



JavaScript

به زبان ساده

تقدیم به همه جویندگان علم

این اثر رایگان بوده و هرگونه استفاده تجاری از آن پیگرد قانونی دارد.
استفاده از مطالب آن، بدون ذکر منبع، غیراخلاقی و غیرقانونی است.

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com

5.....	JavaScript چیست.....
5.....	ساخت یک برنامه ساده JavaScript.....
9.....	توضیحات.....
10.....	کاراکترهای کنترلی.....
12.....	متغیر.....
14.....	انواع داده.....
15.....	استفاده از متغیرها.....
18.....	ثابت ها.....
19.....	تبدیل انواع داده.....
22.....	عبارات و عملگرها.....
22.....	عملگرهای ریاضی.....
25.....	عملگرهای تخصیصی (جایگزینی).....
26.....	عملگرهای مقایسه ای.....
28.....	عملگرهای منطقی.....
30.....	عملگرهای بیتی.....
35.....	تقدم عملگرها.....
37.....	گرفتن ورودی از کاربر.....
40.....	ساختارهای تصمیم.....
41.....	دستور if.....
44.....	دستور if...else.....
45.....	عملگر شرطی.....
46.....	دستور if چندگانه.....
48.....	دستور if تو در تو.....
50.....	استفاده از عملگرهای منطقی.....
52.....	دستور Switch.....
55.....	تکرار.....
55.....	حلقه While.....
57.....	حلقه do while.....
58.....	حلقه for.....
59.....	حلقه های تو در تو (Nested Loops).....

61.....	خارج شدن از حلقه با استفاده از break و continue.....
62.....	آرایه
64.....	حلقه for...of.....
65.....	آرایه های چند بعدی
70.....	تابع
71.....	مقدار برگشتی از یک تابع
73.....	پارامترها و آرگومان ها
74.....	پارامترهای اختیاری.....
75.....	نامیدن آرگومان ها
77.....	Rest parameters
78.....	محدوده متغیر
79.....	Arrow Function
80.....	توابع بی نام و توابع خود فراخوان
82.....	برنامه نویسی شیء گرا (Object Oriented Programming).....
82.....	کلاس
84.....	سازنده
87.....	سطح دسترسی
88.....	کپسوله سازی
89.....	خواص (Properties)
93.....	وراثت
94.....	متد super () و کلمه کلیدی super
96.....	override
97.....	عملگر instanceof
98.....	اعضای Static
98.....	مدیریت استثناءها و خطایابی
99.....	دستورات try و catch
101	استفاده از بلوک finally

چيست JavaScript

جاوااسکریپت (JavaScript) یک از زبان برنامه نویسی شیء گرا و پرترفدار وب می‌باشد. این زبان را در ابتدا شخصی به نام Brendan Eich (برندان ایچ) در شرکت Netscape با نام Mocha طراحی نمود. این نام بعداً به LiveScript و نهایتاً به جاوااسکریپت تغییر یافت. این تغییر نام تقریباً با افزوده شدن پشتیبانی از جاوا در مرورگر وب Netscape Navigator همزمانی دارد.

اولین نسخه جاوااسکریپت در نسخه B32.0 این مرورگر در دسامبر 1995 معرفی و عرضه شد. این نام گذاری منجر به سردرگمی‌های زیادی شده و این ابهام را ایجاد می‌کند که جاوااسکریپت با جاوا مرتبط است در حالی که این طور نیست. عده زیادی این کار را یک ترفند تجاری برای به دست آوردن بخشی از بازار جاوا که در آن موقع زبان جدید مطرح برای برنامه‌نویسی تحت وب بود می‌دانند.

JavaScript به صورت «جاوااسکریپت» خوانده می‌شود، ولی در فارسی به صورت «جاوااسکریپت» ترجمه می‌شود و اگر به صورت «جاوا اسکریپت» ترجمه شود اشتباه است چون دو کلمه جدا از هم نیست و اگر به صورت دو کلمه جدا نوشته شود خطاهای نگارشی ایجاد می‌شود، به طور مثال ممکن است کلمه جاوا در انتهای خط و کلمه اسکریپت در ابتدای خط بعدی نوشته شود.

علیرغم اشتباه عمومی، زبان جاوااسکریپت با زبان جاوا ارتباطی ندارد، اگر چه ساختار این زبان به سی پلاس پلاس (C++) و جاوا شباهت دارد که این امر برای یادگیری آسان در نظر گرفته شده است.

ساخت یک برنامه ساده JavaScript

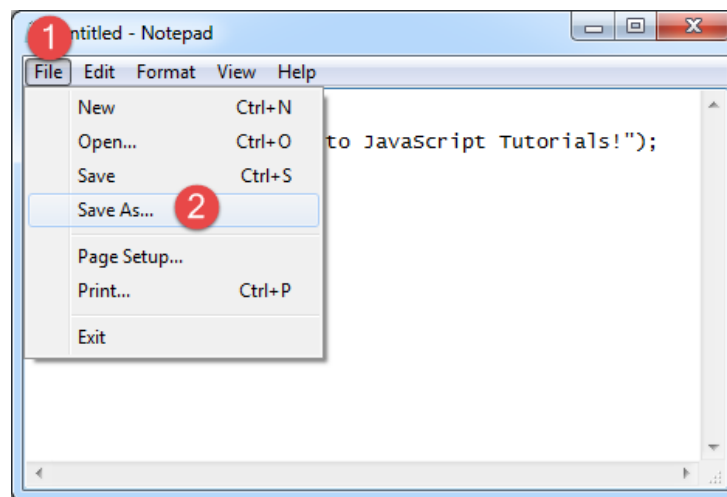
اجازه بدهید یک برنامه بسیار ساده به زبان جاوااسکریپت بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده جاوااسکریپت را توضیح دهم. برای اجرای کدهای جاوااسکریپت، هیچ ابزار خاصی نیاز نیست. در واقع بر خلاف زبان‌های دیگر که نیاز به یک کامپایلر برای اجرای کدها دارند، جاوااسکریپت روی مرورگرهای اینترنت اجرا می‌شود. برای نوشتن کدهای این زبان می‌توانید از یک ویرایشگر متن ساده، مانند NotePad پیش فرض ویندوز استفاده کنید؛ اما برای راحتی کار توصیه می‌کنیم از IDE (محیط‌ها و نرم‌افزارهای کدنویسی و توسعه) مناسب استفاده کنید. پیشنهاد ما به شما، Visual Studio Code میکروسافت یا نرم افزار NotePad++ است. ما فرض را بر راحت ترین حالت ممکن می گذاریم. به منوی Start رفته و برنامه NotePad ویندوز را باز کرده و کدهای زیر را در داخل آن بنویسید:

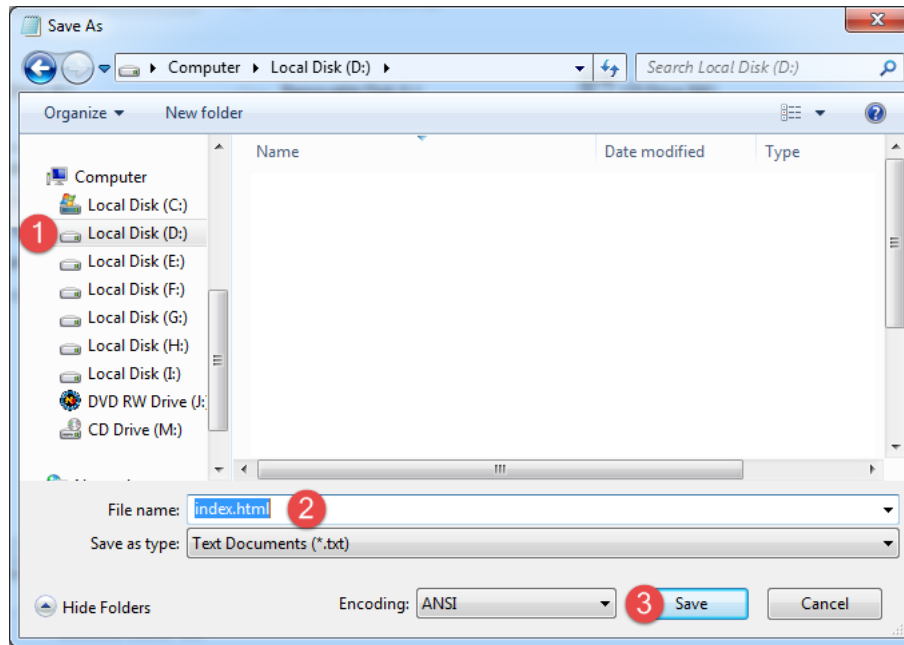
```
<script>
    console.log("Welcome to JavaScript Tutorials!");
</script>
```

```

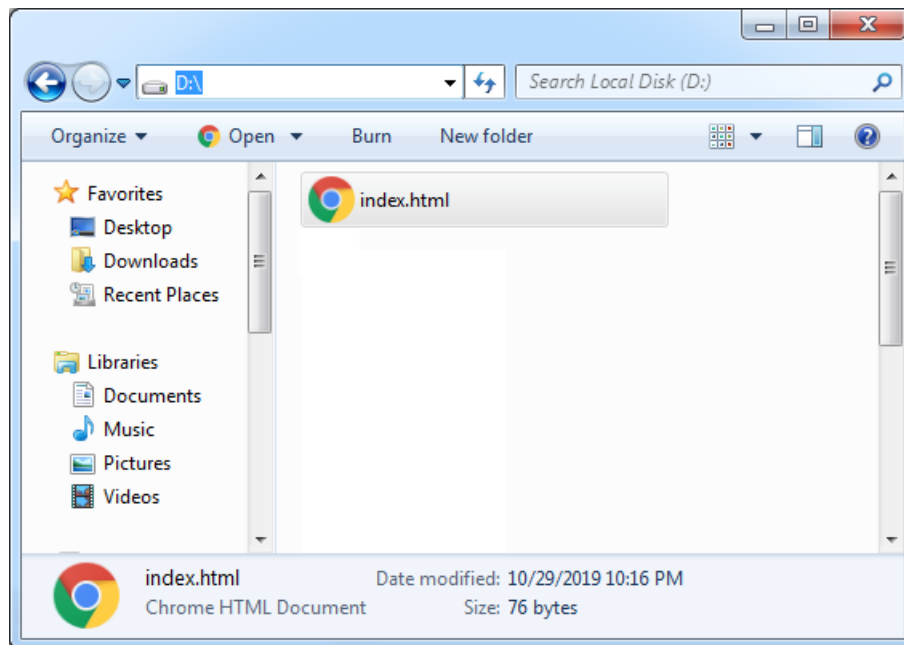
<script>
  console.log("welcome to Javascript Tutoria!");
</script>
    
```

حال برنامه بالا را با نام index.html در درایو D به صورت زیر ذخیره کنید:

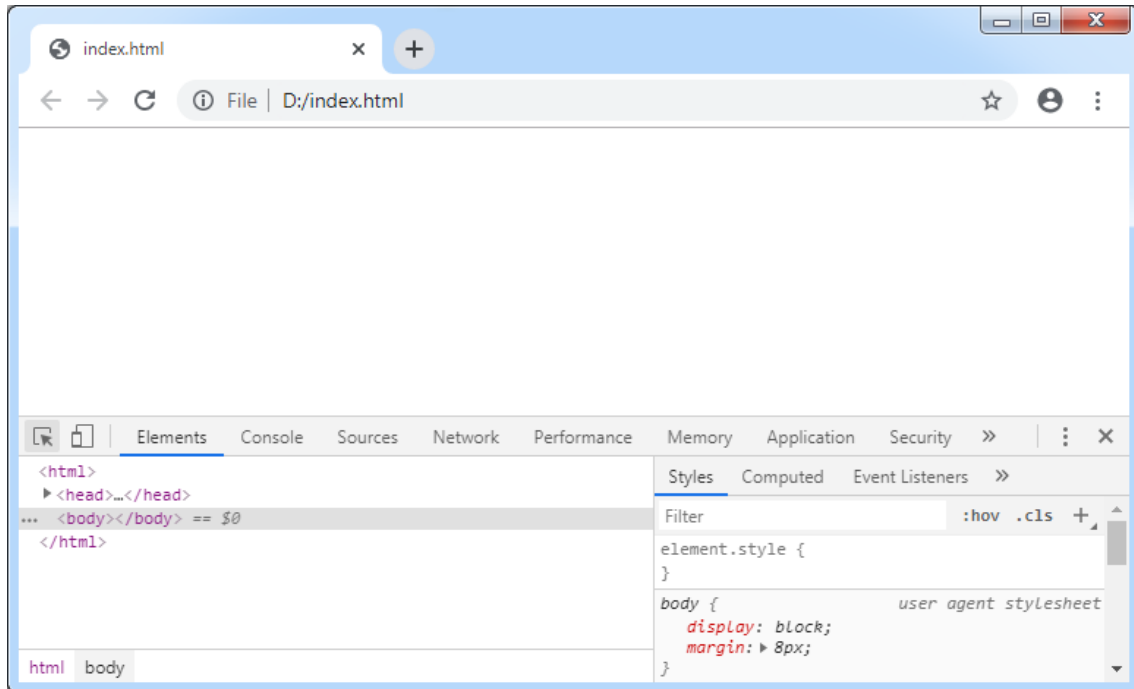




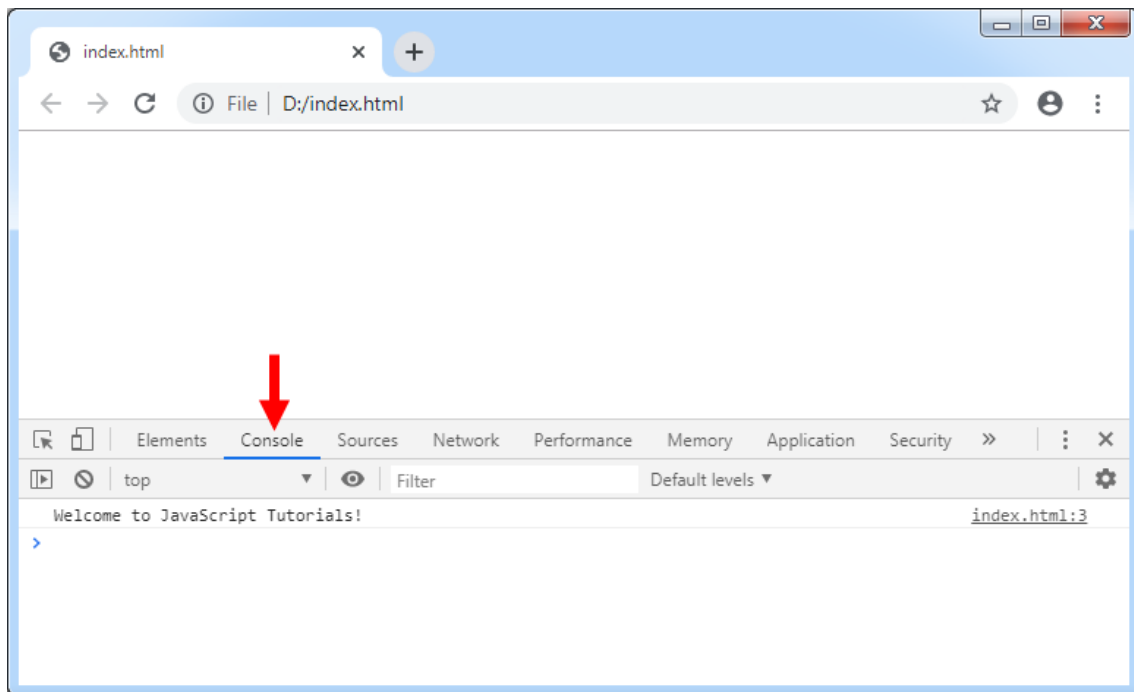
سپس به درایو D رفته و بر روی فایل index.html دو بار کلیک کنید تا به وسیله مرورگرتان اجرا شود:



اگر از یکی از مرورگرهای Firefox، Opera و یا Chrome استفاده می کنید، کافست بعد از باز شدن مرورگر، بر روی دکمه های ترکیبی Ctrl+Shift+C بزنید تا پنجره ای به صورت زیر باز شود:



در این پنجره به سربرگ Console رفته تا نتیجه اجرای برنامه را ببینید :



مثال بالا سادهترین برنامه‌ای است که شما می‌توانید در Javascript بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است.

توصیه می‌کنیم که برای اجرای کدهای Javascript از آخرین نسخه مرورگر Chrome استفاده کنید، چون از تمام قابلیت‌ها و امکانات نسخه نهایی جاوااسکریپت پشتیبانی می‌کند.

هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. Javascript دارای توابع از پیش تعریف شده‌ای است که هر کدام برای مقاصد خاصی به کار می‌روند. هر چند که در آینده در مورد توابع بیشتر توضیح می‌دهیم، ولی در همین حد به توضیح تابع `console.log()` می‌کنیم که توابع مجموعه‌ای از کدها هستند که دارای یک نام بوده و در جلوی نام آنها علامت `()` قرار می‌گیرد. یکی از این توابع، تابع `console.log()` است. از تابع `console.log()` برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است، که به وسیله دابل کوتیشن `"` محصور شده است. مانند `console.log("Welcome to javascript Tutorials!")`.

یک کاراکتر می‌تواند یک حرف، عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از تابع `console.log()` است. توضیحات بیشتر در درس‌های آینده آمده است. Javascript فضاهای خالی را نادیده می‌گیرد. مثلاً از کد زیر اشکال نمی‌گیرد:

```
console.log(
  "Welcome to JavaScript Tutorials!");
```

همیشه به یاد داشته باشید که Javascript به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال `MAN` و `man` در Javascript با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
Console.log("Welcome to JavaScript Tutorials!");
console.LoG("Welcome to JavaScript Tutorials!");
CONSOLE.log("Welcome to JavaScript Tutorials!");
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است:

```
console.log("Welcome to JavaScript Tutorials!");
```

حال که با خصوصیات و ساختار اولیه Javascript آشنا شدید در درس‌های آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت.

