ساختار معمول:

|  |
| --- |
| <body>  <header>  <!-- هدر صفحه -->  </header>    <nav>  <!-- منوی -->  </nav>    <main>  <!-- محتوای اصلی -->  <article>  <!-- مقاله یا پست -->  </article>    <aside>  <!-- محتوای جانبی -->  </aside>  </main>    <footer>  <!-- پاورقی -->  </footer>  </body> |

تگ‌های معنایی جدید HTML5 در body:

|  |  |
| --- | --- |
| تگ | کاربرد |
| `<header>` | سربرگ صفحه یا بخش |
| `<nav>` | منوی ناوبری |
| `<main>` | محتوای اصلی (یکبار استفاده) |
| `<article>` | محتوای مستقل (پست، مقاله) |
| `<section>` | بخش بندی محتوا |
| `<aside>` | محتوای جانبی |
| `<footer>` | پاورقی |

مثال کامل یک سند HTML5

|  |
| --- |
| <!DOCTYPE html>  <html lang="fa" dir="rtl">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <meta name="description" content="وبلاگ شخصی علی محمدی - مقالات برنامه‌نویسی">  <meta name="keywords" content="برنامه‌نویسی, وب, آموزش, HTML5">  <meta name="author" content="علی محمدی">  <title>وبلاگ شخصی علی محمدی - برنامه‌نویسی وب</title>  <link rel="stylesheet" href="css/styles.css">  </head>  <body>  <header>  <h1>وبلاگ شخصی علی محمدی</h1>  <p>مقالات آموزشی برنامه‌نویسی وب</p>  </header>    <nav>  <ul>  <li><a href="home">خانه</a></li>  <li><a href="articles">مقالات</a></li>  <li><a href="about">درباره من</a></li>  </ul>  </nav>    <main>  <article>  <h2>آموزش HTML5 برای مبتدیان</h2>  <p>محتوای مقاله...</p>  </article>  </main>    <footer>  <p>© 2024 کلیه حقوق محفوظ است.</p>  </footer>  </body>  </html> |

[full](webdesign/html/HtmlStructure/full.html)

نکات کلیدی

بهترین روش‌ها:

1. همیشه با `<!DOCTYPE html>` شروع کنید

2. زبان و جهت صفحه را مشخص کنید

3. viewport را برای موبایل تنظیم کنید

4. از تگ‌های معنایی HTML5 استفاده کنید

5. عنوان توصیفی و منحصر به فرد انتخاب کنید

خطاهای رایج:

|  |
| --- |
| <!-- ❌ فراموش کردن DOCTYPE -->  <html>  <head>...</head>  <!-- ❌ قرار دادن محتوای قابل مشاهده در head -->  <head>  <h1>این درست نیست!</h1>  </head>  <!-- ❌ چندین تگ main -->  <main>...</main>  <main>...</main> |

# معرفی BOM و DOM

BOM - Browser Object Model (مدل شیءگرای مرورگر)

مفهوم کلی:

BOM مربوط به مرورگر است و به شما امکان کنترل پنجره مرورگر، تب‌ها، تاریخچه و... را می‌دهد.

ساختار سلسله مراتبی BOM:

|  |
| --- |
| window (پنجره مرورگر)  ├── navigator (اطلاعات مرورگر)  ├── location (آدرس صفحه)  ├── history (تاریخچه)  ├── screen (اطلاعات صفحه نمایش)  └── document (سند HTML - که خود DOM است) |

اجزای اصلی BOM:

1. window - شیء اصلی

|  |
| --- |
| // باز کردن پنجره جدید  window.open('https://example.com', '\_blank');  // بستن پنجره جاری  window.close();  // اندازه پنجره  console.log(window.innerWidth); // عرض داخلی  console.log(window.innerHeight); // ارتفاع داخلی |

1. navigator - اطلاعات مرورگر

|  |
| --- |
| console.log(navigator.userAgent); // شناسه مرورگر  console.log(navigator.language); // زبان مرورگر  console.log(navigator.onLine); // وضعیت اتصال اینترنت |

1. location - مدیریت آدرس

|  |
| --- |
| // تغییر مسیر  location.href = 'https://google.com';  // رفرش صفحه  location.reload();  // اطلاعات آدرس  console.log(location.protocol); // https:  console.log(location.hostname); // example.com  console.log(location.pathname); // /page.html |

1. history - مدیریت تاریخچه

|  |
| --- |
| // بازگشت به صفحه قبل  history.back();  // رفتن به صفحه بعد  history.forward();  // رفتن به چند صفحه قبل/بعد  history.go(-2); // دو صفحه قبل |

1. screen - اطلاعات نمایشگر

|  |
| --- |
| console.log(screen.width); // عرض کل نمایشگر  console.log(screen.height); // ارتفاع کل نمایشگر  console.log(screen.colorDepth); // عمق رنگ |

### DOM - Document Object Model (مدل شیءگرای سند)

مفهوم کلی:

DOM مربوط به سند HTML است و به شما امکان دسترسی و تغییر محتوای صفحه را می‌دهد.

