### آموزش وب دزاین

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

## 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/index.html)

### 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](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (Functional Programming)

### 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)); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (ترکیب پارادایم‌ها)

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 بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (مثال ساده)

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']  ]); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  Map() )

#### 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' |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  get() )

* 1. بررسی وجود کلید قبل از دسترسی

|  |
| --- |
| // بررسی وجود کلید قبل از دسترسی  if (initializedMap.has('name')) {  const name = initializedMap.get('name');  console.log('Name found:', name); // خروجی: 'Name found: Alice'  } else {  console.log('Key not found');  } |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  get()  بررسی وجود کلید قبل از دسترسی )

1. دسترسی به همه مقادیر با تکرار (Iteration)

|  |
| --- |
| // 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}`);  }); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  get()  بررسی وجود کلید قبل از دسترسی )

1. دریافت همه کلیدها و مقادیر

|  |
| --- |
| // 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']] |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  get()  بررسی وجود کلید قبل از دسترسی )

##### مثال‌های کاربردی بیشتر

مثال ۱: مدیریت خطا برای کلیدهای ناموجود

|  |
| --- |
| 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) |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   get() مدیریت خطا برای کلیدهای ناموجود)

مثال ۲: استفاده در تابع

|  |
| --- |
| 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 } |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   get() استفاده در تابع)

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' |

##### 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 |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   map()  متدهای اصلی:)

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);  }); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()  Iteration)

#### 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)); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()  مدیریت کاربران)

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();  }  } |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()  Caching)

مثال 3: شمارش تکرار عناصر

|  |
| --- |
| function countOccurrences(array) {  const countMap = new Map();    for (const item of array) {  const currentCount = countMap.get(item) || 0;  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 } |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   شمارش تکرار عناصر)

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);  // تبدیل Map به Array  const arrayFromMap = Array.from(mapFromObj);  // یا  const arrayFromMap2 = [...mapFromMap];  // تبدیل Array به Map  const array = [['a', 1], ['b', 2], ['c', 3]];  const mapFromArray = new Map(array); |  |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   تبدیل بین ساختارها)

### 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' } |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   Object)

#### 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 |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   NaN)

#### 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 } |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   زنجیره‌ای)

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]] |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   توابع)

##### 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)); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   JSON)

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] |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) ( map()   مپ کردن آرایه )

1. مپ کردن آرایه با مربع عناصر:

|  |
| --- |
| 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] |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  map()    مربع عناصر)

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); |

[مثال بالا در دنیای وب](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (  map()    simple list to li)

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); |

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); |

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; |

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>  `; |

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>  `; |

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('container').appendChild(ulElement); |

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); |

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>`  });  console.log(html); |

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\*\* 🧩

```javascript

// 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;

});

```

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(); |

|  |
| --- |
| Html:  <ul id="ul"></ul>  Js:  const element = document.getElementById("ul"); // 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); |

## 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] |

[مقدار در سایت](webdesign/js/Object/index.html) (   filter() فیلتر کردن اعداد زوج)

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'] |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() فیلتر کردن بر اساس طول رشته(

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'}] |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() فیلتر کردن اشیاء در آرایه(

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] |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() استفاده از index در فیلتر کردن(

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] |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() حذف مقادیر falsy از آرایه (

### 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 |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() مثال مقایسه‌ای(

### 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 }] |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() side effects)

مثال صحیح جداسازی concerns

|  |
| --- |
| const users = [  { id: 1, name: 'Alice', active: true },  { id: 2, name: 'Bob', active: false },  { id: 3, name: 'Charlie', active: true }  ];  // 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 }] |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() concerns)

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 |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (   filter() اجرای مضاعف عملیات)

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;  }); |

[مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html)  filter() در زنجیره متدها)

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); |

1. [مقدار در سایت](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html)  filter() مشکلات در دیباگ کردن)

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 |

[مثال در وبسایت واقعی](webdesign/js/Object/index.html) (reduce جمع کردن اعداد آرایه)

1. محاسبه حاصل ضرب

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

[مثال در سایت واقعی](webdesign/js/Object/index.html) (  reduce   محاسبه حاصل ضرب)

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 |

[مشاهده در سایت](webdesign/js/Object/index.html) (  reduce   پیدا کردن بیشترین مقدار )

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"} |

[مشاهده در سایت واقعی](webdesign/js/Object/index.html) (   reduceتبدیل آرایه به شیء)

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} |

[مشاهده در سایت واقعی](webdesign/js/Object/index.html) (reduce    شمارش تکرار عناصر)

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] |

[مشاهده مثال واقعی](webdesign/js/Object/index.html) ( reduce     مسطح کردن آرایه‌های تو در تو)

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 }]  }  / |

[مثال واقعی](webdesign/js/Object/index.html) (    reduce    گروه‌بندی اشیا بر اساس ویژگی)

### 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 |

[در سایت واقعی](webdesign/js/Object/index.html) ( reduceمحاسبه میانگین)