توضیحات

وقتی که کدی تایپ می‌کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در Javascript (و بیشتر زبانهای برنامه نویسی) می‌توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط مفسر نادیده گرفته می‌شوند و به عنوان بخشی از کد محسوب نمی‌شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می‌خواهید در مورد یک کد خاص، توضیح بدهید، می‌توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می‌شود. در برنامه زیر نقش توضیحات نشان داده شده است :

```
// This line will print the message hello world
console.log("Hello World!");
```

```
Hello World!
```

در کد بالا، خط 2 یک توضیح درباره خط 3 است که به کاربر اعلام می‌کند که وظیفه خط 3 چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط 2 در خروجی نمایش داده نمی‌شود چون مفسر توضیحات را نادیده می‌گیرد. توضیحات بر دو نوعند :

توضیحات تک خطی

```
// single line comment
```

توضیحات چند خطی

```
/* multi
   Line
   comment */
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می‌روند. این توضیحات با علامت // شروع می‌شوند و هر نوشته‌ای که در سمت راست آنها قرار بگیرد جز توضیحات به حساب می‌آید. این نوع توضیحات معمولاً در بالا یا کنار کد قرار می‌گیرند. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /* شروع و با */ پایان می‌یابند. هر نوشته‌ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می‌شود.

کاراکترهای کنترلی

کاراکترهای کنترلی، کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آنها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد :

```
console.log("Hello\nWorld!");
```

```
Hello
World
```

مشاهده کردید که مفسر بعد از مواجهه با کاراکتر کنترل `\n` نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد :

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	<code>\f</code>	چاپ کوتیشن	<code>\'</code>
خط جدید	<code>\n</code>	چاپ دابل کوتیشن	<code>\"</code>
سر سطر رفتن	<code>\r</code>	چاپ بک اسلش	<code>\\</code>
حرکت به صورت افقی	<code>\t</code>	چاپ فضای خالی	<code>\0</code>
حرکت به صورت عمودی	<code>\v</code>	صدای بیپ	<code>\a</code>
چاپ کاراکتر یونیکد	<code>\u</code>	حرکت به عقب	<code>\b</code>

ما برای استفاده از کاراکترهای کنترلی، از بک اسلش (`\`) استفاده می‌کنیم. از آنجاییکه `\` معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش (`\`) باید از `\\` استفاده کنیم :

```
console.log("We can print a \\ by using the \\\\ escape sequence.");
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از `\\`، نشان دادن مسیر یک فایل در ویندوز است :

```
console.log("C:\\Program Files\\Some Directory\\SomeFile.txt");
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (") برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از `\"` استفاده می‌کنیم :

```
console.log("I said, \"Motivate yourself!\".");
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (') از \ ' استفاده می‌کنیم :

```
console.log("The programmer\'s heaven.");
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می‌شود :

```
console.log("Left\tRight");
```

```
Left Right
```

برای چاپ کاراکترهای یونیکد می‌توان از \u استفاده کرد. برای استفاده از \u، مقدار در مبنای 16 کاراکتر را درست بعد از علامت \u قرار می‌دهیم. برای مثال اگر بخواهیم علامت کپی رایت (©) را چاپ کنیم، باید بعد از علامت \u مقدار A900 را قرار دهیم مانند :

```
console.log("\u00A9");
```

```
©
```

برای مشاهده لیست مقادیر مبنای 16 برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید :

```
http://www.asciicl/htmlcodes.htm
```

اگر مفسر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می‌افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

متغیر

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

باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود یکی است.

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

برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد، مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد :

- نام متغیر باید با یکی از حروف الفبا (a-z or A-Z) یا علامت _ شروع شود.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند \$, ^, ?, # باشد.
- نمی‌توان از کلمات رزرو شده در جاوااسکریپت برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.
- اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در جاوااسکریپت دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نامهای myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. متغیر دارای نوع هست که نوع داده‌ای را که در خود ذخیره می‌کند را نشان می‌دهد. در درس بعد در مورد انواع داده‌ها در جاوااسکریپت توضیح می‌دهیم. لیست کلمات کلیدی جاوااسکریپت، که نباید از آنها در نامگذاری متغیرها استفاده کرد در زیر آمده است :

abstract	arguments	*await	boolean
break	byte	case	catch
char	*class	const	continue
debugger	default	delete	do

double	else	*enum	eval
*export	*extends	false	final
finally	float	for	function
goto	if	implements	*import
in	instanceof	int	interface
*let	long	native	new
null	package	private	protected
public	return	short	static
*super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

انواع داده

انواع داده هایی که در JavaScript وجود دارند عبارتند از :

داده	توضیح
عددی (Numeric)	شامل اعداد مثبت و منفی (اعشاری و صحیح) می باشد.
رشته ای (String)	به مجموعه ای از کاراکترها که بین دو علامت کوتیشن یا دابل کوتیشن قرار گرفته باشند، اطلاق می شود.
آرایه (Array)	مجموعه ای از آیتم ها هستند که بین دو علامت [] قرار گرفته و با علامت کاما (,) از هم جدا شده اند.

مجموعه ای از آیتم ها هستند که به صورت کلید و مقدار بوده، بین دو علامت {} قرار گرفته، و با علامت کاما (,) از هم جدا شده اند.	شیء (Object)
شامل دو مقدار true یا false می باشد.	boolean
یک مقدار خاص است که به معنای هیچ چیز، خالی و یا مقدار ناشناخته است.	Null
اگر متغیری تعریف شود و مقداردهی اولیه نشود آنگاه مقدار آن برابر با undefined خواهد بود	undefined

در مورد انواع داده های بالا و نحوه استفاده از آنها در متغیرها، در درس بعد توضیح می دهیم.

استفاده از متغیرها

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

```
var variableName = Value
```

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

```
1 var numericVar = 10
2 var boolVar = true
3 var StringVar = "Hello World!"
4 var arrayVar = [1, 5, 8]
5 var objectVar = { 'Name': 'jack', 'family': 'Scalia', 'Age': 7 }
6
7 console.log("numericVar = ", numericVar)
8 console.log("boolVar = ", boolVar)
9 console.log("StringVar = ", StringVar)
10 console.log("arrayVar = ", arrayVar)
11 console.log("objectVar = ", objectVar)
```

```
numericVar = 10
boolVar = true
StringVar = Hello World!
arrayVar = Array(3)
  0: 1
  1: 5
  2: 8
  length: 3
objectVar = Object
  Name: "jack"
  family: "Scalia"
  Age: 7
```

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

```
var arrayVar = [1, 5, 8]
```

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

```
var objectVar = { Key1: Value1, Key2: Value2, Key3: Value3 }
```

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

```
var identifier1 = identifier2 = ...identifierN = Value
```

به مثال زیر توجه کنید :

```
var num1 = num2 = num3 = num4 = num5 = 10
var message1 = message2 = message3 = "Hello World!"
```

```
console.log(num1)
console.log(num4)
console.log(message1)
console.log(message3)
```

```
10
10
Hello World!
Hello World!
```

دقت کنید که برای متغیرهای تعریف شده در حالت بالا یک خانه حافظه تخصیص داده می شود، یعنی مقدار 10 در حافظه ذخیره شده و متغیرهای num1 و num2 و num3 و num4 و num5 به آن خانه از حافظه اشاره می کنند. همچنین می توان چند متغیر را تعریف کرد و برای هر یک از آن ها مقدار جداگانه ای مشخص نمود :


```
var identifier1, identifier2, ...identifierN = Value1, Value2, ...ValueN
```

به مثال زیر توجه کنید :

```
var [num1, num2, message1] = [10, 12.5, "Hello World!"]
console.log(num1)
console.log(num2)
console.log(message1)
```

```
10
12.5
Hello World!
```

در کد بالا مقدار num1 برابر 10، num2 برابر 12.5 و message1 برابر Hello World! می باشد. در JavaScript، متغیرها هم باید تعریف و هم مقداری شوند. یعنی اگر متغیری را تعریف کرده و به آن مقداری را اختصاص ندهید و برنامه را اجرا کنید با خطا مواجه می شوید :

```
var number
console.log(number)
```

```
Uncaught ReferenceError: number is not defined
```

همانطور که در درس قبل هم اشاره کردیم، یک رشته در اصل یک مجموعه از کاراکترهاست که در داخل علامت "" یا '' قرار دارند. هر کدام از این کاراکترها دارای یک اندیس است که به وسیله آن اندیس قابل دسترسی هستند. اندیس کاراکترها در رشته از 0 شروع می شود. به رشته زیر توجه کنید :

```
var message = "Hello World!"
```

در رشته بالا اندیس کاراکتر 0 برابر 4 است. برای درک بهتر به شکل زیر توجه کنید :

```
H e l l o   W o r l d !
0 1 2 3 4 5 6 7 8 9 10 11
```

حال برای چاپ یک کاراکتر (مثلا w) از این رشته کفایت که به صورت زیر عمل کنیم :

```
var message = "Hello World! "
console.log(message[6])
```

```
w
```

همانطور که در کد بالا مشاهده می کنید کافیسیت که نام متغیر را نوشته، در جلوی آن یک جفت کروشه و در داخل کروشه ها اندیس آن کاراکتری را که می خواهیم چاپ شود را بنویسیم. چاپ مقدار با استفاده از اندیس در مورد آرایه هم صدق می کند :

```
var arrayVar = [1, 5, 8]
console.log(arrayVar[2])
```

```
8
```

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

```
var objectVar = { 'Name': 'jack', 'Family': 'Scalia', 'Age': 7 }
console.log(objectVar['Family'])
```

```
Scalia
```

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

```
var objectVar = { 1: 'Jack', '2': 'Scalia', 3: 7 }
console.log(objectVar['2'])
```

```
Scalia
```

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

```
var objectVar = { 1: 'Jack', '2': 'Scalia', 3: 7 }
console.log(objectVar[2])
```

```
Scalia
```

ثابت ها

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

```
const data_type identifier = initial_value;
```

مثال :

```
const NUMBER = 1;
NUMBER = 10; //Uncaught TypeError: Assignment to constant variable.
```

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

تبدیل انواع داده

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

- parseInt()
- parseFloat()

توجه کنید که حروف F و I باید به صورت حرف بزرگ نوشته شوند.

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

```
console.log(parseInt("25Number"));
```

```
25
```

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

```
console.log(parseFloat("25.5Number"));
console.log(parseFloat("25.45.2Number"));
```

```
25.5
25.45
```

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

- Boolean()
- Number()

String() •

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

```
var b1 = Boolean("");
var b2 = Boolean("String");
var b3 = Boolean(100);
var b4 = Boolean(null);
var b5 = Boolean(0);
var b6 = Boolean(new Object());

console.log(b1);
console.log(b2);
console.log(b3);
console.log(b4);
console.log(b5);
console.log(b6);
```

```
false
true
true
false
false
true
```

متد Number() کاری شبیه به متدهای parseInt() و parseFloat() انجام می‌دهد ولی تفاوت‌هایی هم با این دو متد دارد. اگر به یاد داشته باشید متدهای parseInt() و parseFloat() آرگومان دریافتی را فقط تا اولین کاراکتر بی ارزش بر می‌گردانند. مثلاً رشته "25.5.2" را به ترتیب به 25 و 25.5 تبدیل خواهند کرد. اما متد Number() مقدار NaN را می‌گرداند. زیرا این رشته از نظر متد Number() امکان تبدیل به یک عدد را ندارد. اگر رشته‌ای امکان تبدیل به یک عدد را داشته باشد متد Number() خود برای استفاده از یکی از توابع parseInt() و parseFloat() تصمیم می‌گیرد. مثال زیر حاصل اجرای تابع Number() برای انواع داده‌ها را نشان می‌دهد:

```
console.log(Number(false));
console.log(Number(true));
console.log(Number(undefined));
console.log(Number(null));
console.log(Number("5.5"));
console.log(Number("56"));
console.log(Number("5.6.7"));
console.log(Number(new Object()));
console.log(Number(100));
```

```
0
1
NaN
0
5.5
```

```
56  
NaN  
NaN  
100
```

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

```
console.log(String(null));
```

```
null
```

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

```
var value1 = 10.0;  
var value2 = "Hello World!";  
  
console.log(typeof (value1));  
console.log(typeof (value2));
```

```
number  
string
```

برای دریافت مطالب جدید به سایت w3-farsi.com مراجعه فرمایید

انتقادات و پیشنهادات خود را به ایمیل younes.ebrahimi.1391@gmail.com ارسال فرمایید

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید :

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

مثلاً $X+Y$: یک عبارت است که در آن X و Y عملوند و علامت $+$ عملگر به حساب می‌آیند.

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

اشاره کرد. سه نوع عملگر در جاوااسکریپت وجود دارد :

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

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

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه‌ای
- عملگرهای منطقی
- عملگرهای بیتی

عملگرهای ریاضی

JavaScript از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی JavaScript را نشان می‌دهد

:

عملگر	دسته	مثال	نتیجه
+	Binary	<code>var1 = var2 + var3;</code>	Var1 برابر است با حاصل جمع var2 و var3

var1 برابر است با حاصل تفریق var2 و var3	var1 = var2 - var3;	Binary	-
var1 برابر است با حاصلضرب var2 در var3	var1 = var2 * var3;	Binary	*
var1 برابر است با var2 به توان var3	var1 = var2 ** var3;	Binary	**
var1 برابر است با حاصل تقسیم var2 بر var3	var1 = var2 / var3;	Binary	/
var1 برابر است با باقیمانده تقسیم var2 و var3	var1 = var2 % var3;	Binary	%
var1 برابر است با مقدار var2	var1 = +var2;	Unary	+
var1 برابر است با مقدار var2 ضربدر -1	var1 = -var2;	Unary	-

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

عملگر	دسته	مثال	نتیجه
++	Unary	var1 = ++var2;	مقدار var1 برابر است با var2 بعلاوه 1
--	Unary	var1 = --var2;	مقدار var1 برابر است با var2 منهای 1
++	Unary	var1 = var2++;	مقدار var1 برابر است با var2 به متغیر var2 یک واحد اضافه می شود
--	Unary	var1 = var2--;	مقدار var1 برابر است با var2 از متغیر var2 یک واحد کم می شود

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

```
x = 0;
y = 1;

x = ++y;

console.log("x = ", x);
console.log("y = ", y);
```

```
x = 2
```

```
y = 2
```

```
x = 0;
y = 1;

x = --y;

console.log("x = ", x);
console.log("y = ", y);
```

```
x=0
y=0
```

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

```
x = 0;
y = 1;

x = y--;

console.log("x = ", x);
console.log("y = ", y);
```

```
x = 1
y = 0
```

```
x = 0;
y = 1;

x = y++;

console.log("x = ", x);
console.log("y = ", y);
```

```
x = 1
y = 2
```

همانطور که در دو مثال بالا مشاهده می کنید، درج عملگرهای ++ و -- بعد از عملوند y باعث می شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به آن اضافه شود. حال می توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در جاوااسکریپت را یاد بگیریم :

```
1 //Variable declarations
2 var num1, num2;
3 var msg1, msg2;
4
5 //Assign test values
6 num1 = 5;
```



```

7 num2 = 3;
8
9 //Demonstrate use of mathematical operators
10 console.log("The sum of", num1, "and", num2, "is", (num1 + num2));
11 console.log("The difference of", num1, "and", num2, "is", (num1, num2));
12 console.log("The product of", num1, "and", num2, "is", (num1 * num2));
13 console.log("The quotient of", num1, "and", num2, "is", (num1 / num2).toFixed(2));
14 console.log("The remainder of", num1, "divided by", num2, "is", (num1 % num2));
15 console.log("The result of", num1, "powered by", num2, "is", (num1 ** num2));
16
17 //Demonstrate concatenation on strings using the + operator
18 msg1 = "Hello ";
19 msg2 = "World!";
20 console.log(msg1 + msg2);

```

```

The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.67.
The remainder of 5 divided by 3 is 2
The result of 5 powered by 3 is 125
Hello World!

```

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

عملگرهای تخصیصی (جایگزینی)

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

عملگر	مثال	نتیجه
=	<code>var1 = var2;</code>	مقدار var1 برابر است با مقدار var2
+=	<code>var1 += var2;</code>	مقدار var1 برابر است با حاصل جمع var1 و var2
-=	<code>var1 -= var2;</code>	مقدار var1 برابر است با حاصل تفریق var1 و var2
*=	<code>var1 *= var2;</code>	مقدار var1 برابر است با حاصل ضرب var1 در var2

مقدار var1 برابر است با حاصل تقسیم var1 بر var2	var1 /= var2;	/=
مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2	var1 %= var2;	%=

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

```
var number;

console.log("Assigning 10 to number...");
number = 10;
console.log("Number = ", number);

console.log("Adding 10 to number...");
number += 10;
console.log("Number = ", number);

console.log("Subtracting 10 from number...");
number -= 10;

console.log("Number = ", number);
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

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

عملگرهای مقایسه‌ای

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

عملگر	مثال	نتیجه
==	var1 = var2 == var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت false است

var1 در صورتی true است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت false است	var1 = var2 != var3	!=
var1 در صورتی true است که مقدار var2 کوچکتر از var3 مقدار باشد در غیر اینصورت false است	var1 = var2 < var3	<
var1 در صورتی true است که مقدار var2 بزرگتر از مقدار var3 باشد در غیر اینصورت false است	var1 = var2 > var3	>
var1 در صورتی true است که مقدار var2 کوچکتر یا مساوی مقدار var3 باشد در غیر اینصورت false است	var1 = var2 <= var3	<=
var1 در صورتی true است که مقدار var2 بزرگتر یا مساوی مقدار var3 باشد در غیر اینصورت false است	var1 = var2 >= var3	>=

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد :

```
var num1 = 10;
var num2 = 5;

console.log(num1 + " == " + num2 + " :", num1 == num2);
console.log(num1 + " != " + num2 + " :", num1 != num2);
console.log(num1 + " < " + num2 + " :", num1 < num2);
console.log(num1 + " > " + num2 + " :", num1 > num2);
console.log(num1 + " <= " + num2 + " :", num1 <= num2);
console.log(num1 + " >= " + num2 + " :", num1 >= num2);
```

```
10 == 5 : false
10 != 5 : true
10 < 5 : false
10 > 5 : true
10 <= 5 : false
10 >= 5 : true
```

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

عملگرهای منطقی

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

عملگر	نام	مثال
&&	منطقی AND	var1 = var2 && var3;
	منطقی OR	var1 = var2 var3;
!	منطقی NOT	var1 = !var1;

عملگر منطقی (&&) AND

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

X	Y	X && Y
true	true	true
true	false	false
false	true	false
false	false	false

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

```
result = (age > 18) && (salary < 1000);
```

عملگر AND زمانی کارآمد است که ما با محدود خاصی از اعداد سر و کار داریم. مثلا عبارت $100 \leq x \leq 10$ بدین معنی است که x می تواند مقداری شامل اعداد 10 تا 100 را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100);
```

عملگر منطقی (||) OR

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

X	Y	X Y
true	true	true
true	false	true
false	true	true
false	false	false

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

```
isPassed = (finalGrade >= 75) || (finalExam == 100);
```

عملگر منطقی (!) NOT

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

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که (age سن) بزرگتر یا مساوی 18 نباشد.