ساختار درختی DOM:

|  |
| --- |
| document (ریشه)  ├── html  ├── head  │ ├── title  │ └── meta  └── body  ├── h1  ├── p  └── div |

انواع اصلی نودها در DOM:

|  |  |  |
| --- | --- | --- |
| نوع نود | مثال | توضیح |
| Element | P , div | تگ‌های HTML |
| Attribute | class="red" | ویژگی‌های تگ‌ها |
| Text | سلام دنیا | متن داخل تگ‌ها |
| Comment | `<!-- کامنت -->` | کامنت‌ها |

## نمایش درخت Html

درخت DOM چیست؟

DOM Tree یک نمایش سلسله مراتبی و درختی از ساختار HTML صفحه است که مرورگر آن را می‌سازد.

مثال ساده:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>صفحه من</title>  </head>  <body>  <h1>سلام دنیا</h1>  <p>این یک پاراگراف است</p>  </body>  </html> |

درخت DOM مربوطه:

|  |
| --- |
| document (ریشه)  │  ├── <!DOCTYPE html>  │  └── <html>  ├── <head>  │ └── <title>  │ └── "صفحه من" (متن)  │  └── <body>  ├── <h1>  │ └── "سلام دنیا" (متن)  │  └── <p>  └── "این یک پاراگراف است" (متن) |

ساختار کامل درخت DOM

مثال پیچیده‌تر:

|  |
| --- |
| <!DOCTYPE html>  <html lang="fa">  <head>  <meta charset="UTF-8">  <title>فروشگاه اینترنتی</title>  </head>  <body>  <header class="main-header">  <h1 id="site-title">فروشگاه من</h1>  <nav>  <ul>  <li><a href="home">خانه</a></li>  <li><a href="products">محصولات</a></li>  </ul>  </nav>  </header>    <main>  <article>  <h2>محصولات ویژه</h2>  <div class="product">  <img src="product1.jpg" alt="محصول ۱">  <p>توضیحات محصول</p>  </div>  </article>  </main>  </body>  </html> |

درخت DOM کامل:

|  |
| --- |
| document  │  ├── <!DOCTYPE html>  │  └── <html lang="fa">  ├── <head>  │ ├── <meta charset="UTF-8">  │ └── <title>  │ └── "فروشگاه اینترنتی"  │  └── <body>  ├── <header class="main-header">  │ ├── <h1 id="site-title">  │ │ └── "فروشگاه من"  │ │  │ └── <nav>  │ └── <ul>  │ ├── <li>  │ │ └── <a href="home">  │ │ └── "خانه"  │ │  │ └── <li>  │ └── <a href="products">  │ └── "محصولات"  │  └── <main>  └── <article>  ├── <h2>  │ └── "محصولات ویژه"  │  └── <div class="product">  ├── <img src="product1.jpg" alt="محصول ۱">  └── <p>  └── "توضیحات محصول" |

انواع Node در درخت DOM

1. Element Node (گره عنصر)

|  |
| --- |
| <div>, <p>, <span>, <img> |

1. Text Node (گره متن)

|  |
| --- |
| "این یک متن است" |

1. Attribute Node (گره ویژگی)

|  |
| --- |
| class="header", id="title", src="image.jpg" |

1. Comment Node (گره توضیح)

|  |
| --- |
| <!-- این یک توضیح است --> |

نمایش گرافیکی درخت DOM

|  |
| --- |
| document  │  ┌──────┴──────┐  <!DOCTYPE> <html>  │  ┌──────┴──────┐  <head> <body>  │ │  ┌──────┴──────┐ ┌───┴───┐  <meta charset> <title> <header> <main>  │ │  ┌───┴───┐ │  <h1> <nav> <article>  │ │ │  "فروشگاه من" <ul> ┌─┴─┐  │ <h2> <div>  ┌─┴─┐ │ │  <li><li> │ ┌─┴─┐  │ │ │ │ │  <a> <a> "محصولات" <img> <p>  │ │ │ │  "خانه" "محصولات" │ "توضیحات"  │  "محصول ۱" (alt) |

