



JavaScript

به زبان ساده

آریا تهران

چيست.....	JavaScript 5 5
.....	ساخت يك برنامه ساده JavaScript.....
9	توضيحات
10.....	کاراکترهای کنترلی
12.....	متغير
14.....	انواع داده
15.....	استفاده از متغيرها
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.....	چندگانه if دستور
48.....	تو در تو if دستور
50.....	استفاده از عملگرهای منطقی
52.....	دستور Switch
55.....	تکرار
55.....	While حلقه
57.....	do while حلقه
58.....	for حلقه
59.....	حلقه های تو در تو (Nested Loops)

61.....	خارج شدن از حلقه با استفاده از break و continue.....
62.....	آرایه
64.....	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 پرطرفدار 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 و Netscape Navigator همزمانی دارد.

اولین نسخه جاوااسکریپت در نسخه 3.0 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 اینطور 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 برنامه نویسی تحت وب بود میدانند.

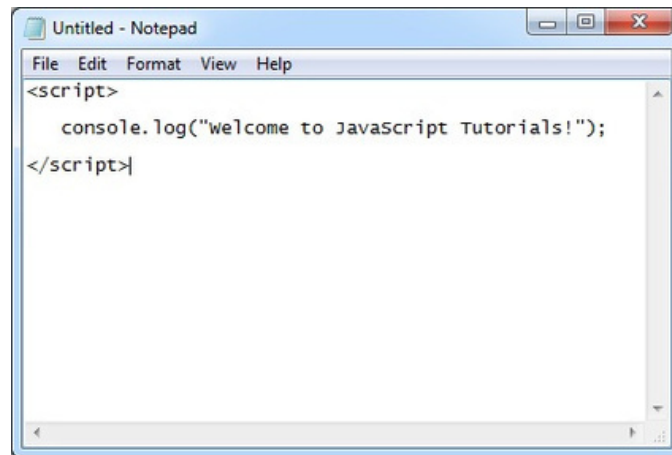
JavaScript به صورت «جاوااسکریپت» خوانده میشود، ولی در فارسی به صورت «جاوااسکریپت» ترجمه میشود و اگر به صورت «جاوا اسکریپت» ترجمه شود اشتباه است چون دو کلمه جدا از هم نیست و اگر به صورت دو کلمه جدا نوشته شود خطاهای نگارشی ایجاد میشود، بهطور مثال ممکن است کلمه جاوا در انتهای خط و کلمه اسکریپت در ابتدای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

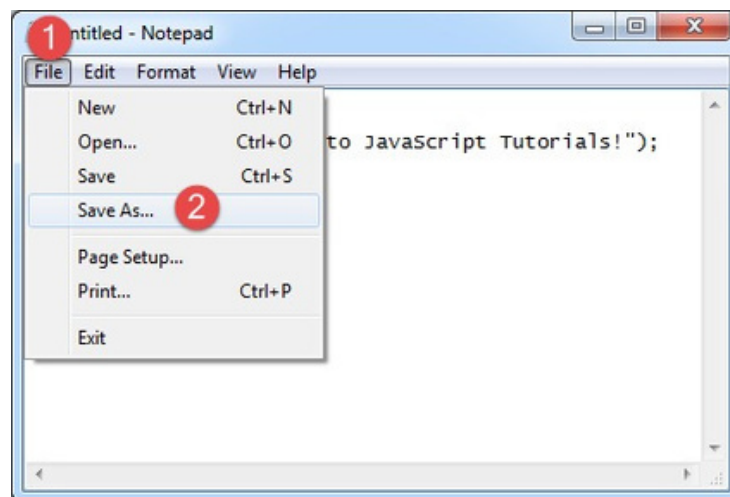
ساخت یک برنامه ساده JavaScript

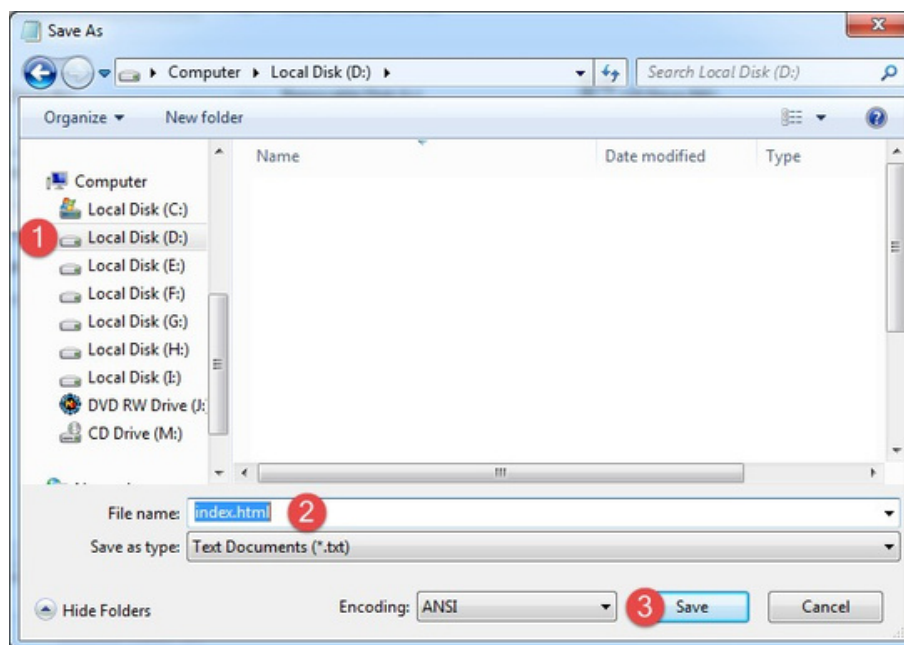
اجازه بدهید یک برنامه بسیار ساده به زبان جاوااسکریپت بنویسیم. این برنامه یک پیغام را نمایش میدهد. در این درس میخواهم ساختار و دستور زبان یک برنامه ساده جاوااسکریپت را توضیح دهم. برای اجرای کدهای جاوااسکریپت، هیچ ابزار خاصی نیاز نیست. در واقع بر خلاف زبانهای دیگر که نیاز به یک کامپایلر برای اجرای کدها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