```
isMinor = !(age >= 18);
```


تقدم عملگرها

تقدم عملگرها مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند ابتدا کدام عملگر اثرش را اعمال کند. عملگرها در جاوااسکریپت در محاسبات دارای حق تقدم هستند. به عنوان مثال :

```
var number = 1 + 2 * 3 / 1;
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه 9 خواهد شد ($1+2=3$ سپس $3 \times 3=9$ و در آخر $9/1=9$). اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد 2 ضربدر 3 و سپس نتیجه آنها تقسیم بر 1 می‌شود که نتیجه 6 به دست می‌آید. در آخر عدد 6 با 1 جمع می‌شود و عدد 7 حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای جاوااسکریپت آمده است :

اولویت	شرکت پذیری	عملگر
بیشترین اولویت	از چپ به راست	()
1	از چپ به راست	[]
1	از چپ به راست	.
2	از راست به چپ	++
2	از راست به چپ	-
2	از راست به چپ	-
2	از راست به چپ	!
2	از راست به چپ	~
2	از راست به چپ	delete
2	از راست به چپ	new
2	از راست به چپ	typeof
2	از راست به چپ	void
3	از چپ به راست	/

*	از چپ به راست	3
%	از چپ به راست	3
+	از چپ به راست	4
+	از چپ به راست	4
-	از چپ به راست	4
>>	از چپ به راست	5
<<	از چپ به راست	5
>, >=	از چپ به راست	6
<, <=	از چپ به راست	6
==	از چپ به راست	7
!=	از چپ به راست	7
===	از چپ به راست	7
!==	از چپ به راست	7
&	از چپ به راست	8
^	از چپ به راست	9
	از چپ به راست	10
&&	از چپ به راست	11
	از چپ به راست	12
?:	از چپ به راست	13
=	از راست به چپ	14
*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, =	از راست به چپ	14
,	از چپ به راست	کمترین اولویت

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند. به این نکته توجه کنید که تقدم عملگرها ++ و - به مکان قرارگیری آنها بستگی دارد (در سمت چپ یا راست عملوند باشند). به عنوان مثال :

```
var number = 3;
var number1 = 3 + ++number; //results to 7
var number2 = 3 + number++; //results to 7
```

در هر دو عبارت بالا به دلیل تقدم بالاتر عملگر ++ نسبت به عملگر +، ابتدا به مقدار number یک واحد اضافه شده و 4 می‌شود و سپس مقدار جدید با عدد 3 جمع می‌شود و در نهایت عدد 7 به دست می‌آید. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم :

```
var number = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ) );
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته‌ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید. در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد یعنی مقدار 6 ضربدر 7 شده و سپس از 5 کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال :

```
var number = 3 * 2 + 8 / 4;
```

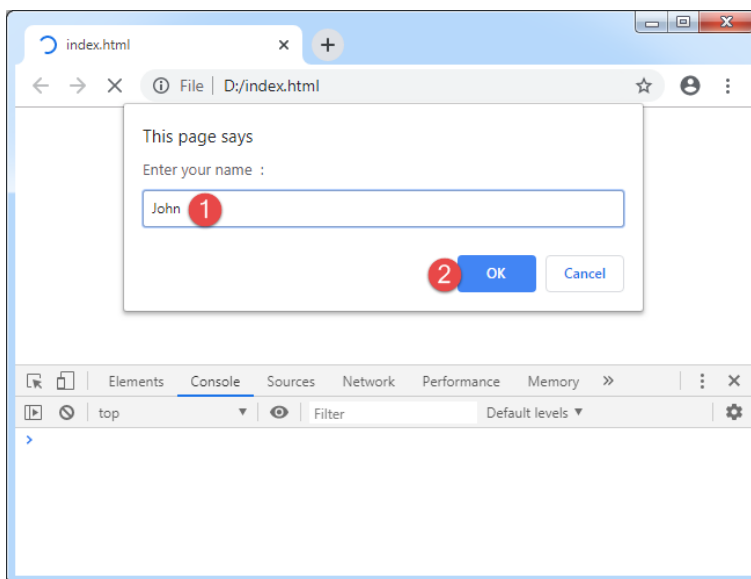
هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آنها را در محاسبات تأثیر دهید. یعنی ابتدا 3 را ضربدر 2 می‌کنید و سپس عدد 8 را بر 4 تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید .

گرفتن ورودی از کاربر

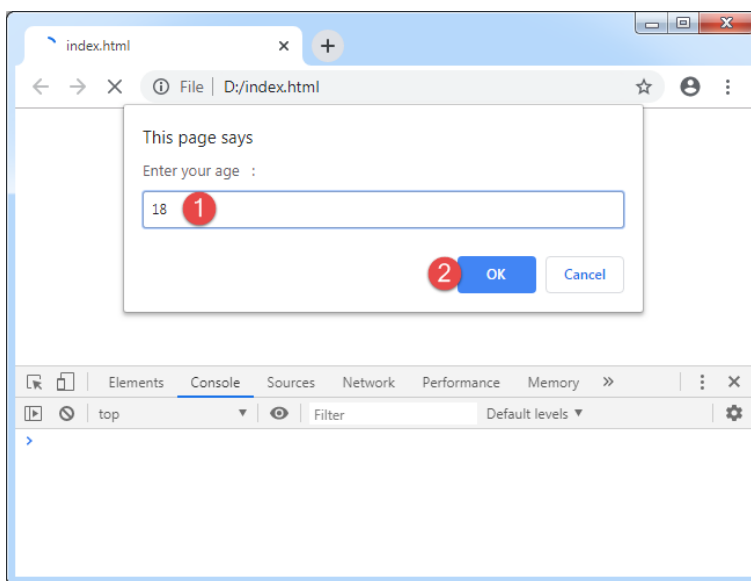
JavaScript متد `prompt()` را برای گرفتن ورودی از کاربر، در اختیار شما قرار می‌دهد. این متد باعث ظاهر شدن یک پنجره `Popup` می‌شود. این پنجره یک جعبه متن در اختیار شما می‌گذارد و تمام کاراکترهایی را که شما در محیط برنامه نویسی تایپ می‌کنید تا زمانی که دکمه `Enter` را می‌زنید، می‌خواند. به برنامه زیر توجه کنید :

```
1 var name = prompt("Enter your name : ");
2 var age = prompt("Enter your age : ");
3 var height = prompt("Enter your height: ");
4
5 console.log("Your name is : ", name);
6 console.log("Your age is : ", age);
7 console.log("Your height is : ", height);
```

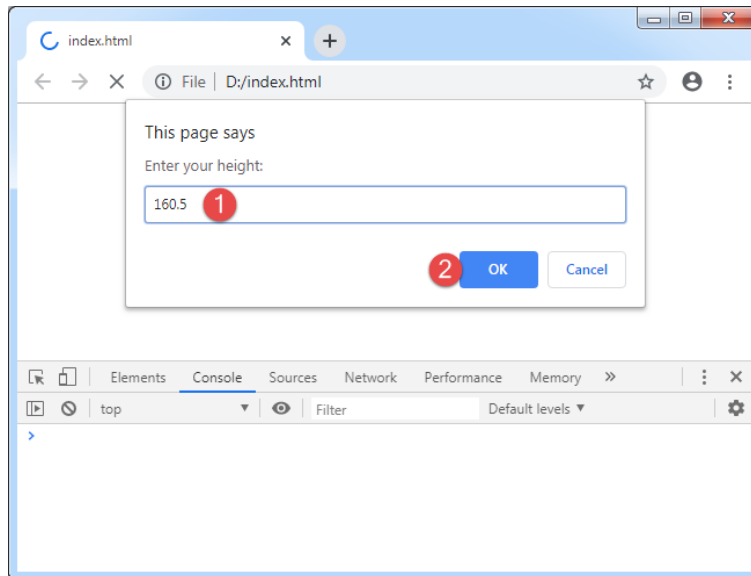
ابتدا 3 متغیر را برای ذخیره داده در برنامه تعریف می‌کنیم (خطوط 1 و 2 و 3). با اجرای برنامه پنجره Popup ظاهر می‌شود و از کاربر می‌خواهد که نام خود را وارد کند (خط 1):



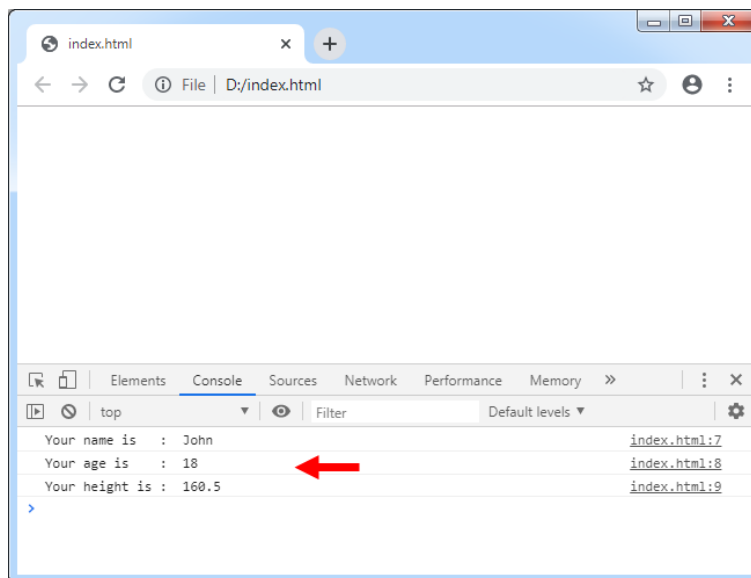
با زدن دکمه OK، دوباره پنجره ظاهر شده و از ما سن را سؤال می‌کند (خط 3):



اگر بر روی دکمه OK کلیک کنیم، دوباره پنجره ظاهر شده و از شما می‌خواهد که قدتان را وارد کنید:



بعد از زدن دکمه OK، نتیجه نهایی اجرای کد در قسمت Console نمایش داده می شود:



البته JavaScript دارای منتهای دیگری برای گرفتن ورودی از کاربر نیز هست، که در درس های آینده در مورد آنها توضیح می دهیم.



از سایر کتاب های یونس ابراهیمی در لینک های زیر دیدن فرمایید

<https://bit.ly/2Is8a0t>

www.w3-farsi.com/product

ساختارهای تصمیم

تقریباً همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. Javascript راه های مختلفی برای رفع این نوع مشکلات ارائه می دهد. در این بخش با مطالب زیر آشنا خواهید شد :

- دستور if
- دستور if...else
- عملگر سه تایی
- دستور if چندگانه
- دستور if تو در تو
- عملگرهای منطقی
- دستور switch

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است کد معینی را انجام بده. ساختار دستور if به صورت زیر است :

```
if (condition)
{
    code to execute;
}
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از 10 و Goodbye World را اگر مقدار number از 10 بزرگ‌تر باشد در صفحه نمایش می‌دهد .

```
1 //Declare a variable and set it a value less than 10
2 var number = 5;
3
4 //If the value of number is less than 10
5 if (number < 10)
6     console.log("Hello World.");
7
8 //Change the value of a number to a value which
9 // is greater than 10
10 number = 15;
11
12 //If the value of number is greater than 10
13 if (number > 10)
14     console.log("Goodbye World.");
```

```
Hello World.
Goodbye World.
```

در خط 2 یک متغیر با نام number تعریف و مقدار 5 به آن اختصاص داده شده است. وقتی به اولین دستور if در خط 5 می‌رسیم برنامه تشخیص می‌دهد که مقدار number از 10 کمتر است یعنی 5 کوچک‌تر از 10 است .

منطقی است که نتیجه مقایسه درست می‌باشد بنابراین دستور if دستور را اجرا می‌کند (خط 6) و پیغام Hello World چاپ می‌شود. حال مقدار number را به 15 تغییر می‌دهیم (خط 10). وقتی به دومین دستور if در خط 13 می‌رسیم برنامه مقدار number را با 10 مقایسه می‌کند و چون مقدار number یعنی 15 از 10 بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط 14). به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت :

```
if (number > 10) console.log("Goodbye World.");
```

شما می‌توانید چندین دستور را در داخل دستور `if` بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدنه دستور `if` هستند. نحوه تعریف چند دستور در داخل بدنه `if` به صورت زیر است :

```
if (condition)
{
    statement1;
    statement2;
    .
    .
    .
    statementN;
}
```

این هم یک مثال ساده :

```
var x = 11;

if (x > 10)
{
    console.log("x is greater than 10.");
    console.log("This is still part of the if statement.");
}
```

در مثال بالا اگر مقدار `x` از 10 بزرگتر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار `x` از 10 بزرگتر نباشد مانند کد زیر :

```
var x = 5;

if (x > 10)
    console.log("x is greater than 10.");
    console.log("This is still part of the if statement.");
```

کد بالا در صورتی بهتر خوانده می‌شود که بین دستورات فاصله بگذاریم.

```
if (x > 10)
    console.log("x is greater than 10.");

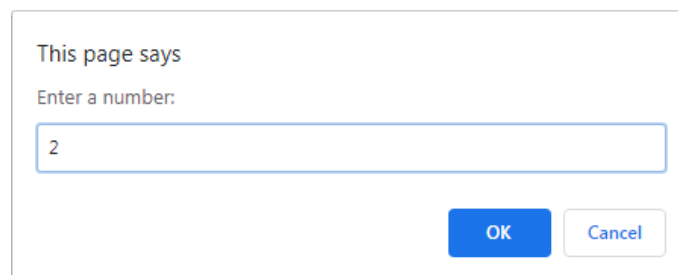
    console.log("This is still part of the if statement.");
```

می‌بیند که دستور دوم یعنی خط آخر در مثال بالا، جز دستور `if` نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار `x` از 10 کوچکتر است پس خط (Really?) `This is still part of the if statement.` چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه `if` داشتید برای آن یک آکولاد بگذارید. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند. مثالی دیگر در مورد دستور `if`:

```
1 var firstNumber;
2 var secondNumb ;
3
```

```
4 firstNumber = parseInt(prompt("Enter a number: "));
5 secondNumber = parseInt(prompt("Enter another number: "));
6
7 if (firstNumber == secondNumber)
8 {
9     console.log(firstNumber + "==" + secondNumber);
10 }
11 if (firstNumber != secondNumber)
12 {
13     console.log(firstNumber + "!=" + secondNumber);
14 }
15 if (firstNumber < secondNumber)
16 {
17     console.log(firstNumber + "< " + secondNumber);
18 }
19 if (firstNumber > secondNumber)
20 {
21     console.log(firstNumber + "> " + secondNumber);
22 }
23 if (firstNumber <= secondNumber)
24 {
25     console.log(firstNumber + "<=" + secondNumber);
26 }
27 if (firstNumber >= secondNumber)
28 {
29     console.log(firstNumber + ">=" + secondNumber);
30 }
```

ما از عملگرهای مقایسه‌ای در دستور `if` استفاده کرده‌ایم. ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر می‌گیریم (خطوط 4 و 5). توجه کنید که چون ورودی‌های گرفته شده توسط متد `prompt()` از نوع رشته است، پس باید آنها را با استفاده از متد `parseInt()` به نوع عددی تبدیل کنیم تا کار مقایسه راحت‌تر انجام شود. با اجرای برنامه پنجره‌ای به صورت زیر برای شما نمایش داده می‌شود:



The image shows a standard browser dialog box. At the top, it says "This page says". Below that is the prompt "Enter a number:". There is a text input field containing the number "2". At the bottom right, there are two buttons: "OK" and "Cancel".

در پنجره باز شده عدد 2 را نوشته و سپس دکمه OK را فشار می‌دهیم. با زدن دکمه OK، پنجره‌ای دیگر به صورت زیر نمایش داده می‌شود:

در این پنجره نیز بعد از وارد کردن عدد 5 و زدن دکمه OK، نتیجه به صورت زیر در محیط کنسول نمایش داده می شود:

```
2 != 5
2 < 5
2 <= 5
```

به این نکته توجه داشته باشید که شرطها مقادیر بولی هستند، بنابراین شما می‌توانید نتیجه یک عبارت را در داخل یک متغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور if استفاده کنید. اگر مقدار year برابر 2000 باشد سپس حاصل عبارت در متغیر isNewMillenium ذخیره می‌شود. می‌توان از متغیر برای تشخیص کد اجرایی بدنه دستور if استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

```
var isNewMillenium = 2000;
Boolean(isNewMillenium);
if (isNewMillenium)
{
    console.log("Happy New Millenium!");
}
```

دستور if...else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود یعنی اگر حالتی برقرار بود کار خاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود دستور دیگر اجرا شود باید از دستور if else استفاده کنید. ساختار دستور if else در زیر آمده است :

```
if (condition)
{
    code to execute if condition is true;
}
else
{
    code to execute if condition is false;
}
```

از کلمه کلیدی `else` نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با `if` به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه `if` و بدنه `else` دارید استفاده از آکولاد اختیاری است. کد داخل بلوک `else` فقط در صورتی اجرا می‌شود که شرط داخل دستور `if` نادرست باشد. در زیر نحوه استفاده از دستور `if...else` آمده است.

```

1  var number = 5;
2
3  //Test the condition
4  if (number < 10)
5  {
6      console.log("The number is less than 10.");
7  }
8  else
9  {
10     console.log("The number is either greater than or equal to 10.");
11 }
12
13 //Modify value of number
14 number = 15;
15
16 //Repeat the test to yield a different result
17 if (number < 10)
18 {
19     console.log("The number is less than 10.");
20 }
21 else
22 {
23     console.log("The number is either greater than or equal to 10.");
24 }

```

```

The number is less than 10.
The number is either greater than or equal to 10.

```

در خط 1 یک متغیر به نام `number` تعریف کرده‌ایم و در خط 4 تست می‌کنیم که آیا مقدار متغیر `number` از 10 کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک `if` اجرا می‌شود (خط 6) و اگر مقدار `number` را تغییر دهیم و به مقداری بزرگتر از 10 تغییر دهیم (خط 14)، شرط نادرست می‌شود (خط 17) و کد داخل بلوک `else` اجرا می‌شود (خط 23). مانند بلوک `if` نباید به آخر کلمه کلیدی `else` سیمیکولن اضافه شود.