## بررسی مهفوم Semantic

فرض کنید می‌خواهید یک کتاب بنویسید. به جای اینکه همه چیز را فقط «فصل» بنامید، از بخش‌های معناداری مانند «پیشگفتار»، «فهرست»، «فصل‌های اصلی»، «نتیجه‌گیری» و «نمایه» استفاده می‌کنید. تگ‌های معنایی در HTML دقیقاً همین کار را برای صفحات وب انجام می‌دهند. آن‌ها به مرورگرها، موتورهای جستجو مانند گوگل و توسعه‌دهندگان کمک می‌کنند تا معنای هر بخش از صفحه را به سرعت درک کنند، نه فقط اینکه آن بخش چگونه به نظر می‌رسد.

مزایای اصلی استفاده از این تگ‌ها عبارتند از:

سئوی بهتر: موتورهای جستجو محتوای صفحه شما را بهتر درک می‌کنند.

دسترسی‌پذیری: برای کاربرانی که از نرم‌افزارهای صفحه‌خوان استفاده می‌کنند، درک ساختار صفحه آسان‌تر می‌شود.

کد خوانا و قابل نگهداری: سازماندهی و فهم کد برای توسعه‌دهندگان، به ویژه در پروژه‌های بزرگ، ساده‌تر می‌شود.

جدول کامل تگ‌های سمانتیک:

|  |  |
| --- | --- |
| نام تگ | توضیح کاربرد |
| article | محتوای مستقل و خودکفا را تعریف می‌کند، مانند یک پست وبلاگ، مقاله خبری یا نظر کاربر. |
| aside | محتوایی را تعریف می‌کند که جدا از محتوای اصلی صفحه است، مانند یک نوار کناری (سایدبار). |
| details | جزئیات اضافی را مشخص می‌کند که کاربر می‌تواند به صورت اختیاری آن را باز یا بسته کند (یک ویجت تعاملی). |
| figcaption | یک عنوان برای عنصر `<figure>` تعریف می‌کند (توضیح برای عکس یا نمودار). |
| figure | محتوای خود-contained مانند عکس، نمودار یا قطعه کد را مشخص می‌کند. |
| footer | یک پاورقی برای سند یا یک بخش خاص تعریف می‌کند. معمولاً شامل اطلاعات تماس، حق تکثیر و لینک‌های مرتبط است. |
| header | یک سربرگ برای سند یا یک بخش خاص تعریف می‌کند. اغلب شامل لوگو، عنوان و منوی می‌شود. |
| main | محتوای اصلی و منحصربه‌فرد سند را مشخص می‌کند. این تگ باید در هر صفحه فقط یک بار استفاده شود |
| Mark | متنی را که باید هایلایت یا علامت‌گذاری شود، تعریف می‌کند |
| nav | بخشی از صفحه که حاوی لینک‌های ناوبری اصلی است را تعریف می‌کند، مانند منوی سایت. |
| section | یک بخش عمومی در یک سند را تعریف می‌کند. معمولاً برای گروه‌بندی محتوای مرتبط حول یک موضوع واحد استفاده می‌شود. |
| summary | یک عنوان قابل مشاهده برای عنصر `<details>` تعریف می‌کند. این اولین چیزی است که کاربر می‌بیند و با کلیک روی آن، جزئیات باز یا بسته می‌شود. |
| time | یک زمان (یا تاریخ) را به صورت قابل خواندن توسط ماشین تعریف می‌کند. |

چگونه از تگ‌های معنایی استفاده کنیم؟

برای شروع، سعی کنید در پروژه بعدی خود به جای تکیه بر تگ‌های عمومی مانند `<div>` برای همه چیز، از تگ‌های معنایی مناسب استفاده کنید. برای مثال، ساختار اصلی صفحه خود را با `header، main` و `<footer>` بچینید و منوها را داخل `<nav>` قرار دهید.

این کار در ابتدا ممکن است کمی تمرین نیاز داشته باشد، اما به مرور کدهای شما حرفه‌ای‌تر و سازمان‌یافته‌تر خواهند شد.

## معرفی تگ p و display آن

## 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!

# Async در جاوا اسکریپت

# Defer در جاوا اسکریپت