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

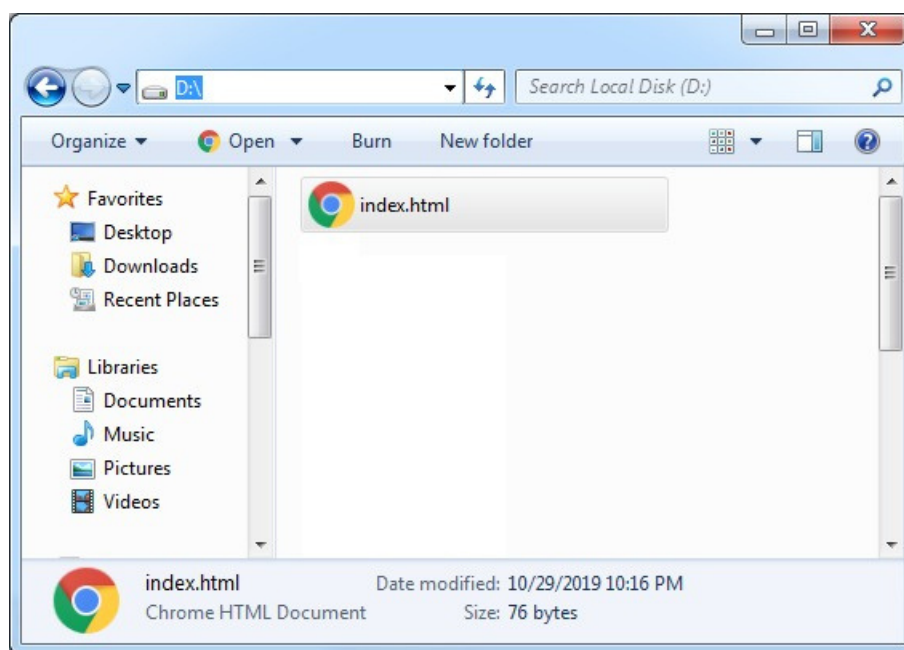


حال برنامه بالا را با نام `index.html` در درایو D به صورت زیر ذخیره کنید

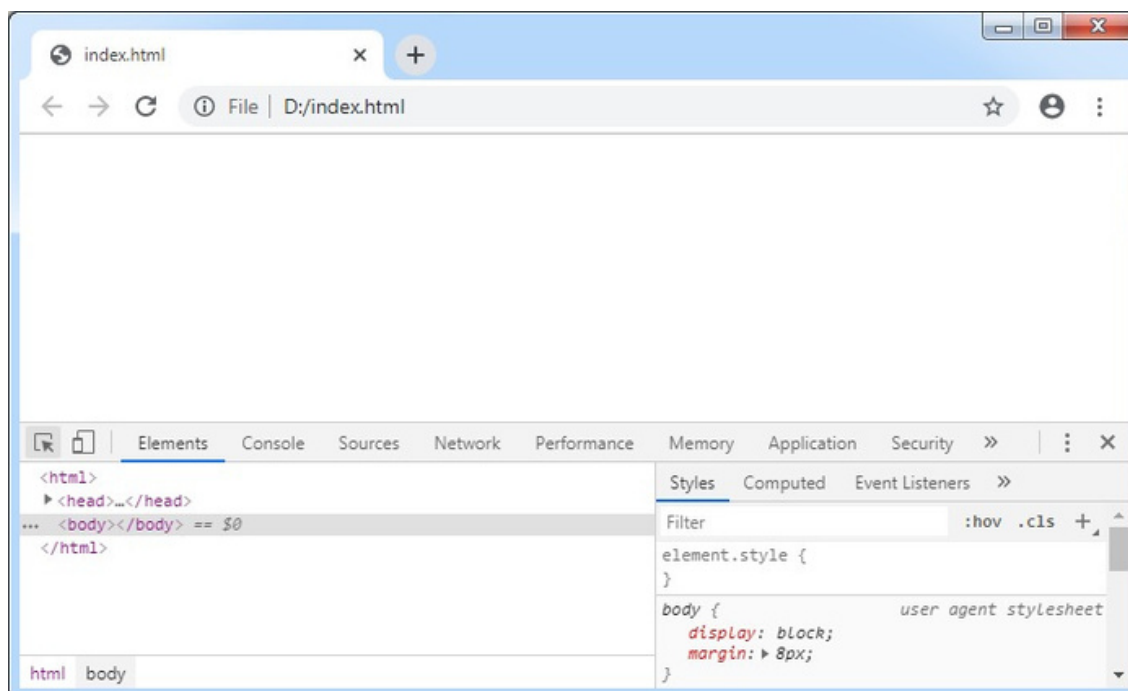




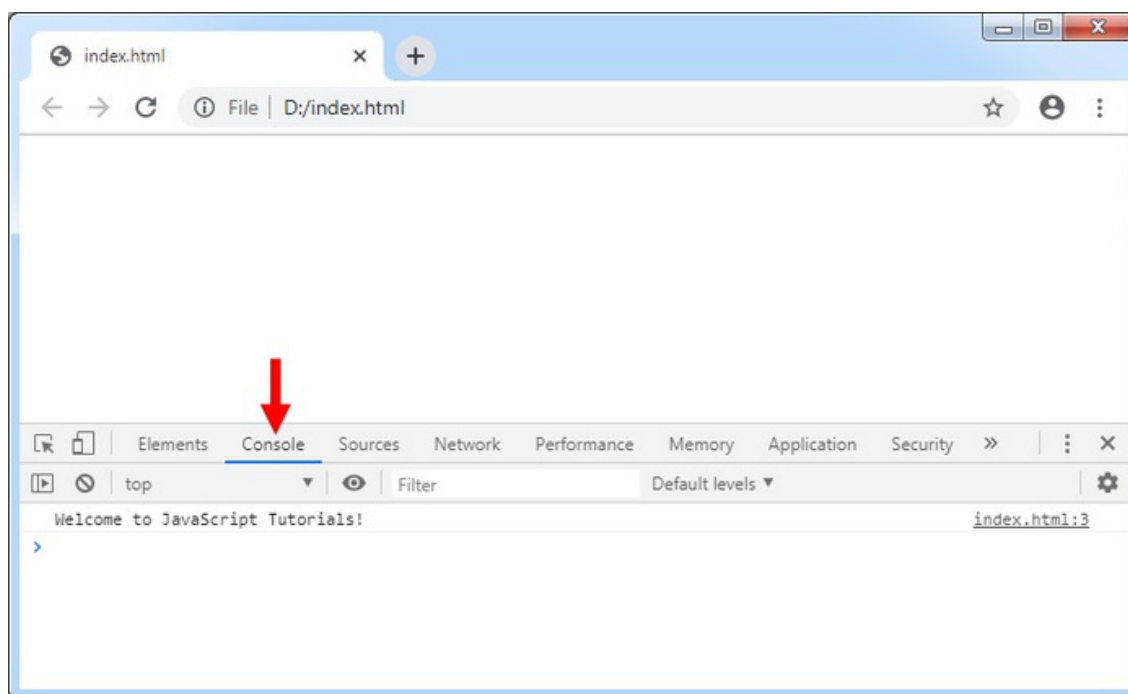
سپس به درایو رفته و بر روی فایل `index.html` دو بار کلیک کنید تا به وسیله مرورگرتان اجرا شود:



اگر از یکی از مرورگرهای `Firefox`، `Opera` و یا `Chrome` استفاده می کنید، کافیست 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 برنامه‌های است که شما میتوانید در javascript بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است.

توصیه می کنیم که برای اجرای کدهای Javascript از آخرین نسخه مرورگر Chrome استفاده کنید، چون از تمام قابلیت ها و امکانات نسخه نهایی جاوااسکریپت پشتیبانی می کند.

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 می دهیم، ولی در همین حد به توضیح تابع بسنده می کنیم که توابع مجموعه ای از کدها هستند که دارای یک نام بوده و در جلوی نام آنها علامت () قرار میگیرد. یکی از این توابع، تابع () log است. از تابع () log برای چاپ یک رشته استفاده میشود. یک رشته گروهی از کاراکترها است، که به وسیله دابل کوتیشن "" محصور شده است. مانند "Welcome to javascript Tutorials!".

یک کاراکتر میتواند یک حرف، عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از تابع () 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 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 نوشتههای 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 نوشتههای 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 کرد :

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

Hello
World
```

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

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

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.
```

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

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

```
Left Right
```

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

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

```
©
```

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

```
http://www.ascii.cl/htmlcodes.htm
```

اگر مفسر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا میدهد. بیشترین خطا زمانی اتفاق می افتد 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 میندازید 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.
- ☐ دو کاراکتر مختلف A و a اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در جاوااسکریپت دو حرف مانند به حساب میآیند.

دو متغیر با نامهای myNumber و MyNumber و دیگری با حرف m دو متغیر مختلف محسوب میشوند چون یکی از آنها با حرف کوچک 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.