عملگر شرطی

عملگر شرطی (`?:`) در جاوااسکریپت مانند دستور شرطی `if...else` عمل می‌کند. در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی جاوااسکریپت است که نیاز به سه عملوند دارد، شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد. اجازه بدهید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```

1 var pet1 = "puppy";
2 var pet2 = "kitten";
3 var type1;
4 var type2;
5
6 type1 = (pet1 == "puppy" ) ? "dog" : "cat";
7 type2 = (pet2 == "kitten") ? "cat" : "dog";
8
9 console.log(type1);
10 console.log(type2);

```

```

dog
cat

```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می‌دهد. خط 6 به صورت زیر ترجمه می‌شود: اگر مقدار pet1 برابر با puppy سپس مقدار dog را در type1 قرار بده در غیر این صورت مقدار cat را type1 قرار بده. خط 7 به صورت زیر ترجمه می‌شود: اگر مقدار pet2 برابر با kitten سپس مقدار cat را در type2 قرار بده در غیر این صورت مقدار dog. حال برنامه بالا را با استفاده از دستور if else می‌نویسیم:

```

if (pet1 == "puppy")
  type1 = "dog";
else
  type1 = "cat";

```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید :

```

if (condition)
{
  code to execute;
}
else
{
  if (condition)
  {
    code to execute;
  }
  else
  {
    if (condition)
    {
      code to execute;
    }
    else
    {
      code to execute;
    }
  }
}

```

```

    }
  }
}

```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک `else` بنویسید. می‌توانید کد بالا را ساده‌تر کنید :

```

if (condition)
{
    code to execute;
}
else if (condition)
{
    code to execute;
}
else if (condition)
{
    code to execute;
}
else
{
    code to execute;
}

```

حال که نحوه استفاده از دستور `else if` را یاد گرفتید باید بدانید که مانند `else if`، `else if` نیز به دستور `if` وابسته است. دستور `else if` وقتی اجرا می‌شود که اولین دستور `if` اشتباه باشد حال اگر `else if` اشتباه باشد دستور `else if` بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور `else` اجرا می‌شود. برنامه زیر نحوه استفاده از دستور `if else` را نشان می‌دهد :

```

var choice;

console.log("What's your favorite color?");
console.log("[1] Black");
console.log("[2] White");
console.log("[3] Blue");
console.log("[4] Red");
console.log("[5] Yellow");

choice = parseInt(prompt("Enter your choice: "))

if (choice == 1)
{
    console.log("You might like my black t-shirt.");
}
else if (choice == 2)
{
    console.log("You might be a clean and tidy person.");
}
else if (choice == 3)
{
    console.log("You might be sad today.");
}
else if (choice == 4)
{
    console.log("You might be inlove right now.");
}
else if (choice == 5)

```

```

{
  console.log("Lemon might be your favorite fruit.");
}
else
{
  console.log("Sorry, your favorite color is not in the choices above.");
}

```

```

What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 1
You might like my black t-shirt.
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 999
Sorry, your favorite color is not in the choices above.

```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغامهای مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالت‌های انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

دستور if تو در تو

می‌توان از دستور if تو در تو در جاوااسکریپت استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```

if (condition)
{
  code to execute;

  if (condition)
  {
    code to execute;
  }
  else if (condition)
  {
    if (condition)
    {
      code to execute;
    }
  }
}
else
{
  if (condition)
  {
    code to execute;
  }
}

```



```

    }
}

```

اجازه بدهید که نحوه استفاده از دستور `if` تو در تو را نشان دهیم :

```

1  var age;
2  var gender;
3
4  age = parseInt(prompt("Enter your age: "));
5
6  gender = prompt("Enter your gender (male/female): ");
7
8  if (age > 12)
9  {
10     if (age < 20)
11     {
12         if (gender == "male")
13         {
14             console.log("You are a teenage boy.");
15         }
16         else
17         {
18             console.log("You are a teenage girl.");
19         }
20     }
21     else
22     {
23         console.log("You are already an adult.");
24     }
25 }
26 else
27 {
28     console.log("You are still too young.");
29 }

```

```

Enter your age: 18
Enter your gender: male
You are a teenage boy.
Enter your age: 12
Enter your gender: female
You are still too young.

```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا یک پنجره باز شده و برنامه از شما درباره سنتان (خط 4) و سپس پنجره دوم باز شده (خط 5) و درباره جنسیت از شما سؤال می‌کند. سپس به اولین دستور `if` می‌رسد (خط 8). در این قسمت اگر سن شما بیشتر از 12 سال باشد برنامه وارد بدنه دستور `if` می‌شود در غیر اینصورت وارد بلوک `else` (خط 26) مربوط به همین دستور `if` می‌شود. حال فرض کنیم که سن شما بیشتر از 12 سال است و شما وارد بدنه اولین `if` شده‌اید. در بدنه اولین `if` دو دستور `if` دیگر را مشاهده می‌کنید. اگر سن کمتر از 20 باشد شما وارد بدنه `if` دوم می‌شوید و اگر نباشد به قسمت `else` متناظر با آن می‌روید (خط 21). دوباره فرض می‌کنیم که سن شما کمتر از 20 باشد، در اینصورت وارد بدنه `if` دوم شده و با یک `if` دیگر مواجه می‌شوید (خط 12). در اینجا جنسیت شما مورد بررسی قرار می‌گیرد که اگر برابر "male" باشد، کدهای داخل بدنه سومین `if` اجرا می‌شود در غیر اینصورت قسمت `else` مربوط به این `if` اجرا می‌شود (خط 16). پیشنهاد می‌شود که از `if` تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را در گیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است :

تأثیر	مثال	تلفظ	عملگر
مقدار Z در صورتی true است که هر دو شرط دو طرف عملگر مقدارشان true باشد. اگر فقط مقدار یکی از شروط false باشد مقدار false، z خواهد شد.	$z = (x > 2) \ \&\& \ (y < 10)$	And	&&
مقدار Z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشان false باشد مقدار false، z خواهد شد.	$z = (x > 2) \ \ (y < 10)$	Or	
مقدار Z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد.	$z = !(x > 2)$	Not	!

به عنوان مثال جمله $z = (x > 2) \ \&\& \ (y < 10)$ را به این صورت بخوانید: "در صورتی مقدار z برابر true است که مقدار x بزرگتر از 2 و مقدار y کوچکتر از 10 باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد. عملگر منطقی (||) OR تأثیر متفاوتی نسبت به عملگر منطقی (&&) AND دارد. نتیجه عملگر منطقی OR برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد. می‌توان عملگرهای منطقی AND و OR را با هم ترکیب کرده و در یک عبارت به کار برد مانند:

```
if ((x == 1) && ((y > 3) || z < 10)) {
    //do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می‌کنیم. در اینجا ابتدا عبارت $(z < 10) \ || \ (y > 3)$ مورد بررسی قرار می‌گیرد (به علت تقدم عملگرها). سپس نتیجه آن بوسیله عملگر AND با نتیجه $(x == 1)$ مقایسه می‌شود. حال بیابید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم :

```
1 var age;
2 var gender;
3
4 age = parseInt(prompt("Enter your age: "));
5 gender = prompt("Enter your gender (male/female): ");
6
7 if (age > 12 && age < 20)
8 {
```

```

9     if (gender == "male")
10    {
11        console.log("You are a teenage boy.");
12    }
13    else
14    {
15        console.log("You are a teenage girl.");
16    }
17 }
18 else
19 {
20     console.log("You are not a teenager.");
21 }

```

```

Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
Enter your age: 10
Enter your gender (male/female): male
You are not a teenager.

```

برنامه بالا نحوه استفاده از عملگر منطقی AND را نشان می‌دهد (خط 7). وقتی به دستور if می‌رسید (خط 7) برنامه سن شما را چک می‌کند. اگر سن شما بزرگتر از 12 و کوچکتر از 20 باشد (سنتان بین 12 و 20 باشد) یعنی مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود. عملگر AND عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند.

```

if (x == 2 & y == 3)
{
    //Some code here
}

if (x == 2 | y == 3)
{
    //Some code here
}

```

نکته مهم اینجاست که شما می‌توانید از عملگرهای & و | به عنوان عملگر بیتی استفاده کنید. تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد عملوند سمت چپ به وسیله عملگر بیتی (&) AND ارزیابی می‌شود. اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی AND (&&) و OR (||) به جای عملگرهای بیتی (&) AND و (|) OR بهتر خواهد بود. یکی دیگر از عملگرهای منطقی عملگر (!) NOT است که نتیجه یک عبارت را خنثی یا منفی می‌کند. به مثال زیر توجه کنید:

```

if (!(x == 2))
{
    console.log("x is not equal to 2.");
}

```

```
}

```

اگر نتیجه عبارت `x == false` باشد عملگر `!` آن را `True` می‌کند.

دستور Switch

در جاوا اسکریپت ساختاری به نام `switch` وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور `switch` معادل دستور `if` تو در تو است با این تفاوت که در دستور `switch` متغیر فقط مقادیر ثابتی از اعداد، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغییر نیستند. در زیر نحوه استفاده از دستور `switch` آمده است :

```
switch (testVar)
{
  case compareVa11:
    code to execute if testVar == compareVa11;
    break;
  case compareVa12:
    code to execute if testVar == compareVa12;
    break;
  .
  .
  .
  case compareVa1N:
    code to execute if testVer == compareVa1N;
    break;
  default:
    code to execute if none of the values above match the testVar;
    break;
}
```

ابتدا یک مقدار در متغیر `switch` که در مثال بالا `testVar` است قرار می‌دهید. این مقدار با هر یک از عبارتهای `case` داخل بلوک `switch` مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات `case` برابر بود کد مربوط به آن `case` اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور `case` از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. آخر هر دستور `case` با کلمه کلیدی `break` تشخیص داده می‌شود که باعث می‌شود برنامه از دستور `switch` خارج شده و دستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه نمی‌شود ولی مفسر به اجرای دستور هر یک از `case` های زیرین تا پایان ادامه خواهد داد. دستور `switch` یک بخش `default` دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات `case` برابر نباشد. دستور `default` اختیاری است و اگر از بدنه `switch` حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور `switch` توجه کنید :

```
1 var choice;
2
3 console.log("What's your favorite pet?");
4 console.log("[1] Dog");
5 console.log("[2] Cat");
6 console.log("[3] Rabbit");
```

```

7 console.log("[4] Turtle");
8 console.log("[5] Fish");
9 console.log("[6] Not in the choices");
10
11 choice = parseInt(prompt("Enter your choice: "));
12
13 console.log("Enter your choice: " + choice);
14
15 switch (choice)
16 {
17     case 1:
18         console.log("Your favorite pet is Dog.");
19         break;
20     case 2:
21         console.log("Your favorite pet is Cat.");
22         break;
23     case 3:
24         console.log("Your favorite pet is Rabbit.");
25         break;
26     case 4:
27         console.log("Your favorite pet is Turtle.");
28         break;
29     case 5:
30         console.log("Your favorite pet is Fish.");
31         break;
32     case 6:
33         console.log("Your favorite pet is not in the choices.");
34         break;
35     default:
36         console.log("You don't have a favorite pet.");
37         break;
38 }

```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. با اجرای برنامه یک پنجره به صورت زیر نمایش داده می‌شود که شما باید در آن یک عدد وارد کنید:

شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر case ها برابر نبود دستور default اجرا می‌شود. مثلا اگر در پنجره باز شده عدد 2 را بنویسیم، خروجی برنامه به صورت زیر خواهد بود:

```

What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit

```

```
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.
```

و اگر مثلاً عدد 99 را وارد کنیم، دستورات بخش default اجرا می شوند و خروجی به صورت زیر خواهد شد:

```
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

یکی دیگر از ویژگیهای دستور switch این است که شما می‌توانید از دو یا چند case برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number عدد 1، 2 یا 3 باشد یک کد اجرا می‌شود. توجه کنید که case ها باید پشت سر هم نوشته شوند.

```
switch (number)
{
  case 1:
  case 2:
  case 3:
    console.log("This code is shared by three values.");
    break;
}
```

همانطور که قبلاً ذکر شد دستور switch معادل دستور if تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان نوشت :

```
if (choice == 1)
  console.log("Your favorite pet is Dog.");
else if (choice == 2)
  console.log("Your favorite pet is Cat.");
else if (choice == 3)
  console.log("Your favorite pet is Rabbit.");
else if (choice == 4)
  console.log("Your favorite pet is Turtle.");
else if (choice == 5)
  console.log("Your favorite pet is Fish.");
else if (choice == 6)
  console.log("Your favorite pet is not in the choices.");
else
  console.log("You don't have a favorite pet.");
```

کد بالا دقیقاً نتیجه‌ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد.

تکرار

ساختارهای تکرار به شما اجازه می‌دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید. بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است. مثلاً شما مجبورید 10 بار جمله "Hello World" را تایپ کنید مانند مثال زیر :

```
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
console.log("Hello World.");
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه‌ها است. حلقه‌ها در جاوااسکریپت عبارتند از :

- while
- do while
- for
- for...In
- for...Of

درباره حلقه for...In، بعد از مبحث آرایه‌ها توضیح می‌دهیم.

حلقه While

ابتدایی‌ترین ساختار تکرار در جاوااسکریپت حلقه While است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار حلقه While به صورت زیر است :

```
while (condition)
{
    code to loop;
}
```

می‌بینید که ساختار While مانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است می‌نویسیم اگر نتیجه درست یا true باشد سپس کدهای داخل بلوک While اجرا می‌شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه While برسد هیچکدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه While اصلاح شوند.

به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه While آمده است :

```
1 var counter = 1;
2
3 while (counter <= 10)
4 {
5     console.log("Hello World!");
6     counter++;
7 }
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا 10 بار پیام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام 10 خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق ببندیم. ابتدا در خط 1 یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار 1 را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد.

در خط 3 حلقه While را وارد می‌کنیم. در حلقه While ابتدا مقدار اولیه شمارنده با 10 مقایسه می‌شود که آیا از 10 کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه While و چاپ پیام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط 6). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از 10 کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت < از <= استفاده شده است. اگر از علامت < استفاده می‌کردیم که ما 9 بار تکرار می‌شد چون مقدار اولیه 1 است و هنگامی که شرط به 10 برسد false می‌شود چون 10 < 10 نیست. اگر می‌خواهید یک حلقه بی‌نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while (true)
{
    //code to loop
}
```


این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات `break` و `return` که در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقه `do while`

حلقه `do while` یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه `while` است با این تفاوت که در این حلقه ابتدا کد اجرا می شود و سپس شرط مورد بررسی قرار می گیرد. ساختار حلقه `do while` به صورت زیر است :

```
do
{
    code to repeat;
} while (condition);
```

همانطور که مشاهده می کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می شوند. برخلاف حلقه `while` که اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی شوند. برای اثبات این موضوع به کدهای زیر توجه کنید :

```
var number = 1;

do
{
    console.log("Hello World!");
} while (number > 10);
```

```
Hello World!
```

با اجرای کد بالا، اول دستورات بلوک `do` اجرا می شوند و بعد مقدار `number` با عدد 10 مقایسه می شود. در نتیجه حتی اگر شرط نادرست باشد باز هم قسمت `do` حداقل یک بار اجرا می شوند.

```
var number = 1;

while (number > 10)
{
    console.log("Hello World!");
}
```

اما در کد بالا چون اول مقدار `number` ابتدا مورد مقایسه قرار می گیرد، اگر شرط درست نباشد دیگر کدی اجرا نمی شود. یکی از موارد برتری استفاده از حلقه `do while` نسبت به حلقه `while` زمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. در دو کد زیر، یک عملیات یکسان توسط دو حلقه `while` و `do while` پیاده سازی شده است :

```
//while version

number = parseInt(prompt("Enter a number greater than 10: "));

while (number < 10)
```

```

{
    number = parseInt(prompt("Enter a number greater than 10: "));
}

//do while version

do
{
    number = parseInt(prompt("Enter a number greater than 10: "));
} while (number < 10);

```

مشاهده می‌کنید که از کدهای کمتری در بدنه do while نسبت به while استفاده شده است.

حلقه for

یکی دیگر از ساختارهای تکرار حلقه for است. این حلقه عملی شبیه به حلقه while انجام می‌دهد و فقط دارای چند خصوصیت اضافی است. ساختار حلقه for به صورت زیر است :

```

for (initialization; condition; operation)
{
    code to repeat;
}

```

- مقدار دهی اولیه (initialization) اولین مقداری است که به شمارنده حلقه می‌دهیم. شمارنده فقط در داخل حلقه قابل دسترسی است.
- شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقه ادامه یابد یا نه.
- عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد.

در زیر یک مثال از حلقه for آمده است:

```

for (var i = 1; i <= 10; i++)
{
    console.log("Number " + i);
}

```

```

Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10

```

برنامه بالا اعداد 1 تا 10 را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار 1 مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار 10 مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته Number و سپس مقدار جاری i یعنی 1 را چاپ می‌کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر 2 می‌شود و بار دیگر i با عدد 10 مقایسه می‌شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می‌یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید :

```
for (var i = 10; i > 0; i--)
{
    //code omitted
}
```

کد بالا اعداد را از 10 به 1 چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را 10 می‌دهیم و با استفاده از عملگر کاهش (-) برنامه‌ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. می‌توان قسمت شرط و عملگر را به صورت‌های دیگر نیز تغییر داد. به عنوان مثال می‌توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می‌توانید از چندین متغیر در ساختار حلقه for استفاده کنید.

```
for (var i = 1, y = 20; i < 10 && y >= 2; i++ , y -= 2)
{
    //some code here
}
```

به این نکته توجه کنید که اگر از چندین متغیر شمارنده یا عملگر در حلقه for استفاده می‌کنید باید آنها را با استفاده از کاما از هم جدا کنید.

حلقه های تو در تو (Nested Loops)

جاوااسکریپت به شما اجازه می‌دهد که از حلقه‌ها به صورت تو در تو استفاده کنید. اگر یک حلقه در داخل حلقه دیگر قرار بگیرد، به آن حلقه تو در تو گفته می‌شود. در این نوع حلقه‌ها، به ازای اجرای یک بار حلقه بیرونی، حلقه داخلی به طور کامل اجرا می‌شود. در زیر نحوه ایجاد حلقه تو در تو آمده است :

```
for (init; condition; increment)
{
    for (init; condition; increment)
    {
        //statement(s);
    }
    //statement(s);
}

while (condition)
{
    while (condition)
```

```

    {
        //statement(s);
    }
    //statement(s);
}

do
{
    //statement(s);
    do
    {
        //statement(s);
    }
    while (condition);
}
while (condition);

```

نکته‌ای که در مورد حلقه‌های تو در تو وجود دارد این است که، می‌توان از یک نوع حلقه در داخل نوع دیگر استفاده کرد. مثلاً می‌توان از حلقه for در داخل حلقه while استفاده نمود. در مثال زیر نحوه استفاده از این حلقه‌ها ذکر شده است. فرض کنید که می‌خواهید یک مستطیل با 3 سطر و 5 ستون ایجاد کنید :

```

for (var i = 1; i <= 4; i++)
{
    for (var j = 1; j <= 5; j++)
    {
        console.log(" * ");
    }
    console.log("\n");
}

```

```

* * * * *
* * * * *
* * * * *
* * * * *

```

در کد بالا به ازای یک بار اجرای حلقه for اول (خط 1)، حلقه for دوم (3-6) به طور کامل اجرا می‌شود. یعنی وقتی مقدار i برابر عدد 1 می‌شود، علامت * توسط حلقه دوم 5 بار چاپ می‌شود، وقتی i برابر 2 می‌شود، دوباره علامت * پنج بار چاپ می‌شود و ... در کل منظور از دو حلقه for این است که در 4 سطر علامت * در 5 ستون چاپ شود یا 4 سطر ایجاد شود و در هر سطر 5 بار علامت * چاپ شود. خط 7 هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد می‌شود و علامت‌های * در خطوط جدید چاپ می‌شوند. البته به جای این خط می‌توان `console.log("\n");` را هم نوشت.

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از break و continue را نشان می‌دهد:

```

1 console.log("Demonstrating the use of break.\n");
2
3 for (var x = 1; x < 10; x++)
4 {
5     if (x == 5)
6         break;
7
8     console.log("Number " + x);
9 }
10
11 console.log("\nDemonstrating the use of continue.\n");
12
13 for (var x = 1; x < 10; x++)
14 {
15     if (x == 5)
16         continue;
17
18     console.log("Number " + x);
19 }

```

Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
Number 9

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای for از حلقه‌های while و do...while استفاده می‌شد نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط 5) آمده است، وقتی که مقدار x به عدد 5 برسد، سپس دستور break اجرا (خط 6) و حلقه بلافاصله متوقف می‌شود، حتی اگر شرط $x < 10$ برقرار باشد. از طرف دیگر در خط 15 حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد. (وقتی مقدار x برابر 5 شود حلقه از 5 رد شده و مقدار 5 را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند).

آرایه

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

```
var numbers = [];
```

numbers نام آرایه را نشان می دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه ای که اعداد را در خود ذخیره می کند از کلمه number استفاده کنید. حتی می توانیم به هنگام ایجاد آرایه مقادیر خانه های آن را نیز مشخص کنیم :

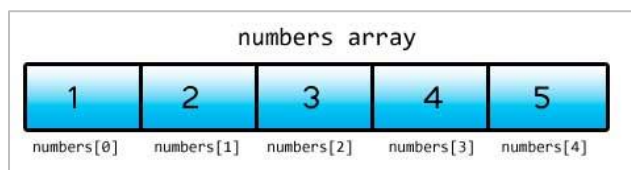
```
var numbers = [1, 2, 3, 4, 5];
```

در این مثال 5 مقدار در 5 آدرس از فضای حافظه کامپیوتر شما ذخیره می شود. مثلا بالا را به روش زیر هم می توان پیاده سازی کرد :

```
var numbers = [];
```

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

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



به هر یک از اجزاء آرایه و اندیسهای داخل گروه توجه کنید. کسانی که تازه شروع به برنامه نویسی کرده اند معمولا در گذاشتن اندیس دچار اشتباه می شوند و مثلا ممکن است در مثال بالا اندیسها را از 1 شروع کنند. در مثال های بالا، ما یک آرایه با طول نا مشخص تعریف

کردیم. یعنی مشخص نکردیم که چه تعداد عنصر قرار است در آرایه قرار بگیرند. اگر بخواهیم که تعداد عناصر را هم مشخص کنیم می‌توانیم از خاصیت `length` به صورت زیر استفاده کنیم:

```
var numbers = [];
numbers.length = 5;
```

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

```
var numbers = new Array();
```

می‌توان عناصر آرایه را در داخل پرانتز هم نوشت:

```
var numbers = new Array(1, 2, 3, 4, 5);
```

آرایه بالا را به صورت زیر هم می‌توان تعریف کرد:

```
var numbers = new Array(5);

numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

عدد 5 در داخل پرانتز کد بالا بدین معنی است که ما می‌خواهیم 5 عنصر را در داخل آرایه ذخیره کنیم.

