به نام خداوند بخشنده مهربان

آموزش طراحی صفحات وب

Java Script

تالیف و گردآوری

مهرداد فتاحی

Developer Studio Network

www.Developer1.ir

مهر 1391

معرفی Java Script :

در قسمت آموزش HTML ، با زبان طراحی صفحات وب آشنا شدید . به وسیله امکانات زبان HTML ، می توان انواع صفحات وب را با کلیه اجزای مورد نیاز از قبیل نوشته ها ، جداول ، تصاویر ، فرم ها و ... را ایجاد کرد . اما HTML صرفا یک زبان طراحی است و توانایی برنامه سازی ، کنترل فرم ها ، پاسخ به رویدادهای برنامه و عملکرد کاربر و ... را ندارد . به همین دلیل باید از یک زبان برنامه نویسی در صفحات وب استفاده کرد . Java Script یکی از زبان های برنامه نویسی اسکریپتی است ، که اولین بار توسط شرکت Netscape Communicator عرضه کننده مرورگر معروف**Netscape** ارائه شد و امروزه متداولترین زبان اسکریپت نویسی صفحات وب است .

توجه : قبل از مطالعه قسمت آموزش Java Script ، شما باید به طور کامل با زبان HTML و به خصوص تگ < script > آشنا باشید . برای مطالعه قسمت های ذکر شده به بخش آموزش HTML و بخش آموزش تگ < script > بروید .

خصوصیات مهم Java Script :

- Java Scriptیک زبان برنامه نویسی اسکریپتی است . دستور العمل های زبان های اسکریپتی ، در کامپیوتر کاربر و توسط مرورگر اجرا شده و برای اجرا نیازی به برنامه کمکی خاصی ندارند . به این زبان ها در اصطلاح طرف مشتری (Client Side) می گویند . در مقابل زبان های مثل ASP.NET ابتدا توسط سرور ارسال کننده وب اجرا شده و سپس نتایج خروجی به زبان HTML برای اجرا در مرورگر فرستاده می شود . به این زبان ها در اصطلاح طرف‌سرور (Server Side) می گویند.
- زبان های اسکریپتی ، جزء زبان های برنامه نویسی سبک هستند . این زبان ها در هنگام اجرا فازی به **نام کامپایل** *را طی نکرده و دستورات آن ها به صورت خط به خط اجرا می شوند.

کامپایل : برنامه های نوشته شده به زبان های برنامه نویسی مثل C , VB یا C# در هنگام اجرا ابتدا توسط کامپایلر به طور کامل خوانده شده و اشکال زدایی می شوند و در صورت عدم وجود اشکال ، اجرا خواهند شد . اما برنامه های نوشته شده به زبان های اسکریپتی ، به صورت خط به خط توسط مرورگر خوانده شده و اجرا می شوند .

برخی از امکانات Java Script :

- Java Script به طراحان وب ، یک ابزار برنامه نویسی ساده و کارا می دهد.
- Java Script به رویدادهای مختلف در صفحه واکنش نشان می دهد . برای مثال می توان یک تابع Java Script تعریف کرده تا در هنگام وقوع یک رویداد مثل کلیک بر روی یک دکمه یا لود شدن صفحه ، اجرا شود.
- Java Script می تواند اطلاعات ارسالی یک فرم را اعتبار سنجی و کنترل نموده و در صورت صحیح بود ، آنها را به سرور ارسال کند . این کار باعث جلوگیری از ورود اطلاعات نادرست به سرور و کاهش ترافیک آن می شود.
- Java Script توانایی تشخیص نوع و نسخه مرورگر مورد استفاده کاربر را داشته و می تواند بر حسب آن نوع مرورگر خاص ، تنظیمات و صفحات ویژه ای را بارگذاری نماید.