شیء (Object)	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.
boolean	شامل دو مقدار true یا false می باشد.
Null	یک مقدار خاص است که به معنای هیچ چیز، خالی و یا مقدار ناشناخته است.
undefined	اگر متغیری تعریف شود و مقداردهی اولیه نشود آنگاه مقدار آن برابر با undefined خواهد بود

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 کافیت

و سپس نام متغیر را نوشته و به وسیله علامت مساوی یک مقدار به آن اختصاص دهیم var که ابتدا کلمه کلیدی:

```
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؟ نوع متغیرها را بسته به مقداری که به آنها اختصاص JavaScript

داده می شود در نظر می گیرد. مثلاً 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 آیتیم 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. آیتیم 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 }
```

مثلاً در مثال بالا یک `Object` تعریف کرده ایم که سه آیتیم 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 مقداردهی 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 Q برابر است. برای درک بهتر به شکل زیر توجه کنید 4:

```
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 (مثلاً این رشته کافیهست که به صورت زیر عمل کنیم) W:

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

```
W
```

همانطور که در کد بالا مشاهده می کنید کافیسست 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 '2'

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

```
Scalia
```

ثابت ها

ثابتها، انواعی هستند که مقدار آنها در طول برنامه تغییری نمیکنند. 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 مقداری 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()` ☐

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

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() □

Boolean() بر میگرداند که پارامتر دریافتیاش، رشتهای شامل حداقل یک کارکتر، یک عدد غیر از صفر و یا یک true متد زمانی مقدار undefined و null مقادیر یا یکی عدد صفر یا یکی 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() گومان 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 حمایت از w3-f ars مراجعه فرمایید

انتقادات و پیشنهادات خود را به ایمیل youne.s.e.brahimi@gmail.com ارسال فرمایید

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید :

- ☐ عملگر: نمادهایی هستند که اعمال خاص انجام میدهند.
 - ☐ عملوند: مقداری که عملگرها بر روی آنها عملی انجام میدهند.
- مثلاً $x+y$. عملوند و علامت + عملگر به حساب میآیند و x و y یک عبارت است که در آن :

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 می کند. جدول زیر عملگرهای ریاضی JavaScript را نشان می دهد :

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

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 باقیمانده تقسیم $var2$ بر $var3$	$var1 = var2 / var3;$	Binary	/
$var1$ برابر است با مقدار $var2$ و $var3$	$var1 = var2 \% var3;$	Binary	%
$var1$ برابر است با مقدار $var2$	$var1 = +var2;$	Unary	+
$var1$ برابر است با مقدار $var2$ - ضریب 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 $var2$ به علاوه 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
```

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

```
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
```

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

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

```

7 num2 = 3; //Demonstrate use of mathematical operators console.log("The sum of", num1,
8 "and", num2, "is", (num1 + num2)); console.log("The difference of", num1, "and", num2,
9 "is", (num1, num2)); console.log("The product of", num1, "and", num2, "is", (num1 *
10 num2));
11 console.log("The quotient of", num1, "and", num2, "is", (num1 / num2).toFixed(2));
12 console.log("The remainder of", num1, "divided by", num2, "is", (num1 % num2));
13 console.log("The result of", num1, "powered by", num2, "is", (num1 ** num2));
14 //Demonstrate concatenation on strings using the + operator msg1 = "Hello "; msg2 =
15 "World!";
16
17
18
19
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 <code>var1</code> برابر است با مقدار <code>var2</code>
+=	<code>var1 += var2;</code>	مقدار <code>var1</code> برابر است با حاصل جمع <code>var1</code> و <code>var2</code>
-=	<code>var1 -= var2;</code>	مقدار <code>var1</code> برابر است با حاصل تفریق <code>var1</code> و <code>var2</code>
*=	<code>var1 *= var2;</code>	مقدار <code>var1</code> برابر است با حاصل ضرب <code>var1</code> در <code>var2</code>

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 مقایسه ای

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	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 مقدار var3 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 $x = y$ اختصاص می‌دهد. عملگر x را در به y مقدار == 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;
!	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 ANDمانی کارآمد است که ما با محدود خاصی از اعداد سر و کار داریم. مثلاً عبارت $x \leq 100$ می x بدین معنی است که $x \geq 10$ => 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

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

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

عملگر منطقی (!) NOT

و OR برخلاف دو اپراتور 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. مثلاً 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);
```

عملگرهای بیتی به شما اجازه میدهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه میشود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را از لینک زیر یاد بگیرید :

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده میکند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد 1 و برای نشان دادن حالت خاموش از عدد 0 استفاده میشود. بنابراین اعداد باینری فقط میتوانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای 2 و اعداد اعشاری را اعداد در مبنای 10 می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده بیت است. به عنوان مثال برای یک داده از نوع `int8` به 32 بیت یا 4 بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از 32 رقم 0 و 1 برای ذخیره استفاده میکنند. برای مثال عدد 100 وقتی به عنوان یک متغیر از نوع `int` ذخیره میشود در کامپیوتر به صورت زیر خوانده میشود

عدد 100 در مبنای ده معادل عدد 1100100 در مبنای 2 است. در اینجا 7 رقم سمت راست نشان دهنده عدد 100 در مبنای 2 است و مابقی صفرهای سمت چپ برای پر کردن بیت هایی است که عدد از نوع int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده میشوند. عملگرهای بیتی Javascript در جدول زیر نشان داده شدهاند :

عملگر	نام	مثال
&	بیت-AND	x = y & z;
	بیت-OR	x = y z;
	بیت-XOR	x = y ^ z;
^	بیت-NOT	x = ~y;
~	بیت-AND Assignment	x &= y;
&=	بیت-OR Assignment	x = y;
=	بیت-XOR Assignment	x ^= y;
^=		

2

عملگر	نام	مثال
>>	تغییر مکان به سمت چپ	$x = y \ll 2;$
<<	تغییر مکان به سمت راست	$x = y \gg 2;$

این عملگر بیت‌های عملوند سمت چپ را به تعداد

مثال :

40

6

هر بیت به اندازه واحد به سمت راست منتقل میشود، بنابراین ⁴بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه میشود.

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 $^{9/1}$.) 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 : با 1 جمع میشود و عدد 7 حاصل میشود. در جدول زیر تقدم برخی از عملگرهای جاوااسکریپت آمده است 6 :

اولویت	شرکت پذیری	عملگر
بیشترین اولویت	از چپ به راست	()
1	از چپ به راست	[]
1	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	~
2	از راست به چپ	delete
2		new
2		typeof
2		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 راست به چپ	10
&&	از راست به چپ	11
		12
?:		13
=		14
*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, =		14
,	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 د). 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 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 داخلترین 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;
```

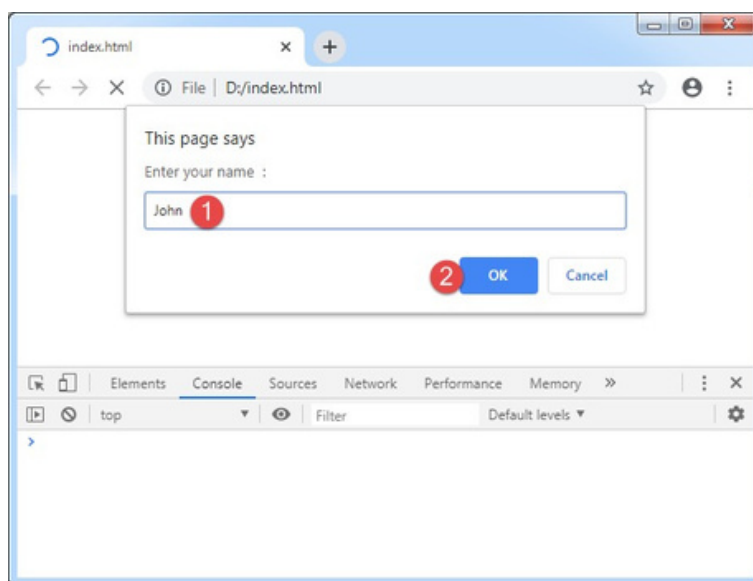
هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آنها را در محاسبات تأثیر دهید. یعنی ابتدا 3 را ضربدر 2 میکنید و سپس عدد 8 را بر 4 تقسیم میکنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار میدهید.

گرفتن ورودی از کاربر

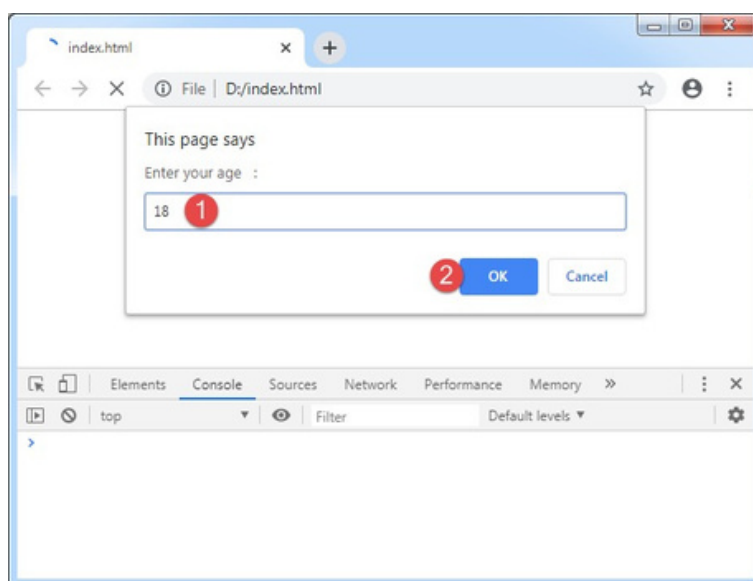
JavaScript متد prompt() را برای گرفتن ورودی از کاربر، در اختیار شما قرار میدهد. این متد باعث ظاهر شدن یک پنجره Popup می شود. این پنجره یک جعبه متن در اختیار شما می گذارد و تمام کاراکترهایی را که شما در محیط برنامه نویسی تایپ میکنید تا زمانی که دکمه Enter را میزنید، میخواند. به برنامه زیر توجه کنید :