دستیابی به مقادیر آرایه با استفاده از حلقه `for`

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه 5 مقدار از کاربر گرفته شده و میانگین آنها حساب می‌شود:

```
1 var numbers = new Array(5);
2 var total = 0;
3 var average;
4
5 for (var i = 0; i < numbers.length; i++)
6 {
7     numbers[i] = parseInt(prompt("Enter a number: "));
8     console.log("Number " + i + ": " + numbers[i]);
9 }
10
11 for (var i = 0; i < numbers.length; i++)
12 {
13     total += numbers[i];
14 }
15
16 average = total / numbers.length;
17
```

```
18 console.log("Average = ", average);
```

با اجرای برنامه بالا، 5 بار پنجره ای به شما نمایش داده می شود که هر بار شما باید یک عدد را وارد کنید تا این عدد در داخل یکی از خانه های آرایه قرار بگیرد. در خط 1 یک آرایه تعریف شده است که می تواند 5 عدد صحیح را در خود ذخیره کند. خطوط 2 و 3 متغیرهایی تعریف شده اند که از آنها برای محاسبه میانگین استفاده می شود. توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط 5 تا 9 حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از خاصیت طول (length) ارائه برای تشخیص تعداد اجزای آرایه استفاده می شود. اگر چه می توانستیم به سادگی در حلقه for مقدار 5 را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت تری است و می توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ می شود. در خط 7 ورودی دریافت شده از کاربر به نوع int تبدیل و در آرایه ذخیره می شود. اندیس استفاده شده در number (خط 7) مقدار 1 جاری در حلقه است. برای مثال در ابتدای حلقه مقدار i صفر است بنابراین وقتی در خط 7 اولین داده از کاربر گرفته می شود اندیس آن برابر 0 می شود. در تکرار بعدی i یک واحد اضافه می شود و در نتیجه در خط 7 و بعد از ورود دومین داده توسط کاربر اندیس آن برابر 1 می شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می یابد. خط 8 خم اعدادی را که شما در پنجره وارد می کنید در محیط console چاپ می کند.

در خطوط 11-14 از حلقه for دیگر برای دسترسی به مقدار هر یک از داده های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می کنیم. هر یک از اجزای عددی آرایه به متغیر total اضافه می شوند. بعد از پایان حلقه می توانیم میانگین اعداد را حساب کنیم (خط 16). مقدار total را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می کنیم. برای دسترسی به تعداد اجزای آرایه می توان از خاصیت length آرایه استفاده کرد. خط 18 مقدار میانگین را در صفحه نمایش چاپ می کند. خروجی برنامه بالا می تواند به صورت زیر باشد (البته ممکن است برای شما متفاوت باشد):

```
Number 0: 95
Number 1: 85
Number 2: 80
Number 3: 87
Number 4: 92
Average = 87.8
```

آرایه ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید بسیار مهم است.

حلقه for...of

حلقه for...of یکی دیگر از ساختارهای تکرار در جاوااسکریپت می باشد که مخصوصاً برای آرایه ها، رشته ها و مجموعه ها طراحی شده است. حلقه for...of با هر بار گردش در بین اجزاء، مقادیر هر یک از آنها را در داخل یک متغیر موقتی قرار می دهد و شما می توانید بواسطه این متغیر به مقادیر دسترسی پیدا کنید. در زیر نحوه استفاده از حلقه for...of آمده است :


```
for (var temporaryVar of array / string)
{
    code to execute;
}
```

temporaryVar متغیری است که مقادیر اجزای آرایه را در خود نگهداری می‌کند. سپس کلمه کلیدی of و بعد از آن نام آرایه، رشته و یا مجموعه را می‌نویسیم. در زیر نحوه استفاده از حلقه for...of آمده است :

```
1 var numbers = [ 1, 2, 3, 4, 5 ];
2
3 for (var n of numbers)
4 {
5     console.log("Number ", n);
6 }
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
```

در برنامه آرایه‌ای با 5 جزء تعریف شده و مقادیر 1 تا 5 در آنها قرار داده شده است (خط 1). در خط 3 حلقه for...of شروع می‌شود. ما یک متغیر موقتی تعریف کرده‌ایم که اعداد آرایه را در خود ذخیره می‌کند. در هر بار تکرار از حلقه for...of متغیر موقتی n، مقادیر عددی را از آرایه استخراج می‌کند. حلقه for...of مقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می‌دهد. حلقه for...of ما را قادر می‌سازد که به داده‌ها دسترسی یابیم و یا آنها را بخوانیم و اصلاح کنیم. برای درک این مطلب در مثال زیر مقدار هر یک از اجزای آرایه افزایش یافته است :

```
var numbers = [1, 2, 3, 4, 5];

for (var n of numbers)
{
    console.log(++n);
}
```

```
2
3
4
5
6
```

آرایه‌های چند بعدی

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

```
var array_name = [[value1, value2, value3], [val1, val2, val3]];
```

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

```
var numbers = [
    [ 1, 2, 3, 4, 5],
    [ 6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
];
```

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

```
var numbers = [];
numbers[0][0] = 1;
numbers[0][1] = 2;
numbers[0][2] = 3;
numbers[0][3] = 4;
numbers[0][4] = 5;
numbers[1][0] = 6;
numbers[1][1] = 7;
numbers[1][2] = 8;
numbers[1][3] = 9;
numbers[1][4] = 10;
numbers[2][0] = 11;
numbers[2][1] = 12;
numbers[2][2] = 13;
numbers[2][3] = 14;
numbers[2][4] = 15;
```

همانطور که مشاهده می کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می توان از اندیسهای سطر و ستون و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه های چند بعدی

گردش در میان عناصر آرایه های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه for...of و یا حلقه for تو در تو است. اجازه دهید ابتدا از حلقه for...of استفاده کنیم:

```
var numbers = [
    [ 1, 2, 3, 4, 5],
    [ 6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
```

```

];
for (var array of numbers)
{
  for (var num of array)
  {
    console.log(num);
  }
}

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```

مشاهده کردید که گردش در میان مقادیر عناصر یک آرایه چند بعدی چقدر راحت است. حلقه `for...of` اول برای گردش در میان عناصر آرایه اصلی یعنی `numbers` و حلقه `for...of` دوم برای گردش در میان عناصر آرایه های عضو، به کار رفته است. حال همین کار را با حلقه `for` انجام می دهیم:

```

1 var numbers = [
2     [ 1, 2, 3, 4, 5],
3     [ 6, 7, 8, 9, 10],
4     [11, 12, 13, 14, 15]
5 ];
6
7 for (var row = 0; row < numbers.length; row++)
8 {
9     for (var col = 0; col < numbers[row].length; col++)
10    {
11        console.log(numbers[row][col]);
12    }
13 }

```

```

1
2
3
4
5
6
7
8
9
10
11
12

```

13
14
15

از آنجاییکه آرایه دو بعدی به صورت یک جدول شامل سطر و ستون است، پس لازم است که از یک حلقه for برای گردش در میان سطرها و از حلقه for دیگر برای گردش در میان ستون‌های این جدول (آرایه) استفاده کنیم. اولین حلقه for (خط 7) برای گردش در میان ردیف‌های آرایه به کار می‌رود. این حلقه به تعداد سطرها یعنی سه بار تکرار می‌شود. در این مثال از خاصیت length آرایه استفاده کرده‌ایم. این خاصیت تعداد عناصر آرایه را نشان می‌دهد.

در داخل اولین حلقه for حلقه for دیگری تعریف شده است (خط 9). در این حلقه یک شمارنده برای شمارش تعداد ستونها (col) که در واقع همان عناصر آرایه ها هستند، تعریف شده است و در شرط داخل آن بار دیگر از خاصیت length استفاده شده است. در مجموع این حلقه بیرونی برای شمارش تعداد عناصر آرایه اصلی و حلقه داخلی برای شمارش تعداد عناصر آرایه های داخلی می باشد. پس به عنوان مثال وقتی که مقدار ردیف (row) صفر باشد، حلقه دوم از [0][0] تا [0][4] اجرا می‌شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر 0 و مقدار ستون (col) برابر 0 باشد مقدار عنصری که در ستون 1 و ردیف 1 (numbers[0][0]) قرار دارد نشان داده خواهد شد که در مثال بالا عدد 1 است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور console.log("\n") که به برنامه اطلاع می‌دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می‌کند. سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. این فرایند تا زمانی اجرا می‌شود که مقدار row کمتر از طول اولین بعد باشد. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار ببریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```

1 var studentGrades = [
2     new Array(4),
3     new Array(4),
4     new Array(4)
5 ];
6 var total;
7
8 for (var student = 0; student < studentGrades.length; student++)
9 {
10     total = 0;
11
12     console.log("Enter grades for Student", student + 1);
13
14     for (var grade = 0; grade < studentGrades[student].length; grade++)
15     {
16         studentGrades[student][grade] = prompt("Enter Grade " + (grade + 1));
17         console.log("Enter Grade #" + (grade + 1) + ": " +
18 studentGrades[student][grade]);
19         total += parseFloat(studentGrades[student][grade]);
20     }

```

```

21
22     console.log("Average is ",(total /
23 (studentGrades[student].length)).toFixed(2));
    console.log("\n");
}

```

```

Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

```

```

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

```

```

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00

```

در برنامه بالا یک آرایه چند بعدی از نوع double تعریف شده است (خط 5-1). همچنین یک متغیر به نام total تعریف می‌کنیم که جمع نمرات وارد شده برای دانش آموز در آن قرار می‌گیرد (خط 6). حال وارد حلقه for تو در تو می‌شویم (خط 23-8). در اولین حلقه for یک متغیر به نام student تعریف کرده‌ایم که تعداد دانش آموزان در آن قرار می‌گیرد. از خاصیت length هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می‌شویم. در خط 10 مقدار متغیر total را برابر 0 قرار می‌دهیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که نمرات دانش آموز را وارد کنید (student + 1). عدد 1 را به student اضافه کرده‌ایم تا به جای نمایش 0 Student، با 1 Student شروع شود، تا طبیعی‌تر به نظر برسد (خط 12). سپس به دومین حلقه for در خط 14 می‌رسیم. وظیفه این حلقه گردش در میان دومین بعد که همان نمرات دانش آموز است می‌باشد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود.

وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خط 21 معدل دانش آموز نشان داده می‌شود. به متد toPrecision(2) توجه کنید. این متد معدل را تا دو رقم اعشار نشان می‌دهد. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید. از studentGrades[student].length هم برای به دست آوردن تعداد نمرات استفاده می‌شود.



از سایر کتاب های یونس ابراهیمی در لینک های زیر دیدن فرمایید

<https://bit.ly/2Is8a0t>

www.w3-farsi.com/product

تابع

تابع به شما اجازه می دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه ای از کدها است که در هر جای برنامه می توان از آنها استفاده کرد. توابع در جاوااسکریپت و اکثر زبانهای برنامه نویسی بر دو نوعند :

- توابع از پیش تعریف شده
- توابعی که توسط کاربر تعریف می شوند.

در مورد توابع از پیش تعریف شده در درس های آینده توضیح می دهیم. در این درس به شما یاد می دهیم که چگونه یک تابع را ایجاد کنید. ساده ترین ساختار یک تابع به صورت زیر است :

```
1 function PrintMessage()  
2 {  
3     console.log("Hello World!");  
4 }  
5  
6 PrintMessage ();
```

```
Hello World!
```

در خطوط 1-4 یک تابع تعریف کرده ایم. همانطور که مشاهده می کنید در خط 1 و برای تعریف تابع از کلمه کلیدی function سپس نام تابع و بعد از آن پرانتز باز و بسته استفاده کرده ایم. نام تابع ما PrintMessage() است. به این نکته توجه کنید که در نامگذاری تابع از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می شود) استفاده کرده ایم. این روش نامگذاری قراردادی است و می توان از این روش

استفاده نکرد، اما پیشنهاد می شود که از این روش برای تشخیص توابع استفاده کنید. بهتر است در نامگذاری توابع از کلماتی استفاده شود که کار آن تابع را مشخص می کند مثلا نام هایی مانند GoToBed یا OpenDoor.

همچنین به عنوان مثال اگر مقدار برگشتی (در درس های آینده توضیح می دهیم) تابع یک مقدار بولی باشد می توانید اسم تابع خود را به صورت یک کلمه سوالی انتخاب کنید مانند IsLeapyear یا IsTeenager... ولی از گذاشتن علامت سوال در آخر اسم تابع خودداری کنید. دو پرانتزی که بعد از نام می آید نشان دهنده آن است که نام متعلق به یک تابع است. بعد از پرانتزها دو آکولاد قرار می دهیم که بدنه تابع را تشکیل می دهد و کدهایی را که می خواهیم اجرا شوند را در داخل این آکولاد ها می نویسیم. در خط 6 تابع را صدا می زنیم. برای صدا زدن یک تابع کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

برای اجرای تابع PrintMessage() برنامه از خط به محل تعریف تابع PrintMessage() می رود. مثلا وقتی ما تابع PrintMessage() را در خط 6 صدا می زنیم برنامه از خط 6 به خط 1، یعنی جایی که تابع تعریف شده می رود و کدهای آن را اجرا می کند .

مقدار برگشتی از یک تابع

توابع می توانند مقدار برگشتی از هر نوع داده ای داشته باشند. این مقادیر می توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک تابع است و شما او را صدا می زنید و از او می خواهید که کار یک سند را به پایان برساند. سپس از او می خواهید که بعد از اتمام کارش سند را به شما تحویل دهد.

سند همان مقدار برگشتی تابع است. نکته مهم در مورد یک تابع، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک تابع آسان است. کافیست در تعریف تابع به روش زیر عمل کنید:

```
function FunctionName()
{
    return value;
}
```

همانطور که مشاهده می کنید مقدار بازگشتی از تابع یعنی value را جلوی دستور return می نویسیم. مثال زیر یک تابعی که دارای مقدار برگشتی است را نشان می دهد:

```
function CalculateSum()
{
    var firstNumber = 10;
    var secondNumber = 5 ;
    var sum = firstNumber + secondNumber ;

    return sum;
}

var result = CalculateSum();
console.log(result);
```

15

در خطوط 3 و 4 مثال فوق، دو متغیر تعریف و مقدار دهی شده اند. توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر توابع قابل دسترسی نیستند و فقط در تابعی که در آن تعریف شده اند قابل استفاده هستند. در خط 5 جمع دو متغیر در متغیر sum قرار می گیرد. در خط 7 مقدار برگشتی sum توسط دستور return فراخوانی می شود. در خط 10 یک متغیر به نام result تعریف می کنیم و تابع CalculateSum() را فراخوانی می کنیم.

تابع CalculateSum() مقدار 15 را بر می گرداند که در داخل متغیر result ذخیره می شود. در خط 11 مقدار ذخیره شده در متغیر result چاپ می شود. تابعی که در این مثال ذکر شد تابع کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در تابع بالا نوشته شده ولی همیشه مقدار برگشتی 15 است، در حالیکه می توانستیم به راحتی یک متغیر تعریف کرده و مقدار 15 را به آن اختصاص دهیم. این تابع در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می خواهیم در داخل یک تابع از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید :

```

1 function GetNumber()
2 {
3     var number = 11 ;
4
5     if (number > 10)
6     {
7         return number;
8     }
9     else
10    {
11        return 0;
12    }
13 }
14
15 var result = GetNumber();
16 console.log(result);

```

11

در خطوط 1-13 یک تابع با نام GetNumber() تعریف شده است. در خط 3 متغیری با مقدار 11 مقداردهی شده است که در خط 5 با مقدار 10 مقایسه می شود و چون مقدار این متغیر از 10 بیشتر است پس دستور return اول مقدار 11 را برمی گرداند. حال اگر مقدار این متغیر از 10 کمتر باشد دستور return مربوط به قسمت else اجرا و مقدار صفر چاپ می شود. که از کاربر یک عدد بزرگتر از 10 را می خواهد. اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه نتیجه چاپ نمی شود. چون اگر شرط دستور if نادرست باشد برنامه به قسمت else می رود تا مقدار صفر را بر گرداند و چون قسمت else حذف شده است برنامه هیچ مقداری را چاپ نمی کند و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد برنامه هیچ مقداری را چاپ

نمی‌کند. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک تابع که مقدار برگشتی ندارد خارج شوید. استفاده از return باعث خروج از بدنه تابع و اجرای کدهای بعد از آن می‌شود.

```
function TestReturnExit()
{
    console.log("Line 1 inside the method TestReturnExit()");

    return;

    console.log("Line 2 inside the method TestReturnExit()");
}
```

```
TestReturnExit();
Line 1 inside the method TestReturnExit()
```

در برنامه بالا نحوه خروج از تابع با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه تابع تعریف شده (TestReturnExit()) فراخوانی و اجرا می‌شود.