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] |

[مثال در سایت](webdesign/js/Object/index.html) ( reduce حذف مقادیر تکراری)

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] |

[مشاهده تمرین در سایت](webdesign/js/Object/index.html) (  reduce    ترکیب چندین آرایه)

تمرین 4: تبدیل آرایه به رشته با فرمت خاص

|  |
| --- |
| 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" |

[مشاهده تمرین در سایت](webdesign/js/Object/index.html) ( reduce تبدیل آرایه به رشته با فرمت خاص)

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] |

مثال : [در سایت واقعی](webdesign/js/Object/index.html) :   Rest Parameters

### 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 |

[مثال واقعی در وبسایت](webdesign/js/Object/index.html)    Spread Operator

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 } |

[مثال وبسایت](webdesign/js/Object/index.html) (Spread Operator با آبچکت ها)

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 } |

[مثال در وبسایت واقعی](webdesign/js/Object/index.html) ( Spread Operator مثال‌های ترکیبی)

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] |

[مثال در سایت واقعی](webdesign/js/Object/index.html) (Rest Parameters, Spread Operator کاربردهای پیشرفته)

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" |

[مثال در سایت واقعی](file:///E:\personal%20job\sam\projects\classes\webdesign\js\Object\index.html) (join() )

## آموزش Destructuring Assignment

آموزش کامل Destructuring Assignment در جاوااسکریپت

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

```javascript

// 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

```

🔹 مقادیر پیش‌فرض

```javascript

const [a = 10, b = 20, c = 30] = [1, 2];

console.log(a); // 1

console.log(b); // 2

console.log(c); // 30 (پیش‌فرض)

```

🔹 رد کردن عناصر

```javascript

const [first, , third] = [1, 2, 3, 4];

console.log(first); // 1

console.log(third); // 3

```

🔹 جمع‌آوری باقی‌مانده‌ها

```javascript

const [first, second, ...rest] = [1, 2, 3, 4, 5];

console.log(first); // 1

console.log(second); // 2

console.log(rest); // [3, 4, 5]

```

🔹 تعویض مقادیر (Swap)

```javascript

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

```

---

🏗️ Destructuring آبجکت‌ها (Object Destructuring)

🔹 مبانی پایه

```javascript

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'

```

🔹 نام‌گذاری مجدد متغیرها

```javascript

const person = {

firstName: 'فاطمه',

lastName: 'احمدی'

};

const { firstName: fname, lastName: lname } = person;

console.log(fname); // 'فاطمه'

console.log(lname); // 'احمدی'

```

🔹 مقادیر پیش‌فرض

```javascript

const { name = 'ناشناس', age = 0, city = 'نامشخص' } = { name: 'محمد' };

console.log(name); // 'محمد'

console.log(age); // 0 (پیش‌فرض)

console.log(city); // 'نامشخص' (پیش‌فرض)

```

🔹 ترکیب نام‌گذاری مجدد و پیش‌فرض

```javascript

const user = {

username: 'user123'

};

const {

username: uname = 'مهمان',

role: userRole = 'user'

} = user;

console.log(uname); // 'user123'

console.log(userRole); // 'user' (پیش‌فرض)

```

🔹 Destructuring تو در تو

```javascript

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'

```

---

کاربردهای عملی Destructuring

1. \*\*پارامترهای تابع\*\*

```javascript

// 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

```

2. \*\*پاسخ‌های API\*\*

```javascript

// 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

```

3. \*\*پیکربندی و تنظیمات\*\*

```javascript

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 });

```

4. \*\*کار با ایونت‌ها\*\*

```javascript

// در React

function Button({ onClick, children, variant = 'primary', size = 'md' }) {

return (

<button

className={`btn btn-${variant} btn-${size}`}

onClick={onClick}

>

{children}

</button>

);

}

// در DOM events

element.addEventListener('click', ({ clientX, clientY, target }) => {

console.log(`کلیک در موقعیت: ${clientX}, ${clientY}`);

});

```

---

🔄 Destructuring با آرایه و آبجکت ترکیبی

🔹 آرایه‌های آبجکت‌ها

```javascript

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`);

}

```

🔹 آبجکت‌های پیچیده

```javascript

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'

```

---

⚠️ نکات مهم و خطاهای رایج

1. \*\*حفاظت در برابر undefined/null\*\*

```javascript

// خطا می‌دهد اگر person null باشد

const { name, age } = person;

// روش ایمن

const { name, age } = person || {};

// یا با پیش‌فرض

const { name = 'ناشناس', age = 0 } = person || {};

```

2. \*\*Destructuring آرایه‌های کوتاه\*\*

```javascript

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

```

3. \*\*نام ویژگی‌های آبجکت\*\*

```javascript

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'

```

---

تمرین‌های عملی

تمرین 1: تبدیل مختصات

```javascript

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 }

```

تمرین 2: پردازش کاربر

```javascript

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}`);

```

تمرین 3: مدیریت خطا

```javascript

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'

```

---

📚 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! 🚀