```
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);
```

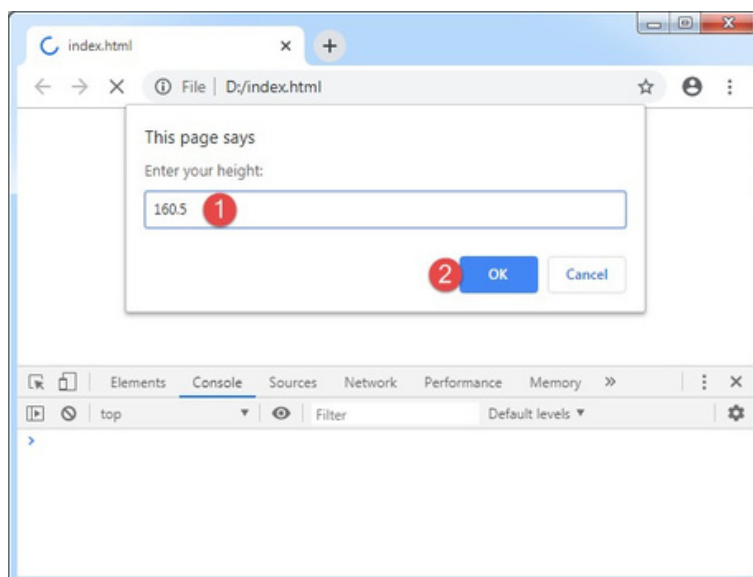
ابتدا 3 متغیر را برای ذخیره داده در برنامه تعریف میکنیم (خطوط 1 و 2 با اجرای برنامه پنجره 3) Popup ظاهر می شود و از کاربر میخواهد که نام خود را وارد کند (خط 1)



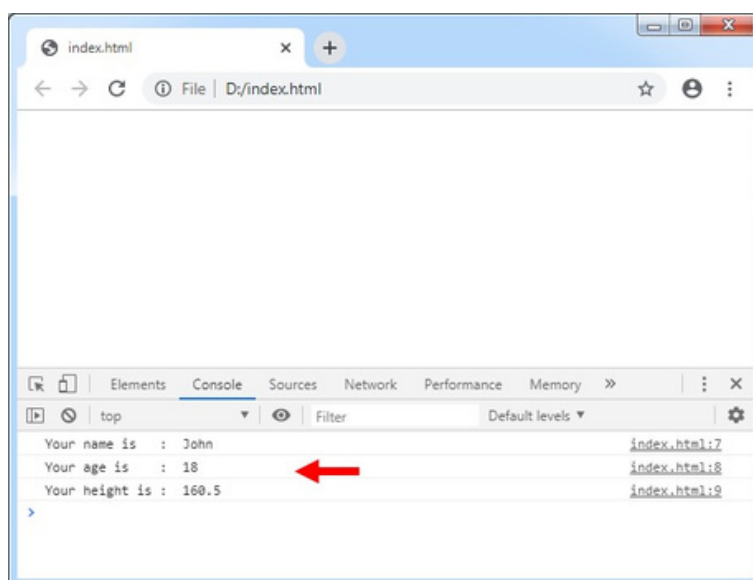
با زدن دکمه OK (دوباره پنجره ظاهر شده و از ما سن را سؤال میکند) خط 3 ،



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



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



البته JavaScript دارای متدهای 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 پیام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید

اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیام 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
- ☐ چنگانه if دستور
- ☐ تو در تو if دستور
- ☐ عملگرهای منطقی
- ☐ دستور switch

دستور if

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

از 10 بزرگتر باشد در صفحه نمایش میدهد number اگر مقدار

```
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 کوچکتر از 10 است.

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

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

بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده if شما میتوانید چندین دستور را در داخل دستور

به صورت زیر است if هستند. نحوه تعریف چند دستور در داخل بدنه if کنید. همه دستورات داخل بین آکولاد جز بدنه دستور

:

```
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 از 10 بزرگتر باشد دو پیغام چاپ میشود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار از 10^x 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.");
```

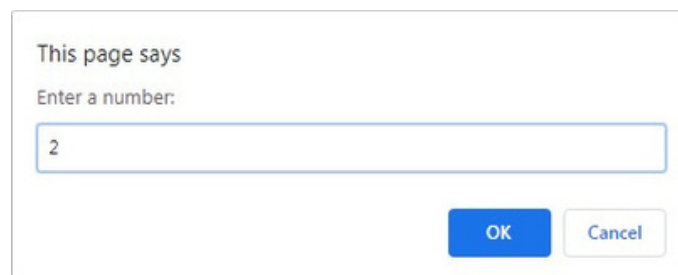
از 10 x نیست. اینجاست که چون ما فرض را بر این گذاشتهایم که مقدار if میبیند که دستور دوم یعنی خط آخر در مثال بالا، جز دستور کوچکتر است پس خط (Really?) This is still part of the if statement. چاپ میشود. در نتیجه اهمیت وجود داشتید برای آن یک آکولاد بگذارید. فراموش نکنید if آکولاد مشخص میشود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه 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 if (firstNumber == secondNumber) {
7 }
8 if (firstNumber != secondNumber)
9 {
10   console.log(firstNumber + "==" + secondNumber);
11 } if (firstNumber < secondNumber) { } if (firstNumber >
12 secondNumber) { } if (firstNumber <= secondNumber) { } if
13 (firstNumber >= secondNumber) { }
14   console.log(firstNumber + "!=" + secondNumber);
15
16
17   console.log(firstNumber + "< " + secondNumber);
18
19
20
21   console.log(firstNumber + "> " + secondNumber);
22
23
24
25   console.log(firstNumber + "<=" + secondNumber);
26
27
28
29   console.log(firstNumber + ">=" + secondNumber);
30
```

استفاده کرده‌ایم ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر if ما از عملگرهای مقایسه‌ای در دستور

از نوع رشته است، پس باید آنها را با استفاده prompt() (میگیریم) خطوط 4 و 5. توجه کنید که چون ورودی‌های گرفته شده توسط متد به نوع عددی تبدیل کنیم تا کار مقایسه راحت تر انجام شود. با اجرای برنامه پنجره‌ای به صورت زیر برای شما نمایش parseInt() از متد داده می شود:



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

در این پنجره نیز بعد از وارد کردن عدد 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 استفاده کرد خواه مقدار متغیر درست if ذخیره میشود. میتوان از متغیر برای تشخیص کد اجرایی بدنه دستور باشد یا نادرست.

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

دستور else...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 if دستور 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 :

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

  code to execute if condition is false;
```

else و بدنه if به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه if نمیتوان به تنهایی استفاده کرد بلکه حتماً باید با else از کلمه کلیدی نادرست if فقط در صورتی اجرا میشود که شرط داخل دستور else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک 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  else console.log("The number is less than 10.");
7  { } //Modify value of number
8  number = 15; //Repeat the test to yield a
9  different result
10 if (number < 10)
11 { } console.log("The number is either greater than or equal to 10.");
12 else
13 { }
14
15
16
17
18
19 console.log("The number is less than 10.");
20
21
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 10 کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک if اجرا میشود (خط 6) و اگر مقدار number تغییرییر 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 از این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```