پارامترها و آرگومان‌ها

پارامترها، داده‌های خامی هستند که تابع آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک تابع می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک تابع با N پارامتر نشان داده شده است :

```
function FunctionName(param1, param2, ...paramN)
{
    code to execute;
}
```

پارامترها بعد از نام تابع و بین پرانتزها قرار می‌گیرند. بر اساس کاری که تابع انجام می‌دهد می‌توان تعداد پارامترهای زیادی به تابع اضافه کرد. بعد از فراخوانی یک تابع باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. اجازه بدهید که یک مثال بزنیم :

```
1 function CalculateSum(number1, number2)
2 {
3     return number1 + number2;
4 }
5
6 var result = CalculateSum(10,5);
7 console.log(result);
```

15

در برنامه بالا یک تابع به نام CalculateSum() (خطوط 1-4) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. تابع دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. در بدنه تابع دستور return نتیجه جمع دو عدد را بر می‌گرداند. در خط 6 دو عدد 5 و 10

را به عنوان آرگومان به تابع ارسال می‌کنیم. بعد از ارسال مقادیر 5 و 10 به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه تابع (خط 3) دو مقدار با هم جمع می‌شوند و در خط 7 نتیجه چاپ می‌شود. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما توابع کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک تابع را به عنوان آرگومان به تابع دیگر ارسال کنید:

```

1 function MyFunction()
2 {
3     return 5;
4 }
5
6 function AnotherFunction(number)
7 {
8     console.log(number);
9 }
10
11 AnotherFunction(MyFunction());

```

5

چون مقدار برگشتی تابع MyFunction() عدد 5 است و به عنوان آرگومان به تابع AnotherFunction() ارسال می‌شود خروجی کد بالا هم عدد 5 است. نکته پایانی اینکه مقدار پیشفرض همه پارامترها، اگر مقداری به آنها ارسال نشود، undefined می‌باشد:

```

function MyFunction(param)
{
    console.log(param);
}

MyFunction();

```

undefined

پارامترهای اختیاری

پارامترهای اختیاری همانگونه که از اسمشان پیداست اختیاری هستند و می‌توان به آنها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند. اگر به اینگونه پارامترها آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می‌کنند. به مثال زیر توجه کنید:

```

1 function PrintMessage(message = "Welcome to Javascript Tutorials!")
2 {
3     console.log(message);
4 }
5
6 PrintMessage();
7
8 PrintMessage("Learn Javascript Today!");

```

```

Welcome to Javascript Tutorials!
Learn Javascript Today!

```

تابع `PrintMessage()` (خطوط 1-4) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت `=` یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط 1). دو بار تابع را فراخوانی می‌کنیم. در اولین فراخوانی (خط 6) ما آرگومانی به تابع ارسال نمی‌کنیم بنابراین تابع از مقدار پیشفرض (`Welcome to Javascript Tutorials!`) استفاده می‌کند. در دومین فراخوانی (خط 8) یک پیغام (آرگومان) به تابع ارسال می‌کنیم که جایگزین مقدار پیشفرض پارامتر می‌شود. اگر از چندین پارامتر در تابع استفاده می‌کنید، بهتر است که همه پارامترهای اختیاری در آخر بقیه پارامترها ذکر شوند. به مثالهای زیر توجه کنید.

```
function SomeFunction(opt1 = 10, opt2 = 20, req1, req2) //Bad
function SomeFunction(req1, opt1 = 10, req2, opt2 = 20) //Bad
function SomeFunction(req1, req2, opt1 = 10, opt2 = 20) //Good
```

از آنجاییکه آرگومانهایی که به توابع ارسال می‌شوند از سمت چپ به راست در داخل پارامترها قرار می‌گیرند، اگر پارامترهای اختیاری در ابتدای پارامترها قرار داده شوند، مقادیر آنها توسط آرگومان های ارسال جایگزین می‌شوند. وقتی توابع با چندین پارامتر اختیاری فراخوانی می‌شوند باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند، مقدار اختصاص داد. به یاد داشته باشید که نمی‌توان برای نادیده گرفتن یک پارامتر به صورت زیر عمل کرد :

```
function SomeFunction(required1, optional1 = 10, optional2 = 20)
{
    //Some Code
}
// ... Code omitted for demonstration
SomeFunction(10, , 100); //Error
```

اگر بخواهید از یک پارامتر اختیاری که در آخر پارامترهای دیگر نیست رد شوید و آن را نادیده بگیرید، به طوریکه این پارامتر از مقدار پیشفرض خود استفاده کند، باید از کلمه `undefined` استفاده کنید :

```
SomeFunction(10, undefined, 100);
```

تابع بالا هیچ آرگومانی برای پارامتر اختیاری `optional1` ندارد بنابراین این پارامتر از مقدار پیشفرض خود یعنی 10 استفاده می‌کند.

نامیدن آرگومان ها

یکی دیگر از راه‌های ارسال آرگومانها استفاده از نام آنهاست. استفاده از نام آرگومانها شما را از به یاد آوری و رعایت ترتیب پارامترها هنگام ارسال آرگومانها راحت می‌کند. در عوض شما باید نام پارامترهای تابع را به خاطر بسپارید. استفاده از نام آرگومانها خوانایی برنامه را بالا می‌برد، چون شما می‌توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است. نامیدن آرگومانها یا `named argument` در ES6 مطرح شده است. در زیر نحوه استفاده از نام آرگومانها وقتی که تابع فراخوانی می‌شود نشان داده شده است :

```
function FunctionToCall({ paramName1, paramName2, ...paramNameN })
```

```
{
  // Some Code
}
```

حال به مثال زیر توجه کنید :

```
1 function SetSalaries( {jack, andy, mark} )
2 {
3   console.log("Jack's salary is ", jack);
4   console.log("Andy's salary is ", andy);
5   console.log("Mark's salary is ", mark);
6 }
7
8 SetSalaries( {jack: 120, andy: 30, mark: 75} );
9 console.log("\n");
10 SetSalaries( {andy: 60, mark: 150, jack: 50} );
11 console.log("\n");
12 SetSalaries( {mark: 35, jack: 80, andy: 150} );
```

```
Jack's salary is 120.
Andy's salary is 30.
Mark's salary is 75.
```

```
Jack's salary is 50.
Andy's salary is 60.
Mark's salary is 150.
```

```
Jack's salary is 80.
Andy's salary is 150.
Mark's salary is 35.
```

برای استفاده از نام پارامترها، باید آنها را در داخل آکولاد بنویسید و هنگام فراخوانی تابع هم به همین صورت عمل کنید. خروجی نشان می‌دهد که حتی اگر ما ترتیب آرگومانها در سه تابع فراخوانی شده را تغییر دهیم مقادیر مناسب به پارامترهای مربوطه‌شان اختصاص داده می‌شود. نمی‌توان از آرگومانهای دارای نام و آرگومانهای ثابت (مقداری) به طور همزمان استفاده کرد:

```
SetSalaries(30, andy: 50, mark: 60);
SetSalaries(30, mark: 60, andy: 50);
SetSalaries(mark: 60, andy: 50, 30);
SetSalaries(mark: 60, 30, andy: 50);
```

همه کدهای بالا، خطا ایجاد می‌کنند. ولی می‌توان از پارامترهای اختیاری به صورت زیر استفاده کرد:

```
function SetSalaries({ jack, andy, mark = 200 })
{
  // Some Code
}
```

Rest parameters

Rest parameters امکان ارسال تعداد دلخواه پارامترهای همونوع و ذخیره آنها در یک آرایه ساده را فراهم می‌آورد. برای ایجاد تابعی که به تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) به صورت زیر استفاده می‌شود:

```
function functionName(...variableName)
{
  ...
}
```

همانطور که در کد بالا مشاهده می‌کنید، کافیسیت، آرگومان‌هایی که تابع قرار است دریافت کن را به صورت ...variableName بنویسید. یعنی علامت سه نقطه (...) و به دنبال آن نام پارامتر را ذکر کنید. به مثال زیر توجه کنید:

```
1 function CalculateSum(... numbers)
2 {
3   var total = 0;
4
5   for (var number of numbers)
6   {
7     total += number;
8   }
9
10  return total;
11 }
12
13 console.log("1 + 2 + 3 = ", CalculateSum(1, 2, 3));
14
15 console.log("1 + 2 + 3 + 4 = ", CalculateSum(1, 2, 3, 4));
16
17 console.log("1 + 2 + 3 + 4 + 5 = ", CalculateSum(1, 2, 3, 4, 5));
```

```
1 + 2 + 3      = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
```

همانطور که در مثال بالا مشاهده می‌کنید، یک تابع به نام CalculateSum() در خط 1 تعریف شده است. برای اینکه این تابع تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) قبل از نوع داده‌ای پارامتر آن استفاده شده است. در اصل کلمه numbers یک آرایه است، که وقتی ما آرگومان‌ها را به تابع ارسال می‌کنیم، در این آرایه ذخیره می‌شوند. حال تابع را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم و سپس با استفاده از حلقه for/of این آرگومانها را جمع و به تابع فراخوان برگشت می‌دهیم. وقتی از چندین پارامتر در یک تابع استفاده می‌کنید، فقط یکی از آنها باید دارای علامت سه نقطه (...) بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد. اگر این پارامتر (پارامتری که دارای سه نقطه است) در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر سه نقطه دار استفاده کنید با خطا مواجه می‌شوید. به مثالهای اشتباه و درست زیر توجه کنید:

```
function SomeFunction(...x, y) //ERROR
function SomeFunction(x, ...y) //Correct
```

محدوده متغیر

متغیرها در جاوااسکریپت دارای محدوده هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک تابع تعریف می‌شود فقط در داخل بدنه تابع قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو تابع مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```

1 function firstLocalVariable()
2 {
3     var number = 10;
4     console.log(number);
5 }
6
7 function secondLocalVariable()
8 {
9     var number = 5;
10    console.log(number);
11 }
12
13 firstLocalVariable ();
14
15 secondLocalVariable ();

```

```

10
5

```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم (خطوط 3 و 9) که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل تابع `firstLocalVariable()` هیچ ارتباطی به متغیر داخل تابع `secondLocalVariable()` ندارد. وقتی به مبحث کلاسها رسیدیم در این باره بیشتر توضیح خواهیم داد. جاوااسکریپت دارای دو نوع محدوده است:

- متغیرهای محلی (Local)
- متغیرهای سراسری (Global)

متغیرهای محلی

متغیرهایی که داخل توابع تعریف می‌شوند محلی هستند و فقط داخل همان تابع قابل استفاده‌اند. به مثال زیر توجه کنید:

```

function LocalVariable()
{
    var number = 10;
    console.log(number);
}

LocalVariable ();

console.log(number);

```

```

10

```

```
Uncaught ReferenceError: number is not defined
```

همانطور که مشاهده می‌کنید با فراخوانی تابع در خط 7 مقدار متغیر number چاپ می‌شود ولی در خط 10 که سعی در چاپ مقدار این متغیر داریم با پیغام خطا مواجه می‌شویم چون طول عمر این متغیر تا زمانی است که تابع به پایان نرسیده است. با پایان تابع متغیر و مقدار آن هم از بین می‌رود در نتیجه در خارج از تابع نمی‌توان مقدار آن را چاپ کرد.

متغیرهای سراسری

متغیرهایی که در بیرون تابع تعریف می‌شوند از نوع سراسری هستند. به مثال زیر توجه کنید:

```
1 var firstNumber = 10;
2 var secondNumber = 5;
3 var Sum;
4
5 function GlobalVariable()
6 {
7     Sum = firstNumber + secondNumber;
8 }
9
10 GlobalVariable ();
11 console.log(Sum);
```

```
15
```

متغیرهای firstNumber و secondNumber و Sum در بیرون تابع تعریف شده‌اند و از نوع سراسری هستند. از این نوع متغیرها را به راحتی می‌توان در هر جای برنامه، استفاده کرد.

Arrow Function

Arrow Function روشی جدید برای تعریف توابع در جاوااسکریپت می‌باشد. این ویژگی بعد از معرفی ES6 در اختیار توسعه‌دهندگان قرار گرفت. Arrow Function به شما اجازه می‌دهند تا با استفاده از => به سرعت یک تابع را همراه با پارامتر و یا بدون پارامتر ایجاد کنید. با این روش می‌توانید بدون استفاده از کلمه کلیدی function و یا return یک تابع را بسازید. نحوه ایجاد یک Arrow Function به صورت زیر می‌باشد:

```
functionName = (param1, param2, ..., paramN) => { statements }
```

ساده‌ترین شکل یک تابع حالتی است که در آن ما هیچ پارامتر ورودی را وارد نمی‌کنیم:

```
var ShowMessage = () => console.log("Hello World!");
ShowMessage();
```

```
Hello World!
```

Arrow Function زیر یک آرگومان به عنوان ورودی دریافت می کند :

```
var ShowMessage = message => (console.log(message));
ShowMessage("Hello world!");
```

```
Hello world!
```

اگر یک Arrow Function دارای دو یا تعداد بیشتری پارامتر باشد باید آنها را در داخل پرانتز قرار دهید :

```
var ShowMessage = (message1, message2) => console.log(message1, message2);
ShowMessage("Hello ", "world!");
```

```
Hello World!
```

اگر بخواهید یک Arrow Function بیش از یک دستور را در خروجی نمایش دهد، باید خروجی ها را داخل پرانتز نوشته و آنها را با کاما از هم جدا کنید :

```
var ShowMessage = message => (console.log(message), console.log("Some more message"));
ShowMessage("Hello world!");
```

```
Hello World!
Some more message
```

Arrow Function نمی تواند دارای کلمه return باشند. می توان گفت که دستورات Arrow Function در حالت عادی برگردانده می شوند و نیازی به این کلمه نیست :

```
var GetSquare = number => number * number;
console.log(GetSquare(5));
```

```
25
```

توابع بی نام و توابع خود فراخوان

در جاوااسکریپت بحثی به نام توابع بی نام یا Anonymous functions مطرح شد که به شما اجازه می دهد یک تابع را به عنوان یک آرگومان به تابع دیگر ارسال کنید و یا آن را در داخل یک متغیر قرار دهید. توابع بی نام همانند توابع عادی تعریف می شوند و فقط نام ندارند و در بعد از آکولاد بسته آنها علامت سیمیکالن (;) قرار می گیرد. دستور استفاده از متدهای بی نام به صورت زیر است :

```
var variableName = function (parameters)
{
};
```


از آنجاییکه توابع بی نام، نام ندارند، پس نمی‌توان آنها را به صورت یک تابع معمولی صدا زد. برای این منظور باید نام متغیری که تابع را به آن اختصاص داده‌ایم را بنویسیم و سپس دو علامت پرانتز و یک سیمیکال در جلوی آن قرار دهیم. به کد زیر توجه کنید :

```
1 var ShowMessage = function(name)
2 {
3   console.log("Hello", name);
4 };
5
6 ShowMessage("World");
```

```
Hello World!
```

همانطور که در کد بالا مشاهده می‌کنید، ابتدا در خط 1 تابع بی نام را به متغیر ShowMessage اختصاص داده‌ایم و سپس در خط 6 نام این متغیر را نوشته و در با گذاشتن دو پرانتز در جلوی نام آن و ارسال آرگومان باعث اجرای کدهای بدنه متد بی نام شده‌ایم. یک نوع دیگر از توابع در جاوا اسکریپت وجود دارند که به صورت خودکار فراخوانی و کدهای آنها اجرا می‌شوند. به این توابع self-invoking گفته می‌شود. نحوه ایجاد یک تابع self-invoking به صورت زیر می‌باشد:

```
(
  function (parameters)
  {
    //Some code
  }
)
(Argument);
```

به مثال زیر توجه کنید:

```
1 (
2   function (name)
3   {
4     console.log("Hello", name);
5   }
6 )
7
8 ("World!");
```

```
Hello World!
```

همانطور که در کد بالا مشاهده می‌کنید، این توابع در داخل یک جفت پرانتز تعریف می‌شوند (خطوط 1-6) و برای صدا زدن آنها کافایت که یک جفت پرانتز گذاشته و در داخل این پرانتزها هم آرگومان هایی که لازم است به تابع ارسال شوند را بنویسیم (خط 8). اگر هم تابع هیچ آرگومانی قبول نکند، کافایت که پرانتزها را خالی بگذارید. کد بالا را به صورت زیر تغییر داده و نتیجه را مشاهده کنید:

```
(
  function ()
  {
    console.log("Hello World!");
  }
)
```

```
();
```

```
Hello World!
```

برنامه نویسی شیء گرا (Object Oriented Programming)

برنامه نویسی شیء گرا (OOP) شامل تعریف کلاسها و ساخت اشیاء مانند ساخت اشیاء در دنیای واقعی است. برای مثال یک ماشین را در نظر بگیرید. این ماشین دارای خواصی مانند رنگ، سرعت، مدل، سازنده و برخی خواص دیگر است. همچنین دارای رفتارها و حرکاتی مانند شتاب و پیچش به چپ و راست و ترمز است. اشیاء در سی شارپ تقلیدی از یک شیء مانند ماشین در دنیای واقعی هستند. برنامه نویسی شیء گرا با استفاده از کدهای دسته بندی شده کلاسها و اشیاء را بیشتر قابل کنترل می کند.

در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاءمان داریم. شیء در برنامه نویسی شیء گرا از روی کلاسی که شما تعریف کرده اید ایجاد می شود. برای مثال نقشه ساختمان شما یک کلاس است که ساختمان از روی آن ساخته شده است. کلاس شامل خواص یک ساختمان مانند مساحت، بلندی و مواد مورد استفاده در ساخت خانه می باشد. در دنیای واقعی ساختمانها نیز بر اساس یک نقشه (کلاس) پایه گذاری (تعریف) شده اند.

برنامه نویسی شیء گرا یک روش جدید در برنامه نویسی است که بوسیله برنامه نویسان مورد استفاده قرار می گیرد و به آنها کمک می کند که برنامه هایی با قابلیت استفاده مجدد، خوانا و راحت طراحی کنند. جاوا اسکریپت نیز یک برنامه شیء گراست. در درس های بعد، به شما نحوه تعریف کلاس و استفاده از اشیاء آموزش داده خواهد شد. همچنین شما با دو مفهوم وراثت و چند ریختی که از مباحث مهم در برنامه نویسی شیء گرا هستند در آینده آشنا می شوید.

کلاس

کلاس به شما اجازه می دهد یک نوع داده ای که توسط کاربر تعریف می شود و شامل متغیرها و خواص (properties) و متدها است را ایجاد کنید. کلاس در حکم یک نقشه برای یک شیء می باشد. شیء یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی می کند. وقتی یک شیء می سازید یعنی اینکه یک نمونه از کلاس ساخته اید (در درس ممکن است از کلمات شیء و نمونه به جای هم استفاده شود). برای تعریف یک کلاس از کلمه کلیدی class استفاده شود :

```
class ClassName
{
    Variable1;
    Variable2;
    ...
    VariableN;
```

```

method1;
method2;
...
methodN;
}

```

این کلمه کلیدی را قبل از نامی که برای کلاس انتخاب می‌کنیم می‌نویسیم. در نامگذاری کلاس‌ها هم از روش نامگذاری Pascal استفاده می‌کنیم. در بدنه کلاس فیلدها و متدهای آن قرار داده می‌شوند.

به متغیرهایی که در داخل کلاس تعریف می‌شوند فیلد، و به توابعی که داخل کلاس تعریف می‌شوند، متد گفته می‌شود.

کلاس از فیلدها برای رفتارها و ذخیره مقادیر خاصیت‌هایش (property) استفاده می‌کند. متدها رفتارها یا کارهایی هستند که یک کلاس می‌تواند انجام دهد. در زیر نحوه تعریف و استفاده از یک کلاس ساده به نام person نشان داده شده است:

```

1 class Person
2 {
3     name;
4     age;
5     height;
6
7     TellInformation()
8     {
9         console.log("Name : ", this.name);
10        console.log("Age : ", this.age , " years old");
11        console.log("Height: ", this.height , "cm");
12    }
13 }
14
15 let firstPerson = new Person();
16 let secondPerson = new Person();
17
18 firstPerson.name = "Jack";
19 firstPerson.age = 21;
20 firstPerson.height = 160;
21 firstPerson.TellInformation();
22
23 console.log("\n"); //Separator
24
25 secondPerson.name = "Mike";
26 secondPerson.age = 23;
27 secondPerson.height = 158;
28 secondPerson.TellInformation();

```

```

Name : Jack
Age : 21 years old
Height: 160 cm

```

```

Name : Mike
Age : 23 years old
Height: 158 cm

```

همانطور که در کد بالا مشاهده می‌کنید، در خطوط 1-13 کلاسی به نام Person تعریف شده است. در خط 1 یک نام به کلاس اختصاص داده‌ایم تا به وسیله آن قابل دسترسی باشد. در داخل بدنه کلاس فیلدهای آن تعریف شده‌اند (خطوط 3-5). این سه فیلد تعریف شده خصوصیات واقعی یک فرد در دنیای واقعی را در خود ذخیره می‌کنند. یک فرد در دنیای واقعی دارای نام، سن و قد می‌باشد. در خطوط 7-12 یک متد هم در داخل کلاس به نام TellInformation() تعریف شده است که رفتار کلاس‌مان است و مثلاً اگر از فرد سوالی بپرسیم در مورد خودش چیزهایی می‌گوید. در داخل متد کدهایی برای نشان دادن مقادیر موجود در فیلدها نوشته شده است. نکته‌ای درباره فیلدها وجود دارد و این است که چون فیلدها در داخل کلاس تعریف و به عنوان اعضای کلاس در نظر گرفته شده‌اند، محدوده آنها یک کلاس است. این بدین معناست که فیلدها فقط می‌توانند در داخل کلاس یعنی جایی که به آن تعلق دارند و یا به وسیله نمونه ایجاد شده از کلاس مورد استفاده قرار بگیرند.

همانطور که در کد بالا مشاهده می‌کنید، برای تعریف فیلدها از کلمه var و برای تعریف متدها از کلمه function استفاده نمی‌کنیم.

در خطوط 16 و 17 دو نمونه یا دو شیء از کلاس Person ایجاد می‌کنیم. برای ایجاد یک نمونه از یک کلاس باید از کلمه کلیدی new و به دنبال آن نام کلاس و یک جفت پرانتز قرار دهیم :

```
16 let firstPerson = new Person();
17 let secondPerson = new Person();
```

در خطوط 18-20 مقادیری به فیلدهای اولین شیء ایجاد شده از کلاس person1 اختصاص داده شده است. برای دسترسی به فیلدها یا متدهای یک شیء از علامت . استفاده می‌شود. به عنوان مثال کد person1.name نشان دهنده متغیر name از شیء person1 می‌باشد. برای چاپ مقادیر فیلدها باید متد TellInformation() شیء person1 را فراخوانی می‌کنیم (خط 21). به کلمه کلیدی this در خطوط 9-11 توجه کنید. این کلمه کلیدی اشاره به شیء جاری دارد. یعنی وقتی مقادیر از طریق شیء person1 ارسال می‌شوند، منظور از this شیء person1 و وقتی از طریق شیء person2 ارسال می‌شوند منظور از this شیء person2 می‌باشد. در خطوط 28-30 نیز مقادیری به شیء دومی که قبلاً از کلاس ایجاد شده تخصیص می‌دهیم و سپس متد TellInformation() را فراخوانی می‌کنیم. به این نکته توجه کنید که person1 و person2 نسخه‌های متفاوتی از هر متغیر دارند بنابراین تعیین یک نام برای person2 هیچ تاثیری بر نام person1 ندارد. در مورد دیگر اعضای کلاس در درسهای آینده توضیح خواهیم داد.

سازنده

سازنده‌ها متدهای خاصی هستند که وجود آنها برای ساخت اشیاء لازم است. آنها به شما اجازه می‌دهند که فیلدهای کلاس را مقداردهی اولیه کنید و کدهایی که را که می‌خواهید هنگام ایجاد یک شیء اجرا شوند را به برنامه اضافه کنید. اگر از هیچ سازنده‌ای در کلاستان استفاده نکنید، جاوااسکریپت از سازنده پیشفرض که یک متد بدون پارامتر است استفاده می‌کند. برای ایجاد سازنده از متد خاص constructor() استفاده می‌کنیم. در مثال زیر یک کلاس که شامل سازنده پیشفرض (خطوط 7-10) است را مشاهده می‌کنید :

```

1 class Person
2 {
3     name;
4     age;
5     height;
6
7     constructor()
8     {
9
10    }
11
12    TellInformation()
13    {
14        console.log("Name : ", this.name);
15        console.log("Age : ", this.age);
16        console.log("Height: ", this.height);
17    }
18 }
19
20 var person1 = new Person();
21 console.log(person1);

```

```
Person {name: undefined, age: undefined, height: undefined}
```

می‌توانیم این سازنده را هم تعریف نکنیم چون جاوااسکریپت به طور خودکار آن را ایجاد می‌کند. همانطور که در خط 20 و 21 مشاهده می‌کنید ما یک شیء یا یک نمونه از کلاس ایجاد کرده‌ایم (در درس بعد بیشتر توضیح می‌دهیم) و با استفاده از تابع `log()` مقادیر موجود در این شیء را چاپ کرده‌ایم. در خروجی مشاهده می‌کنید که سازنده پیشفرض به هر سه فیلد مقدار `undefined` را اختصاص داده است. مثلاً بهتر است که با استفاده از سازنده مقدار پیشفرض به فیلدها اختصاص دهیم. مثلاً فردی که به دنیا می‌آید نام (`name`) ندارد ولی سن (`age`) و قد (`height`) دارد. پس می‌توانیم به صورت زیر این مقادیر را به فیلدها با استفاده از سازنده پیشفرض اختصاص دهیم :

```

1 class Person
2 {
3     name;
4     age;
5     height;
6
7     constructor()
8     {
9         this.name = "";
10        this.age = 9 ;
11        this.height = 30;
12    }
13
14    TellInformation()
15    {
16        console.log("Name : ", this.name);
17        console.log("Age : ", this.age);
18        console.log("Height: ", this.height);
19    }
20 }
21
22 var person1 = new Person();

```

```
23 person1.TellInformation();
```

```
Name :
Age   : 9 Month
Height: 30 cm
```

حال فرض کنید می‌خواهیم سازنده‌ای ایجاد کنیم که بعد از ایجاد یک شیء از کلاس، فیلدهای شیء ایجاد شده را خودمان و با استفاده از سازنده‌ای که تعریف کرده‌ایم مقداردهی کنیم. به کد زیر توجه کنید :

```
1 class Person
2 {
3     name;
4     age;
5     height;
6
7     constructor(n, a, h)
8     {
9         this.name = n;
10        this.age = a;
11        this.height = h;
12    }
13
14    TellInformation()
15    {
16        console.log("Name : ", this.name);
17        console.log("Age : ", this.age);
18        console.log("Height: ", this.height);
19    }
20 }
21
22 var person1 = new Person("Jack", 21, 160);
23 person1.TellInformation();
24
25 console.log("\n");
26
27 var person2 = new Person("Mike", 32, 158);
28 person2.TellInformation();
```

```
Name : Jack
Age   : 21
Height: 160
```

```
Name : Mike
Age   : 32
Height: 158
```

همانطور که مشاهده می‌کنید در مثال بالا سازنده‌ای را سه آرگومان قبول می‌کند به کلاس Person اضافه کرده‌ایم (خطوط 7-12). در خطوط 22 و 27 بعد از ایجاد شیء و در داخل پرانتزها سه مقدار را به سازنده (خط 7) ارسال می‌کنیم و سازنده این مقادیر را به فیلدها (خطوط 3-5) اختصاص می‌دهد.

سطح دسترسی

سطح دسترسی مشخص می‌کند که متدهای یک کلاس یا فیلدهای آن، در چه جای برنامه قابل دسترسی هستند. در جاوااسکریپت سه

سطح دسترسی وجود دارد :

- public (عمومی)
- Private (خصوصی)
- Protect (محافظت شده)

در این درس می‌خواهیم به سطح دسترسی private و public نگاهی بیندازیم. سطح دسترسی public زمانی مورد استفاده قرار

می‌گیرد که شما بخواهید به یک متد یا فیلد در خارج از کلاس دسترسی یابید. به عنوان مثال به کد زیر توجه کنید :

```

1 class Test
2 {
3     number1 = 10;
4     #number2 = 20;
5 }
6
7 var x = new Test();
8
9 console.log(x.number1);
10 console.log(x.#number2);

```

```
Uncaught SyntaxError: Private field '#number2' must be declared in an enclosing class
```

در جاوااسکریپت، در حالت پیشفرض، متدها و فیلدها دارای سطح دسترسی عمومی یا public هستند. در این مثال یک کلاس با نام Test تعریف کرده‌ایم. سپس دو فیلد، یکی به صورت public (خط 3) و دیگری به صورت private در داخل کلاس Test تعریف می‌کنیم (خط 4).

همانطور که در خط 3 مشاهده می‌کنید، برای تعریف یک فیلد یا متد به صورت خصوصی یا private کافیست که قبل از نام آن از علامت # استفاده کنید.

در خط 7 یک نمونه از کلاس ایجاد کرده و در خطوط 9 و 10 سعی می‌کنیم که مقدار فیلدهای آن را چاپ کنیم. همانطور که مشاهده می‌کنید برنامه با خطا مواجه می‌شود. دلیل آن هم این است که اعضای private در خارج از کلاس قابل دسترسی نیستند. حال خط 10 را حذف و دوباره برنامه را اجرا کنید. خروجی عدد 10 را نشان می‌دهد. چون مقدار فیلد number1، عدد 10 بود و این فیلد به صورت عمومی یا public تعریف شده و در خارج از کلاس قابل دسترسی می‌باشد. نکته ای که در خط 10 وجود دارد این است که برای دسترسی به اعضای private باید علامت # را قبل از نام آنها بنویسید. به طور کلی فیلدها و متدهایی که به صورت public تعریف می‌شوند در داخل کلاس و نمونه‌های ایجاد شده از آن قابل دسترسی و فیلدها و متدهایی که به صورت private تعریف می‌شوند فقط در داخل

کلاس قابل دسترسی هستند و برای دسترسی به آنها در خارج از کلاس باید از خاصیت‌ها استفاده کرد که در درس بعد توضیح می‌دهیم. سطح دسترسی protect را بعد از مبحث وراثت در درسهای آینده آموزش می‌دهیم.

کپسوله سازی

کپسوله سازی یا Encapsulation یا مخفی کردن اطلاعات، فرایندی است که طی آن اطلاعات حساس یک موضوع از دید کاربر مخفی می‌شود و فقط اطلاعاتی که لازم باشد برای او نشان داده می‌شود.

وقتی که یک کلاس تعریف می‌کنیم معمولاً تعدادی فیلد برای ذخیره مقادیر مربوط به شیء نیز تعریف می‌کنیم. برخی از این فیلدها توسط خود کلاس برای عملکرد متدها و برخی دیگر از آنها به عنوان یک فیلد موقت به کار می‌روند. لازم نیست که کاربر به تمام فیلدها یا متدهای کلاس دسترسی داشته باشد. اینکه فیلدها را طوری تعریف کنیم که در خارج از کلاس قابل دسترسی باشند بسیار خطرناک است چون ممکن است کاربر رفتار و نتیجه یک متد را تغییر دهد. به برنامه ساده زیر توجه کنید:

```

1 class Test
2 {
3     five = 5;
4
5     AddFive(number)
6     {
7         this.five += number;
8         return this.five;
9     }
10 }
11
12 var test = new Test();
13
14 test.five = 10;
15 console.log(test.AddFive(100))

```

110

متد داخل کلاس Test به نام AddFive() (خطوط 5-9) دارای هدف ساده‌ای است و آن اضافه کردن مقدار 5 به هر عدد می‌باشد (همانطور که از اسم متد پیداست AddFive). در خط 12 یک نمونه از کلاس Test ایجاد کرده‌ایم و مقدار فیلد آن را در خط 14 از 5 به 10 تغییر می‌دهیم (در اصل نباید تغییر کند چون ما از برنامه خواسته‌ایم هر عدد را با 5 جمع کند ولی کاربر به راحتی آن را به 10 تغییر می‌دهد). همچنین متد AddFive() را در خط 15 فراخوانی و مقدار 100 را به آن ارسال می‌کنیم. مشاهده می‌کنید که قابلیت متد AddFive() به خوبی تغییر می‌کند و شما نتیجه متفاوتی مشاهده می‌کنید. اینجاست که اهمیت کپسوله سازی مشخص می‌شود. اینکه ما در درسهای قبلی فیلدها را به صورت public تعریف کردیم و به کاربر اجازه دادیم که در خارج از کلاس به آنها دسترسی داشته باشد کار اشتباهی بود. فیلدها باید همیشه به صورت private تعریف شوند.

خواص (Properties)

Property (خصوصیت) استاندارد در جاوااسکریپت، برای دسترسی به فیلدها با سطح دسترسی private در داخل یک کلاس می‌باشد. همانطور که در درس قبل اشاره شد، تعریف فیلدها در داخل کلاس به صورت public اشتباه است، چون کاربران می‌توانند با ایجاد یک شیء از کلاس به آنها دسترسی داشته باشند و هر مقداری که دوست دارند به آنها اختصاص دهند. برای رفع این مشکل مفهوم property ارائه شد. هر property دارای دو بخش می‌باشد، یک بخش جهت مقدار دهی (بلوک set) و یک بخش برای دسترسی به مقدار (بلوک get) یک داده private می‌باشد. property ها باید به صورت public تعریف شوند تا در کلاسهای دیگر نیز قابل دسترسی می‌باشند. در مثال زیر نحوه تعریف و استفاده از property آمده است :

```

1  class Person
2  {
3      #name ;
4      #age ;
5      #height;
6
7      get Name()
8      {
9          return this.#name;
10     }
11     set Name(value)
12     {
13         this.#name = value;
14     }
15
16     get Age()
17     {
18         return this.#age;
19     }
20     set Age(value)
21     {
22         this.#age = value;
23     }
24
25     get Height()
26     {
27         return this.#height;
28     }
29     set Height(value)
30     {
31         this.#height = value;
32     }
33
34     constructor(name, age, height)
35     {
36         this.#name = name;
37         this.#age = age;
38         this.#height = height;
39     }
40 }
41
42 var person1 = new Person("Jack", 21, 160);
43 var person2 = new Person("Mike", 23, 158);

```

```

44
45 console.log("Name : ", person1.Name);
46 console.log("Age : ", person1.Age , "years old" );
47 console.log("Height: ", person1.Height , "cm");
48
49 console.log("\n"); //Seperator
50
51 console.log("Name : ", person2.Name);
52 console.log("Age : ", person2.Age , "years old" );
53 console.log("Height: ", person2.Height , "cm");
54
55 console.log("\n"); //Seperator
56
57 person1.Name = "Frank";
58 person1.Age = 19;
59 person1.Height = 162;
60
61 person2.Name = "Ronald";
62 person2.Age = 25;
63 person2.Height = 174;
64
65 console.log("Name : ", person1.Name);
66 console.log("Age : ", person1.Age , "years old" );
67 console.log("Height: ", person1.Height , "cm");
68
69 console.log("\n"); //Seperator
70
71 console.log("Name : ", person2.Name);
72 console.log("Age : ", person2.Age , "years old" );
73 console.log("Height: ", person2.Height , "cm");

```

```

Name : Jack
Age : 21 years old
Height: 160cm

Name : Mike
Age : 23 years old
Height: 158cm

Name : Frank
Age : 19 years old
Height: 162cm

Name : Ronald
Age : 25 years old
Height: 174cm

```

در برنامه بالا نحوه استفاده از property آمده است. همانطور که مشاهده می‌کنید در این برنامه ما سه فیلد (خطوط 3-5) تعریف کرده‌ایم (سه فیلد با سطح دسترسی private).

```

#name ;
#age ;
#height;

```

دسترسی به مقادیر این فیلدها فقط از طریق property های ارائه شده (خطوط 7-32) امکان پذیر است.

```

get Name()
{
    return this.#name;
}
set Name(value)
{
    this.#name = value;
}

get Age()
{
    return this.#age;
}
set Age(value)
{
    this.#age = value;
}

get Height()
{
    return this.#height;
}
set Height(value)
{
    this.#height = value;
}

```

به این نکته توجه کنید که طبق قرارداد، نام property ها همانند نام فیلدهای مربوطه می‌باشد با این تفاوت که حرف اول آنها بزرگ نوشته می‌شود. تاکید می‌کنیم که، شباهت نام property ها و فیلدها اجبار نیست و یک قرارداد می‌باشد.

هر property دارای دو بخش می‌باشد، یکی بخش set و دیگری بخش get. بخش set، که با کلمه کلیدی set نشان داده شده است برای مقداردهی به فیلدها (اعضای داده‌ای) به کار می‌رود. بخش get، که با کلمه کلیدی get نشان داده شده است به شما اجازه می‌دهد که یک مقدار را از فیلدها (اعضای داده‌ای) استخراج کنید.

به کلمه value در داخل بلوک set توجه کنید. Value همان مقداری است که از طریق property به فیلد اختصاص می‌دهیم. برای اختصاص یک مقدار به یک فیلد از طریق property کافیسست که به صورت زیر عمل کنید :

```
Object.Property = Value;
```

این کار (قرار دادن یک مقدار بعد از علامت مساوی) به منزله فراخوانی بخش set است. و ما به برنامه می‌فهمانیم که می‌خواهیم از طریق بخش set یک فیلد را مقداردهی کنیم. Object شیء ایجاد شده از کلاس، Property نام پراپرتی و Value مقداری است که می‌خواهیم به فیلد اختصاص دهیم. برای دسترسی به یک خاصیت می‌توانید از علامت دات (.) استفاده کنید. مثلاً برای اختصاص مقدار به سه فیلد age، name و height از طریق property باید به صورت زیر عمل کنید :

```
person1.Name = "Frank";
person1.Age = 19;
person1.Height = 162;
```

دستورات بالا بخش set مربوط به هر property را فراخوانی کرده و مقادیری به هر یک از فیلدها اختصاص می‌دهد. برای فراخوانی بخش get کافیست که نام شیء و سپس علامت نقطه و در آخر نام property را بنویسیم. با این کار به برنامه می‌فهمانیم که ما نیاز به مقدار فیلد داریم.