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

```

```

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 را در type2 قرار بده. خط 8 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 را در type2 قرار بده. خط 9 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 را در type2 قرار بده. خط 10 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 را در type2 قرار بده.

```

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)
{ } else {
  code to execute;

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

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

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

code to execute;

code to execute;
```

را یاد گرفتید باید بدانید که مانند ، نیز به دستور `else` استفاده می‌شود. دستور `else if` حال که نحوه استفاده از دستور

اشتباه باشد حال اگر اشتباه باشد دستور بعدی اجرا می‌شود. و اگر `if` وقتی اجرا می‌شود که اولین دستور `else if`

اجرا می‌شود. برنامه زیر نحوه استفاده از دستور را نشان می‌دهد `else` آن نیز اجرا می‌شود در نهایت دستور:

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

console.log("You might be sad today.");

console.log("You might be inlove right now.");
```

```

    {
    }
else console.log("Lemon might be your favorite fruit.");
{
}

    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)
    {
    }
else code to execute;
{

        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  gender = parseInt(prompt("Enter your age:"));
4  if (age > 12)
5  {
6
7
8
9
10     if (age < 20)
11     {
12         if (gender == "male")
13         { }
14         else console.log("You are a teenage boy.");
15         { }
16
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 ستنان (خط 4) و سپس پنجره دوم باز شده خط 5) و درباره جنستان از شما سؤال میکند. سپس به اولین دستور (`if` میرسد) خط 8. در این قسمت اگر سن شما بیشتر از 12 سال میشود در غیر اینصورت وارد بلوک `if` باشد برنامه وارد بدنه دستور `else` (خط 26) میشود `if` مربوط به همین دستور (حال فرض کنیم که سن شما بیشتر از 12 سال است و شما وارد بدنه اولین `if` دیگر را مشاهده `if` دو دستور `if` شدهاید. در بدنه اولین میکنید. اگر سن کمتر 20 باشد شما وارد بدنه `if` دوم میشوید و اگر نباشد به قسمت `else` متناظر با آن میروید) خط 21. دوباره فرض دیگر مواجه میشوید) خط 12. در اینجا جنسیت `if` دوم شده و با یک `if` میکنیم که سن شما کمتر از 20 باشد، در اینصورت وارد بدنه شما مورد بررسی قرار میگیرد که اگر برابر `male` اجرا میشود در غیر اینصورت قسمت `if` باشد، کدهای داخل بدنه سومین `else` مربوط تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین میآورد `if` اجرا میشود) خط 16. پیشنهاد میشود که از `if` به این

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 :

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

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

```
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 (B) عبارت)

مورد بررسی قرار میگیرد (به علت تقدم عملگرها). سپس نتیجه AND وسیله عملگر (B) مقایسه میشود. حال بیایید

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 if (age > 12 && age < 20)
7 {
8
```

```

9   if (gender == "male")
10  { } else {
11    console.log("You are a teenage boy.");
12
13
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). وقتی به دستور `if` میرسید (خط) برنامه سن شما را چک میکند. اگر سن شما بزرگتر از 12 و کوچکتر از 20 باشد (سنستان بین 12 و 20 باشد) یعنی مقدار هر دو `true` باشد سپس کدهای داخل اجرا میشوند. اگر نتیجه یکی از شروط `if` بلوک `false` باشد کدهای داخل بلوک `else` اجرا میشود. عملگر AND عملوند سمت چپ را مرود 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)
{ }
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 باشد عملوند سمت چپ به وسیله عملگر بیتی ارزیابی میشود. اگر شرطها را در `else` (مثلاً `AND`) حتی اگر مقدار عملوند سمت چپ `false` و به جای عملگرهای بیتی و بهتر خواهد بود `AND` (`&&`) برنامه ترکیب کنید استفاده از عملگرهای منطقی است که نتیجه یک عبارت را خنثی یا منفی میکند. به مثال زیر توجه کنید `NOT` (`!`) یکی دیگر از عملگرهای منطقی عملگر:

```

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 ساختار `switch` وجود دارد که به شما اجازه میدهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته

`switch` باشید. دستور `switch` در تو است با این تفاوت که `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 `switch` آمده است :

```
switch (testVar)
{
  case compareVal1:
    code to execute if testVar == compareVal1;
    break;
  case compareVal2:
    code to execute if testVar == compareVal2;
    break;
  .
  .
  .
  case compareValN:
    code to execute if testVer == compareValN;
    break;
  default:
    code to execute if none of the values above match the testVar;
    break;
}
```

`switch` یک مقدار در متغیر `testVar` مثال بالا است قرار میدهید. این مقدار با هر یک از عبارتهای داخل بلوک

مقایسه میشود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستور `switch` برابر بود کد مربوط به آن اجرا خواهد شد.

به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل `case` از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. آخر هر دستور

با کلمه کلیدی تشخیص داده میشود که باعث میشود برنامه از دستور `switch` خارج شده و دستورات بعد از آن اجرا `case`

های زیرین تا پایان `case` شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه نمیشود ولی مفسر به اجرای دستور هر یک از

دارد. این دستور در صورتی اجرا میشود که مقدار متغیر با هیچ یک از مقادیر `default` یک بخش `switch` ادامه خواهد داد. دستور

اختیاری است و اگر از بدنه حذف شود هیچ اتفاقی نمیافتد. `switch` دستور هم `default` برابر نباشد. دستور `case` دستورات

مهم نیست اما بر طبق تعریف آن را در پایان دستورات مینویسند. به مثالی در مورد `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 choice = parseInt(prompt("Enter your choice: "));
11 console.log("Enter your choice: " + choice);
12 switch (choice)
13 {
14
15
16
17
18     case 1:
19         console.log("Your favorite pet is Dog.");
20         break;
21
22     case 2:
23         console.log("Your favorite pet is Cat.");
24         break;
25
26     case 3:
27         console.log("Your favorite pet is Rabbit.");
28         break;
29
30     case 4:
31         console.log("Your favorite pet is Turtle.");
32         break;
33
34     case 5:
35         console.log("Your favorite pet is Fish.");
36         break;
37
38     case 6:
39         console.log("Your favorite pet is not in the choices.");
40         break;
41
42     default:
43         console.log("You don't have a favorite pet.");
44         break;
45 }

```

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 پیام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر case ها برابر نبود دستور default اجرا می‌شود. مثلاً اگر در پنجره باز شده عدد را بنویسیم، خروجی برنامه به صورت زیر خواهد بود 2

```

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، 3 یا 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 نتایجی 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 میدهم.