```
console.log("Name : ", person1.Name);
console.log("Age : ", person1.Age , "years old" );
console.log("Height: ", person1.Height , "cm");
```

استفاده از property ها کد نویسی را انعطاف پذیر می‌کند مخصوصاً اگر بخواهید یک اعتبارسنجی برای اختصاص یک مقدار به فیلدها یا استخراج یک مقدار از آنها ایجاد کنید. پس می‌توان گفت که :

کاربرد اصلی property ها، اعتبار سنجی مقادیری است که کاربر می‌خواهد به فیلدها اختصاص دهد.

مثلاً در مثال زیر شما می‌توانید یک محدودیت ایجاد کنید که فقط اعداد مثبت به فیلد age (سن) اختصاص داده شود. می‌توانید با تغییر بخش set خاصیت Age این کار را انجام دهید :

```
set Age(value)
{
  if(value > 0 && value < 100)
  {
    this.#age = value;
  }
  else
  {
    return 0;
  }
}
```

حال اگر کاربر بخواهد یک مقدار منفی به فیلد age اختصاص دهد مقدار age صفر خواهد شد. همچنین می‌توان یک property فقط خواندنی (read-only) ایجاد کرد. این property فاقد بخش set است. به عنوان مثال می‌توان یک خاصیت Name فقط خواندنی مانند زیر ایجاد کرد :

```
get Age()
{
  return this.#age;
}
```

در این مورد اگر بخواهید یک مقدار جدید به فیلد name اختصاص دهید با خطا مواجه می‌شوید. یک property می‌تواند دارای دو فیلد باشد. به کد زیر توجه کنید :

```
#firstName;
#lastName;

get FullName() { return firstName + " " + lastName; }
```

همانطور که در مثال بالا مشاهده می‌کنید یک property فقط خواندنی تعریف کرده‌ایم که مقدار برگشتی آن ترکیبی از دو فیلد firstName و lastName است که به وسیله فاصله از هم جدا شده‌اند.

وراثت

وراثت به یک کلاس اجازه می‌دهد که خصوصیات یا متدهایی را از کلاس دیگر به ارث برد. وراثت مانند رابطه پدر و پسر می‌ماند به طوری که فرزند خصوصیتی از قبیل قیافه و رفتار را از پدر خود به ارث برده باشد.

کلاس پایه یا کلاس والد کلاسی است که بقیه کلاسها از آن ارث می‌برند.

کلاس مشتق یا کلاس فرزند کلاسی است که از کلاس پایه ارث بری می‌کند.

همه متد و خصوصیات کلاس پایه می‌توانند در کلاس مشتق مورد استفاده قرار بگیرند به استثنای اعضا و متدهای با سطح دسترسی private. مفهوم اصلی وراثت در مثال زیر نشان داده شده است :

```
1 class classParent
2 {
3
4     #privateField = "This is private Filed From Parent Class!";
5
6
7     publicMessage()
8     {
9         console.log("This is public Message From Parent Class!");
10    }
11 }
12
13 class classChild extends classParent
14 {
15
16 }
17
18 var child = new classChild();
19
20 child.publicMessage();
21 //child.#privateField; Error : Private field '#privateField' must be declared in an
    enclosing class
```

This is public Message From Parent Class!

همانطور که مشاهده می‌کنید در کد بالا دو کلاس تعریف کرده‌ایم: یکی کلاس classParent (خطوط 1-11) که دارای یک فیلد با سطح دسترسی private و یک متد با سطح دسترسی public است و کلاس دیگر (خطوط 13-16) که در بدنه خود هیچ متد یا متغیری ندارد. نحوه ارث بری یک کلاس به صورت زیر است :

```
class Child extends Parent
```

که در خط 13 مشخص کرده‌ایم که کلاس classChild قرار است از کلاس classParent ارث بری کند:

```
class classChild extends classParent
```

در خط 13 یک نمونه از کلاس فرزند ایجاد می‌کنیم و در خط 20 متد کلاس پدر را فراخوانی می‌کنیم. همانطور که در خروجی مشاهده می‌کنید متدی که دارای سطح دسترسی public است فراخوانی و اجرا شده و پیغام چاپ می‌شود. حال اگر خط 21 را از حالت توضیحات خارج کنیم، یعنی علامت // را حذف و برنامه را دوباره اجرا کنیم با خطا مواجه می‌شویم. چون دسترسی به اعضای private در خارج از کلاس مربوطه شان و یا توسط کلاس مشتق شده، امکان پذیر نیست. در پایان یادآور می‌شویم که کلاس فرزند (خطوط 13-16) هیچ متد یا متغیری در بدنه خود ندارد، ولی چون از کلاس classParent ارث بری کرده است متد با سطح دسترسی public آن را می‌تواند برای خود داشته باشد.

متد () super و کلمه کلیدی super

از متد () super زمانی که بخواهید سازنده کلاس پدر را فراخوانی کرده و از کدهای آن استفاده کنیم، استفاده می‌شود. به مثال زیر توجه کنید :

```
1 class Parent
2 {
3     constructor()
4     {
5         console.log("Message from parent class!");
6     }
7 }
8
9 class Child extends Parent
10 {
11     constructor()
12     {
13         super();
14     }
15 }
16
17 var child = new Child();
```

```
Message from parent class!
```

به این نکته توجه کنید که حتی اگر کلاس پدر دارای هیچ سازنده ای هم نباشد، اگر یک کلاس از آن ارث بری کرد، باز هم باید در سازنده کلاس فرزند از متد `super()` استفاده کنید:

```

1 class Parent
2 {
3
4 }
5
6 class Child extends Parent
7 {
8     constructor()
9     {
10        super();
11        console.log("Message From Child Class!");
12    }
13 }
14
15 var child = new Child();

```

```
Message From Child Class!
```

اگر متد `super()` فرموش شود با خطای `ReferenceError: Must call super constructor in derived class` مواجه می شوید. از کلمه کلیدی `super` زمانی که بخواهید از یک فیلد یا متد کلاس پدر در داخل سازنده کلاس فرزند استفاده کنید، استفاده می شود. به مثال زیر توجه کنید:

```

1 class Parent
2 {
3     constructor()
4     {
5         console.log("Message from parent class!");
6     }
7
8     ShowMessage()
9     {
10        console.log("Hello World!");
11    }
12 }
13
14 class Child extends Parent
15 {
16     constructor()
17     {
18        super();
19        super.ShowMessage();
20    }
21 }
22
23 var child = new Child();

```

```
Message from parent class!
Hello World!
```

override

override یا باز نویسی، یعنی اینکه ما کاری کنیم که متدهای کلاس پایه در داخل کلاس مشتق رفتار متفاوتی از خود نشان دهند. به عنوان مثال شما متد A را در کلاس A دارید و کلاس B از کلاس A ارث بری می‌کند، در این صورت متد A در کلاس B در دسترس خواهد بود. اما متد A دقیق همان متدی است که از کلاس A به ارث برده شده است. حال اگر بخواهید که این متد رفتار متفاوتی از خود نشان دهد چکار می‌کنید؟ Overriding یا بازنویسی این مشکل را برطرف می‌کند. به تکه کد زیر توجه کنید :

```

1 class Person
2 {
3     ShowMessage()
4     {
5         console.log("Message from Parent.");
6     }
7 }
8
9 class Child extends Person
10 {
11     ShowMessage()
12     {
13         super.ShowMessage();
14         console.log("ShowMessage method was overridden !");
15     }
16 }
17
18 var myPerson = new Person();
19 var myChild = new Child();
20
21 myPerson.ShowMessage();
22 console.log("\n");
23 myChild.ShowMessage();

```

```
Message from Parent.
```

```
Message from Parent.
```

```
ShowMessage method was overridden !
```

همانطور که در کد بالا مشاهده می‌کنید، یک متد به نام ShowMessage (خطوط 3-6) در کلاس Person تعریف شده است که یک پیغام چاپ می‌کند. حال می‌خواهیم این متد در کلاس Child علاوه بر این پیغام ShowMessage method was overridden! را نیز چاپ کند. برای این کار همانطور که مشاهده می‌کنید همین متد را در خطوط (11-15) و در داخل کلاس Child می‌نویسیم و سپس با استفاده از کلمه کلیدی super در خط 13 به جاوااسکریپت اعلام می‌کنیم که قصد استفاده از تمام کدهای بدنه همین متد در کلاس مادر را داریم بعلاوه اینکه در خط بعد از این دستور یعنی خط 14 کدهای اضافی را که قرار است این متد در کلاس فرزند یعنی Child داشته باشد می‌نویسیم. شاید این کار برای متدی به این سادگی زیاد کارا نباشد، اما اگر متد کلاس مادر دارای کدهای زیادی در بدنه خود باشد و شما بخواهید کد دیگری در کلاس فرزند به آن اضافه کنید استفاده از این روش کدنویسی را بهینه و ساده‌تر می‌کند.

عملگر instanceof

عملگر instanceof در جاوااسکریپت، به شما اجازه می‌دهد که تست کنید که آیا یک شیء یک نمونه از یک نوع خاص (کلاس، زیر کلاس، ...) است یا نه. عملگر instanceof به دو عملوند نیاز دارد و یک مقدار بولی را برمی‌گرداند. به عنوان مثال، فرض کنید یک کلاس به نام Animal داریم، سپس یک نمونه از آن ایجاد می‌کنیم :

```
class Animal
{
}

var myAnimal = new Animal();

if (myAnimal instanceof Animal)
{
    console.log("myAnimal is an Animal!");
}
```

```
myAnimal is an Animal!
```

رفتار عملگر instanceof را در این مثال مشاهده کردید. همانطور که می‌بینید از آن به عنوان شرط در عبارت if استفاده شده است. کاربرد آن در مثال بالا این است که چک می‌کند که آیا شیء myAnimal یک نمونه از Animal است و چون نتیجه درست است کدهای داخل دستور if اجرا می‌شود. این عملگر همچنین می‌تواند چک کند که آیا یک شیء خاص در سلسله مراتب وراثت یک نوع خاص است. به این مثال توجه کنید :

```
class Animal
{
}

class Dog extends Animal
{
}

var mydog = new Dog();

if (mydog instanceof Animal)
{
    console.log("mydog is an Animal!");
}
```

```
myDog is an Animal!
```

همانطور که در مثال بالا می‌بینید ما یک کلاس به نام Dog ایجاد کرده‌ایم که از کلاس Animal ارث می‌برد. سپس یک نمونه از این کلاس (Dog) ایجاد می‌کنیم و سپس با استفاده از عملگر instanceof تست می‌کنیم که آیا نمونه ایجاد شده جز کلاس Animal است یا یک

کلاس مشتق شده از کلاس Animal می‌باشد. از آنجاییکه کلاس Dog از کلاس Animal ارث می‌برد (سگ من یک حیوان است)، نتیجه عبارت درست (true) است. حال جمله بالا را تغییر دهیم: "حیوان من یک سگ است". وقتی جمله برعکس می‌شود چه اتفاقی می‌افتد؟

```
var myAnimal = new Animal();
if (myAnimal instanceof Dog)
{
    console.log("myAnimal is a Dog!");
}
```

این باعث خطا نمی‌شود، ولی هیچ پیغامی را نیز نمایش نمی‌دهد.

اعضای Static

اگر بخواهیم عضو داده‌ای (فیلد) یا خاصیتی ایجاد کنیم که در همه نمونه‌های کلاس قابل دسترسی باشد از کلمه کلیدی static استفاده می‌کنیم. کلمه کلیدی static برای اعضای داده‌ای و خاصیت‌هایی به کار می‌رود که می‌خواهند در همه نمونه‌های کلاس تقسیم شوند. وقتی که یک متد یا خاصیت به صورت static تعریف شود، می‌توانید آنها را بدون ساختن نمونه‌ای از شیء، فراخوانی کنید. برای فراخوانی یک عضو استاتیک ابتدا نام کلاس سپس علامت نقطه (.) و در آخر نام عضو استاتیک را می‌نویسید :

```
Class Name.Static Member
```

به مثالی در مورد متدها و خاصیت‌های static توجه کنید :

```
1 class SampleClass
2 {
3     static ShowStaticMessage()
4     {
5         console.log("static method has been called.");
6     }
7 }
8
9 SampleClass.ShowStaticMessage();
```

```
static method has been called.
```

همانگونه که در کد بالا مشاهده می‌کنید، یک کلاس (خطوط 1-7) تعریف کرده‌ایم که دارای یک متد (خطوط 3-6) از نوع استاتیک می‌باشد. برای دسترسی به متد در خارج از کلاس، لازم نیست که از کلاس نمونه ایجاد کنیم و فقط کافیست که نام کلاس را نوشته و بعد از آن علامت دو نقطه و در آخر نام عضو استاتیک را بنویسید مانند خط 9 مثال بالا.

مدیریت استثناءها و خطایابی

بهترین برنامه نویسان در هنگام برنامه نویسی با خطاها و باگها در برنامه‌شان مواجه می‌شوند. درصد زیادی از برنامه‌ها هنگام تست برنامه با خطا مواجه می‌شوند. بهتر است برای از بین بردن یا به حداقل رساندن این خطاها، به کاربر در مورد دلایل به وجود آمدن آنها اخطار داده شود. خوشبختانه جاوااسکریپت برای این مشکل راه حلی ارائه داده است. استثناءها در جاوااسکریپت راهی برای نشان دادن دلیل وقوع خطا در هنگام اجرای برنامه است.

جاوااسکریپت دارای مجموعه‌ای از کلاسهای استثناء است که شما می‌توانید با استفاده از آنها خطاهایی که در موقعیت‌های مختلف روی می‌دهند را برطرف کنید. حتی می‌توانید یک کلاس استثناء شخصی ایجاد کنید. استثناءها توسط برنامه به وجود می‌آیند و شما لازم است که آنها را اداره کنید. به عنوان مثال اگر در جاوااسکریپت بخواهید از متغیری استفاده کنید که قبلاً آن را تعریف نکرده اید با خطای `ReferenceError: number is not defined` مواجه می‌شوید.

باگ (Bug) اصطلاحاً خطا یا کدی است که رفتارهای ناخواسته‌ای در برنامه ایجاد می‌کند. خطایابی فرایند برطرف کردن باگها است، بدین معنی که خطاها را از برنامه پاک کنیم. جاوااسکریپت دارای ابزارهایی برای خطایابی می‌باشد، که خطاها را یافته و به شما اجازه می‌دهند آنها را برطرف کنید. در درسهای آینده خواهید آموخت که چگونه از این ابزارهای کارآمد جهت برطرف کردن باگها استفاده کنید. قبل از اینکه برنامه را به پایان برسانید لازم است که برنامه‌تان را اشکال زدایی کنید.

دستورات try و catch

در این درس می‌خواهیم دستورات `try...catch` و نحوه استفاده از آنها برای رفع خطاهای برنامه، را به شما آموزش دهیم. به کد زیر توجه کنید:

```
<script>
  Number
</script>
```

اگر برنامه بالا را اجرا کنید با خطای `ReferenceError: number is not defined` مواجه می‌شوید. چون برنامه نمی‌داند با کلمه `number` چگونه رفتار کند. ما یک کلمه به نام `number` نوشته ایم و نه ذکر کرده ایم که این یک متغیر است و نه مقداری به آن اختصاص داده ایم. اینجاست که ما با یک استثناء مواجه می‌شویم و باید آن را مدیریت کنیم. می‌توان خطا را با استفاده از دستور `try...catch` اداره کرد. بدین صورت که کدی را که احتمال می‌دهید ایجاد خطا کند را در داخل بلوک `try` قرار می‌دهید. بلوک `catch` هم شامل کدهایی است که وقتی اجرا می‌شوند که برنامه با خطا مواجه شود. تعریف ساده‌ی این دو بلوک به این صورت است که بلوک `try` سعی می‌کند که دستورات را اجرا کند و اگر در بین دستورات خطایی وجود داشته باشد برنامه دستورات مربوط به بخش `catch` را اجرا می‌کند. برنامه زیر نحوه استفاده از دستور `try...catch` را نمایش می‌دهد:

```
try
{
    number;
}
catch
{
    console.log("This word is not defined");
}
```

This word is not defined

از آنجاییکه در برنامه بالا خطایی به وجود آمده است کدهای داخل بلوک catch اجرا می‌شوند. بنابراین :

```
try
{
    number;
    console.log("This line will not be executed.");
}
catch
{
    console.log("This word is not defined");
}
```

می‌توانید از یک نوع استثناء مخصوص به یک خطا در داخل بلوک catch استفاده کنید، مثلاً برای خطای بالا می‌توان از یکی از کلاس‌های خطا در جاوااسکریپت به نام err به شکل زیر استفاده کنید :

```
try
{
    number;
}
catch(err)
{
    console.log(err.name);
}
```

ReferenceError

خاصیت name مربوط به کلاس err نوع خطا را مشخص می‌کند. همچنین می‌توانید اطلاعات مربوط به این استثناء را با استفاده از خاصیت message نمایش دهید:

```
try
{
    number;
}
catch(err)
{
    console.log(err.message);
}
```

number is not defined

شما می‌توانید از عمگر instanceof نیز به صورت زیر استفاده نمایید:

```

try
{
  eval("a ++ b");
}
catch(err)
{
  if (err instanceof ReferenceError)
  {
    console.log("value is not defined");
  }
  if (err instanceof TypeError)
  {
    console.log("Format cannot be accepted!");
  }
  if (err instanceof SyntaxError)
  {
    console.log("Causes SyntaxError");
  }
}

```

Causes SyntaxError

بلوک catch از کلاس err برای به دام انداختن همه استثناءهایی که به وسیله برنامه به وجود می‌آید استفاده می‌کند. در داخل بلوک catch می‌توانید با استفاده از یک دستور if و کلمه کلیدی instanceof نوع استثناء به وجود آمده را بیابید.

استفاده از بلوک finally

گاهی اوقات می‌خواهید برخی کدها همیشه اجرا شوند خواه استثناء رخ دهد، خواه رخ ندهد، در این صورت از بلوک finally استفاده می‌شود. قبلاً یاد گرفتیم که اگر در بلوک try یک استثناء رخ دهد همه کدهای موجود در این بلوک نادیده گرفته شده و برنامه به قسمت catch می‌رود. کدهای نادیده گرفته شده ممکن است در برنامه نقش حیاتی داشته باشند.

هدف بلوک finally هم حفظ نقش این کدها به صورت غیر مستقیم است. کدهایی را که فکر می‌کنید کدهای پایه‌ای هستند و برای اجرای برنامه لازم هستند را در داخل بلوک finally قرار دهید. برنامه زیر نحوه استفاده از این بلوک را نشان می‌دهد:

```

try
{
  number;
}
catch(err)
{
  console.log(err.message);
}
finally
{
  console.log("finally blocked was reached.");
}

```

number is not defined
finally blocked was reached.

بلوک finally بعد از بلوک catch نوشته می‌شود.

این اثر رایگان بوده و هرگونه استفاده تجاری از آن پیگرد قانونی دارد.

استفاده از مطالب آن، بدون ذکر منبع، غیراخلاقی و غیرقانونی است.

راههای ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com



از سایر کتاب های یونس ابراهیمی در لینک های زیر دیدن فرمایید

<https://bit.ly/2Is8a0t>

www.w3-farsi.com/product