حلقه While

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

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

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

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 while (counter <= 10)
3 {
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 بیندازیم. 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 کد 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 بی نهایت 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 میگیرد. ساختار حلقه do while به صورت زیر است :

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

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

```
var number = 1;

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

Hello World!
```

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

```
var    number    =    1;

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

اما در کد بالا چون اول number ابتدا مورد مقایسه قرار میگیرد، اگر شرط درست نباشد دیگر کدی اجرا نمیشود. یکی از موارد برتری استفاده از حلقه while نسبت به حلقه while است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. در دو کد زیر، یک عملیات یکسان توسط دو روش 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 for 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

```

میشمارد. ابتدا یک متغیر به عنوان شمارنده تعریف میکنیم و آن را با مقدار 1 مقدار for برنامه بالالا اعداد 1 تا 10 را با استفاده از حلقه 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 و سپس مقدار جاری یعنی را چاپ میکند. آنگاه یک واحد به Number فور اجرا نمیشود. کد اجرا میشود و ابتدا رشته (++i) حلقه با عدد 10 مقایسه میشود و این حلقه تا زمانی که مقدار شرط شود ادامه i برابر 2 میشود و بار دیگر i اضافه شده و مقدار i مقدار مییابد. حال اگر بخواهید معکوس برنامه بالالا 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 بالالا اعداد را از 10 به 1 چاپ میکند (از بزرگ به کوچک). مقدار اولیه شمارنده را 10 میدهیم و با استفاده از عملگر کاهش (–) برنامه‌های 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);
    }

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

استفاده نمود. در مثال زیر نحوه استفاده از این حلقه‌ها ذکر شده است. فرض کنید که میخواهید یک while در داخل حلقه for حلقه

: 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("\n");
    console.log(" * ");
}

```

```

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

```

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 برابر عدد برابر 2 میشود، دوباره علامت * پنج بار چاپ میشود و در کل منظور i میشود، علامت * توسط حلقه دوم 5 بار چاپ میشود، وقتی 1 از دو حلقه 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 هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد میشود و علامتهای * در خطوط 7 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 break کلمه کلیدی حلقه را متوقف کرده و با استفاده از continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت.

برنامه زیر استفاده از break و continue را نشان می‌دهد:

```

1 console.log("Demonstrating the use of break.\n");
2 for (var x = 1; x < 10; x++)
3 {
4     if (x == 5)
5         break;
6
7     console.log("Number " + x);
8 }
9
10 console.log("\nDemonstrating the use of continue.\n");
11 for (var x = 1; x < 10; x++)
12 {
13     if (x == 5)
14         continue;
15
16     console.log("Number " + x);
17 }
18
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 به عدد x 5 استفاده میشد نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط 5) آمده است، وقتی که مقدار برسد، سپس دستور break (خط 6) و حلقه بلافاصله متوقف میشود، حتی اگر شرط $x < 10$ برقرار باشد. از طرف دیگر در خط 15 برابر 5 شود حلقه از 5 رد شده و مقدار 5 را چاپ x فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد. (وقتی مقدار for حلقه 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. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئناً 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 آدرس از فضای حافظه کامپیوتر شما ذخیره می شود. مثلاً بالا را به روش زیر هم می توان پیاده سازی کرد :

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

```
numbers[0] = 1;
```

```
numbers[1] = 2;
```

```
numbers[2] = 3;
```

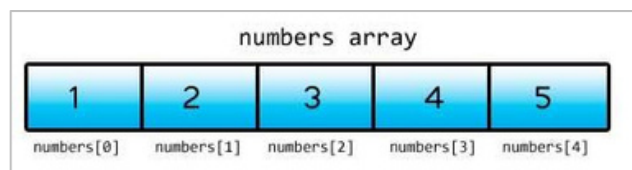
```
numbers[3] = 4;
```

```
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 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 معمولاً در گذاشتن اندیس دچار اشتباه می شوند و مثلاً 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 در داخل پرانتز کد بالا بدین معنی است که ما می‌خواهیم 5 عنصر را در داخل آرایه ذخیره کنیم 5.

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 for (var i = 0; i < numbers.length; i++)  
5 {  
6     numbers[i] = parseInt(prompt("Enter a number: "));  
7     console.log("Number " + i + ": " + numbers[i]);  
8 } for (var i = 0; i < numbers.length;  
9 i++)  
10 { } average = total / numbers.length;  
11  
12 total += numbers[i];  
13  
14  
15  
16  
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 میتواند 5 عدد صحیح را در خود ذخیره کند. خطوط 2 و 3 متغیرهایی تعریف شده اند که از آنها برای محاسبه میانگین استفاده میشود. 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) مقدار 5 را برای شرط قرار for ارائه برای تشخیص تعداد اجزای آرایه استفاده میشود. اگر چه میتوانستیم به سادگی در حلقه (دهیم ولی استفاده از خاصیت طول آرایه کار راحتتری است و میتوانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ تبدیل و در آرایه ذخیره میشود. اندیس استفاده شده در int میشود. در خط 7 ورودی دریافت شده از کاربر به نوع number (خط 7) صفر است بنابراین وقتی در خط 7 اولین داده از کاربر گرفته میشود i جاری در حلقه است. برای مثال در ابتدای حلقه مقدار i مقدار یک واحد اضافه میشود و در نتیجه در خط 7 و بعد از ورود دومین داده توسط کاربر اندیس i اندیس آن برابر 0 میشود. در تکرار بعدی آن برابر 1 میشود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه میابد. 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 میکنیم. هر یک از اجزای عددی آرایه به متغیر total اضافه میشوند. بعد از پایان حلقه میتوانیم میانگین اعداد را حساب کنیم (خط 16). مقدار total را بر تعداد اجزای آرایه (تعداد عددها) تقسیم میکنیم. برای دسترسی به تعداد اجزای آرایه میتوان از خاصیت length آرایه استفاده کرد. خط 18 مقدار میانگین را در صفحه نمایش چاپ میکند. 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
```

آرایهها 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

یکی دیگر از ساختارهای تکرار در جاوااسکریپت میباشد که مخصوص آرایهها، رشته ها و مجموعهها طراحی شده for...of حلقه با هر بار گردش در بین اجزاء، مقادیر هر یک از آنها را در داخل یک متغیر موقتی قرار میدهد و شما میتوانید بواسطه for...of حلقه : آمده است for...of این متغیر به مقادیر دسترسی پیدا کنید. در زیر نحوه استفاده از حلقه

```
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 for (var n of numbers) { }
3
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 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]
];
```

تعریف شده است که دارای سه عنصر است. البته هر کدام از این عناصر خود یک آرایه می باشند numbers در کد بالا یک آرایه به نام در واقع یک numbers آرایه است از 3 آرایه. هر کدام از این آرایه ها هم 5 عنصر دارند. پس می توان گفت که آرایه numbers یعنی آرایه جدول با 3 سطر و 5 ستون می باشد. میتوان 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 میکنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی میتوان از اندیسهای سطر و ستون و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایههای چند بعدی

گردش در میان عناصر آرایههای 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], [ 6,
3     7, 8, 9, 10], [11, 12,
4     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

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

که (col) دیگری تعریف شده است (خط 9). در این حلقه یک شمارنده برای شمارش تعداد ستونها for حلقه for در داخل اولین حلقه

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 صفر باشد، حلقه دوم از 0 تا 4 اجرا میشود. سپس مقدار هر عنصر از آرایه را با استفاده (row) عنوان مثال وقتی که مقدار ردیف از حلقه نشان میدهم، 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 این فرایند را دوباره تکرار میکند. سپس row به برنامه اطلاع میدهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار کمتر از طول اولین row اجرا شده و مقادیر دومین ردیف نمایش داده میشود. این فرایند تا زمانی اجرا میشود که مقدار for دومین حلقه بعد باشد. حال بیایید آنچه را از قبل یاد گرفتیم 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  for (var student = 0; student < studentGrades.length; student++)
7  {
8
9
10     total = 0;
11     console.log("Enter grades for Student " + student + 1);
12     for (var grade = 0; grade < studentGrades[student].length; grade++)
13     {
14
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 1-). همچنین یک متغیر به نام `total` تعریف میکنیم که جمع نمرات وارد شده برای دانش آموز در آن قرار میگیرد (خط 2-). حال وارد حلقه `for` یک `for` تو در تو می‌نویسیم (خط 3-). در اولین حلقه متغیر به نام `student` تعریف کرده‌ایم که تعداد دانش آموزان در آن قرار میگیرد. از خاصیت `length` هم برای تشخیص تعداد دانش میشویم. در خط `for` آموزان استفاده شده است. وارد بدنه حلقه 10 مقدار متغیر `total` را برقرار 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 `Student 0`، با `Student 1` در خط 14 میرسیم `for` شروع شود، لطیف‌تر به نظر برسد (خط 4-). سپس به دومین حلقه 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 `toFixed(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 { } PrintMessage ();
3     console.log("Hello World!");
4
5
6

```

Hello World!

در خطوط 1-4 یک تابع تعریف کرده ایم. همانطور که مشاهده می کنید در خط و برای تعریف تابع از `function` نام

تابع و بعد از آن پرانتز باز و بسته استفاده (`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 مثلاً 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 علاامت سوال در آخر اسم تابع خودداری

کنید. دو پراتنزی که بعد از نام می آید نشان دهنده آن است که نام متعلق به یک تابع است. بعد از پراتنرها 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 کافیت نام آن را نوشته و بعد از نام پراتنرها را قرار دهیم.

برای اجرای تابع PrintMessage() برنامه از خط به محل تعریف تابع PrintMessage() می رود. مثلاً 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. کافیت 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 جمع دو متغیر در متغیر `sum5` 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 `if` یک تابع از دستور `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     if (number > 10)
5     {
6     }
7     elsereturn number;
8     {
9     }
10
11     return 0;
12 }
13
14 var result = GetNumber();
15 console.log(result);
16
```

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 4 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` و یا دستور `if` دستور `return` را از آن حذف کنیم در هنگام اجرای برنامه نتیجه چاپ نمی شود. چون اگر نادرست باشد برنامه به قسمت `if` شرط دستور `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. `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 آرگومان ها

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 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 میتواند 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 { } var result = CalculateSum(10,5);
3 console.log(result);
4
5
6
7
```

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. حرف اول دومین کلمه بزرگ نوشته میشود(نوشته میشود) camelCasing کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا 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 function AnotherFunction(number)
3 { return 5; }
4 AnotherFunction(MyFunction());
5
6
7
8 console.log(number);
9
10
11

```

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 می باشد:

```

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

```

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 { PrintMessage(); }
3 PrintMessage("Learn Javascript Today!");
4
5
6
7
8

```

```

Welcome to Javascript Tutorials!
Learn Javascript Today!

```

خطوط 4-1) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری میتوان به آسانی و با استفاده از علامت (PrintMessage) تابع 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 = استفاده میکنیم. در دومین فراخوانی (!Welcome to Javascript Tutorials) تابع ارسال نمیکیم بنابراین تابع از مقدار پیشفرض 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 می شوند از سمت چپ به راست در داخل پارامترها قرار می گیرند، اگر پارامترهای اختیاری در ابتدای پارامترها قرار داده شوند، مقادیر آنها توسط آرگومان های ارسالی جایگزین می شوند. 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. **Named Parameters** مطرح شده است. در زیر نحوه استفاده از نام آرگومانها وقتی که تابع فراخوانی میشود نشان داده شده است ES6 در:

```
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 SetSalaries( {jack: 120, andy: 30, mark: 75} );
8 console.log("\n");
9 SetSalaries( {andy: 60, mark: 150, jack: 50} );
10 console.log("\n");
11 SetSalaries( {mark: 35, jack: 80, andy: 150} );
12
```

```
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 مربوطه 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 می کنند. ولی می توان 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 می کنید، کافیهست، آرگومان هایی 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; for (var
4     number of numbers)
5     { } return total;
6   total += number;
7
8
9
10
11 }
12 console.log("1 + 2 + 3 = ", CalculateSum(1, 2, 3));
13 console.log("1 + 2 + 3 + 4 = ", CalculateSum(1, 2, 3, 4));
14 console.log("1 + 2 + 3 + 4 + 5 = ", CalculateSum(1, 2, 3, 4, 5));
15
16
17
```

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

```

```

10
5

```

مشاهده میکنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم (خطوط ۳ و ۹) 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; function
4 GlobalVariable()
5 { } GlobalVariable ();
6 console.log(Sum + firstNumber + secondNumber);
7
8
9
10
11
15
```

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

Arrow Function

Arrow Function روشی جدید برای تعریف توابع در جاوااسکریپت می باشد. این ویژگی بعد از معرفی ES6 در اختیار توسعه دهندگان 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 می توانید بدون استفاده از کلمه کلیدی function و یا return یک تابع را بسازید. نحوه ایجاد یک Arrow Function به صورت زیر می باشد:

```
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 باشد. 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 میدهد 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 میشوند و فقط نام ندارند و در بعد از آکولاد بسته آنها علامت سیمیکالن); قرار میگیرد. 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 سیمیکالن در جلوی آن قرار دهیم. به کد زیر توجه کنید :

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

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 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 می شوند. به این توابع self-invoking گفته می شود. نحوه ایجاد یک تابع self-invoking به صورت زیر می باشد:

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

(Argument);

به مثال زیر توجه کنید :

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

Hello World!

همانطور که در کد بالا مشاهده می کنید، این توابع در داخل یک جفت پرانتز تعریف می شوند (خطوط 1 و 2) و برای صدا زدن آنها کافیهست که یک جفت پرانتز گذاشته و در داخل این پرانتزها هم آرگومان هایی که لازم است به تابع ارسال شوند را بنویسیم (خط 3). اگر هم تابع هیچ آرگومانی قبول نکند، کافیهست 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 حرکتی 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 میکند.

در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاءمان داریم. شیء در برنامه نویسی شیء گرا از روی کلاسی که شما تعریف کردهاید ایجاد میشود. 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 میباشد. در دنیای واقعی ساختمانها نیز بر اساس یک نقشه (کلاس) پایه گذاری (تعریف) شدهاند.

برنامه نویسی شیء گرا یک روش جدید در برنامه نویسی است که بوسیله برنامه نویسان مورد استفاده قرار میگیرد و به آنها کمک میکند که برنامههایی 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 میشوید.

کلاس

کلاس به شما اجازه میدهد یک نوع دادهای که توسط کاربر تعریف میشود و شامل متغیرها و خواص (properties) و متدها است را ایجاد کنید. کلاس در حکم یک نقشه برای یک شیء میباشد. شیء یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی میکند. وقتی یک شیء میسازید یعنی اینکه یک نمونه از کلاس ساختهایید (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 کلاس‌مان 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      TellInformation()
7      {
8
9          console.log("Name : ", this.name);
10         console.log("Age : ", this.age , " years old");
11         console.log("Height: ", this.height , "cm");
12     }
13 } let firstPerson = new Person();
14 let secondPerson = new Person();
15
16
17
18 firstPerson.name    = "Jack";
19 firstPerson.age     = 21;
20 firstPerson.height  = 160;
21 firstPerson.TellInformation();
22 console.log("\n"); //Separator
23
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 `TellInformation` تعریف شده است که رفتار کلاس را مشخص میکند و مثلاً اگر از فرد سوالی بپرسیم () یک متد هم در داخل کلاس به نام `TellInformation` تعریف شده است که رفتار کلاس را مشخص میکند. 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 می کنید، برای تعریف فیلدها از کلمه `var` و برای تعریف متدها از کلمه `function` استفاده نمی کنیم.

در خطوط 16 و 17 دو نمونه یا دو شیء از کلاس `Person` و `new` ایجاد میکنیم. برای ایجاد یک نمونه از یک کلاس باید از کلمه کلیدی 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 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

سازندهای 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 کلاس استفاده نکنید، جاوااسکریپت از سازنده پیشفرض که یک متد بدون پارامتر است استفاده میکند. برای ایجاد سازنده از متد خاص `constructor` استفاده می کنیم. در مثال زیر یک کلاس که شامل سازنده پیشفرض (خطوط 7-10) است را مشاهده میکنید :

```

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

```

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

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

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

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

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

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

همانطور که مشاهده می‌کنید در مثال بالا سازنده‌های را سه آرگومان قبول میکند به کلاس `Person` اضافه کرده‌ایم (خطوط 7-9). در خطوط 22 و 27 بعد از ایجاد شیء و در داخل پرانتزها سه مقدار را به سازنده (خط 24) ارسال می‌کنیم و سازنده این مقادیر را به فیلدها (خطوط 22) اختصاص میدهد (3-5).

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 var x = new Test();
7
8 console.log(x.number1);
9 console.log(x.#number2);
10
```

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 می کنید، برای تعریف یک فیلد یا متد به صورت خصوصی یا privately کافایت 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 می شود. دلیل آن هم این است که اعضای private خارج از کلاس قابل دسترسی نیستند. حال خط 10 را حذف و دوباره برنامه را اجرا کنید. خروجی عدد 10 را نشان می دهد. چون مقدار فیلد number1 عدد 10 بود و این فیلد به صورت عمومی یا public تعریف شده و در خارج از کلاس قابل دسترسی می باشد. 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 بحث وراثت در درسهای آینده آموزش میدهیم.

کپسوله سازی

کپسوله سازی یا Encapsulation 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     AddFive(number)
5     {
6
7         this.five += number;
8         return this.five;
9     }
10 }
11 var test = new Test();
12 test.five = 10;
13 console.log(test.AddFive(100))
14
15
```

110

متد داخل کلاس Test به نام AddFive() خطوط 9-5 دارای هدف سادهای است و آن اضافه کردن مقدار 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 می دهیم) 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)

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

آمده است `property` در مثال زیر نحوه تعریف و استفاده از:

```

1  class Person
2  {
3      #name
4      #age
5      #height;
6      get Name()
7      {
8      }
9      set Name(value) {
10         this.#name = value;
11     }
12     get Age()
13     {
14         this.#name = value;
15     }
16     set Age(value)
17     {
18     }
19     get Height() {
20         return this.#age;
21     }
22     set Height(value)
23     {
24         this.#age = value;
25     }
26     constructor(name, age, height)
27     {
28         this.#name = name;
29         this.#age = age;
30         this.#height = height;
31     }
32     return this.#height;
33
34     this.#height = value;
35
36     this.#name = name;
37     this.#age = age;
38     this.#height = height;
39 }
40
41 var person1 = new Person("Jack", 21,
42 160);
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 console.log("\n"); //Seperator
49
50
51 console.log("Name : ", person2.Name);
52 console.log("Age : ", person2.Age , "years old" );
53 console.log("Height: ", person2.Height , "cm");
54 console.log("\n"); //Seperator
55
56
57
58 person1.Name = "Frank";
59 person1.Age = 19;
60 person1.Height = 162;
61 person2.Name = "Ronald";
62 person2.Age = 25;
63 person2.Height = 174;
64 console.log("Name
65 console.log("Age
66
67 : ", person1.Name);
68 : ", person1.Age , "years old" );
69 console.log("Height: ", person1.Height , "cm");
70 console.log("\n"); //Seperator
71
72
73 console.log("Name : ", person2.Name);
74 console.log("Age : ", person2.Age , "years old" );
75 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 کرده ایم (سه فیلد با سطح دسترسی private).

```

#name ;
#age ;
#height;

```

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

```

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

this.#age = value;

return this.#height;

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 می کنیم که، شباهت نام `property` ها و فیلدها اجبار نیست و یک قرارداد میباشد.

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

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

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

است. و ما به برنامه میفهمانیم که میخواهیم از طریق `set` این کار (قرار دادن یک مقدار بعد از علامت مساوی) به منزله فراخوانی بخش یک فیلد را مقداردهی کنیم `set` بخش `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;
```

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

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

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

    return 0;
}
```

حال اگر کاربر بخواهد یک مقدار منفی به فیلد age صفر خواهد شد. همچنین میتوان یک age اختصاص دهد مقدار property فقط خواندنی (read-only) ایجاد کرد. این property است. به عنوان مثال میتوان یک خاصیت set فاقد بخش 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 طوریکه فرزند خصوصیات 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 } class classChild extends classParent
12 { } var child = new classChild(); child.publicMessage();
13 //child.#privateField; Error : Private field '#privateField' must be declared in an
14 enclosing class
15
16
17
18
19
20
21
```

This is public Message From Parent Class!

همانطور که مشاهده میکنید در کد بالا دو کلاس تعریف کردهایم: یکی کلاس

```
classParent
```

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 می شود. حال اگر خط 21 را از حالت توضیحات خارج کنیم، یعنی علامت `//` را حذف و برنامه را دوباره اجرا کنیم با خطا مواجه می شویم. 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`

`super()` زمانی که بخواهید سازنده کلاس پدر را فراخوانی کرده و از کدهای آن استفاده کنیم، 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 class Child extends Parent
9 {
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 سازده کلاس فرزند از متد `super()` استفاده کنید :

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

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

Message from parent class!
Hello World!

override

یا باز نویسی، یعنی اینکه ما کاری کنیم که متدهای کلاس پایه در داخل کلاس مشتق رفتار متفاوتی از خود نشان دهند. به `override` عنوان مثال شما متد را در کلاس دارید و کلاس از کلاس ارث بری میکند، در این صورت متد در کلاس در دسترس خواهد دقیق همان متدی است که از کلاس به ارث برده شده است. حال اگر بخواهید که این متد رفتار متفاوتی از خود نشان A بود. اما متد `override` یا باز نویسی 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 class Child extends Person
9 {
10
11     ShowMessage()
12     {
13         super.ShowMessage();
14         console.log("ShowMessage method was overridden !");
15     }
16 }
17 var myPerson = new Person();
18 var myChild = new Child();
19 myPerson.ShowMessage();
20 console.log("\n");
21 myChild.ShowMessage();
22
23
```

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 در داخل کلاس `Child` مینویسیم و سپس با استفاده از کلمه کلیدی `super` در خط 3 به جاوا اسکریپت 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

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

```
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 اجرا میشود. این عملگر همچنین میتواند چک کند که آیا یک شیء خاص در سلسله مراتب وراثت یک نوع خاص است if داخل دستور : 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 کرده ایم که از کلاس Animal ارث میبرد. سپس یک نمونه از این کلاس ایجاد میکنیم و سپس با استفاده از عملگر instanceof(Dog) تست میکنیم که آیا نمونه ایجاد شده جز کلاس Animal است یا یک

ارث میبرد) سگ من یک حیوان است، نتیجه

کلاس مشتق شده از کلاس

Animal کلاس Dog میباشد. از آنجاییکه کلاس Animal

عبارت درست (true) است. حال جمله بالا را تغییر دهیم: "حیوان من یک سگ است". وقتی جمله برعکس میشود چه اتفاقی می افتد؟

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

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

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 SampleClass.ShowStaticMessage();
9
```

static method has been called.

همانگونه که در کد بالا مشاهده میکنید، یک کلاس (خطوط 1-2) - (تعریف کردهایم که دارای یک متد (خطوط 3-5) - 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 می شوید.

باگ (bug) اصطلاحی 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 می باشد، 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 می خواهیم 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>
```

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

```
try { } catch {
}
number;
// This word is not defined

console.log("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 می توان از یکی از کلاس

های خطا در جاوااسکریپت به نام err به شکل زیر استفاده کنید :

```
try { }
catch(err)
{
    number;
}

console.log(err.name);
```

ReferenceError

خاصیت name مربوط به کلاس err نوع خطا را مشخص می کند. همچنین میتوانید اطلاعات مربوط به این استثناء را با استفاده از

خاصیت message نمایش دهید:

```
try { } catch(err) {
}
number;
// number is not defined

console.log(err.message);
```

شما میتوانید از عمگر instanceof نیز به صورت زیر استفاده نمایید

```
try { }
catch(err)
{ eval("a ++ b");

    if (err instanceof ReferenceError)
    {
        if (err instanceof TypeError) { }
        console.log("value is not defined");
    }
    if (err instanceof SyntaxError) { }

    console.log("Format cannot be accepted!");

    console.log("Causes SyntaxError");

}
```

Causes SyntaxError

بلوک `catch` برای به دام انداختن همه استثنائهایی که به وسیله برنامه به وجود میآید استفاده میکند. در داخل بلوک `err` از کلاس `catch` و کلمه کلیدی `if` میتوانید با استفاده از یک دستور `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
{ }
catch(err)
{ }
finally
{ }
    console.log(err.message);

    console.log("finally blocked was reached.");
```

number is not defined
finally blocked was reached.