- Java Script توانايي خواندن و نوشتن اطلاعات مورد نياز مرورگر را بر روی کامپيوتر بازديد کننده صفحه را داراست ، که در اصطلاح به اين کار ايجاد و خواندن Cookie می گويند.
- Java Script می تواند انواع کادرهای اخطار ، تاييد و دريافت ورودی را به کاربر نمايش دهد.

تفاوت Java و Java Script :

زبان های برنامه نويسی Java و Java Script دارای ساختار دستوری مشابه به هم هستند ، ولی ۲ زبان کاملا مجزا هستند . Java يک زبان شی گرا قدرتمند برای برنامه نويسی تحت ويندوز است ، در حالی که Java Script يک زبان ساده اسکريپت نويسی در مرورگر های وب است .

دستورات مقدماتی Java Script

نحوه تعریف دستورات Java Script در صفحه :

برای تعریف و ایجاد یک اسکریپت ، از تگ < script > استفاده می شود . کلیه دستورات مورد نظر اسکریپت ، درون تگ باز و بسته < script > تعریف شده و به عبارتی محدوده کدهای اسکریپت را تعیین می کند . در هر صفحه HTML ، می توان به تعداد مورد نیاز از تگ < script > استفاده کرد ، که هر تگ به صورت مجموعه ای واحد برای خود عمل می کند .

توجه : قبل از مطالعه قسمت آموزش Java Script ، شما باید به طور کامل با زبان HTML و به خصوص تگ < script > آشنا باشید . برای مطالعه قسمت های ذکر شده به بخش آموزش HTML و بخش آموزش تگ < script > بروید .

مثل : در مثال زیر یک اسکریت ساده در صفحه قرار داده شده است . به نکات زیر دقت کنید :

- **نکته ۱ :** علاوه بر Java Script ، زبان های اسکریپتی دیگر از قبیل VB Script یا EcmaScript نیز وجود دارند و در هر تگ < script >باید به وسیله خاصیت Type نوع و زبان اسکریپتی مورد استفاده در ان اسکریپت را تعیین کرد . در مثال زیر زبان اسکریپت Java Script و نوع آن متن تعیین شده است.
- **نکته ۲ :** در مثال زیر از دستور document.write **استفاده شده است** . این دستور برای نمایش یک متن خروجی که در پرانتز جلوی آن تعیین می شود ، در صفحه به کار می رود . در ادامه با شی document **و خواص و** متدهای آن آشنا خواهید شد.

Example	
< html > < head > < title > Title of Page عنوان صفحه < /title > < /head > < body > < script type="text/javascript" > document.write ("This is an Script !") < /script > < /body > </html>	کد
This is an Script !	خروجی

انواع حالت های دستورات اسکریپتی :

به طور کلی ۲ حالت اسکریپت (برنامه اجرایی) در صفحات وب قابل پیاده سازی است :

۱. اسکریپت های که می خواهیم به محض لود شدن صفحه ، اجرا شده و اثر خود را نمایش دهند . در این حالت باید اسکریپت ها را در قسمت < body > صفحه قرار داد.
۲. اسکریپت هایی که می خواهیم در هنگام بروز یک رویداد در صفحه ، مثل کلیک بر روی یک دکمه خاص و ... اجرا شوند . به عبارت دیگر می خواهیم اجرای آنها کنترل شده باشد . در این حالت دستورات اسکریپت را در قسمت < head > صفحه و یا در یک فایل خارجی تعریف کرد.

نکته : گاهی اوقات می خواهیم از دستورات اسکریپتی یکسان و مشترک در تمام یا گروهی از صفحات یک وب سایت استفاده کنیم . در این حالت برای جلوگیری از تکرار دستورات در تمام صفحه های مذکور ، کاهش حجم کد نویسی ، افزایش سرعت طراحی و اعمال تغییرات سریع و آسان به دستورات ، بهتر است همه آن اسکریپت ها را یکبار در یک فایل خارجی تعریف کرده و از آن به طور مشترک در تمام صفحات استفاده کرد . برای دریافت اطلاعات بیشتر به روش سوم قرار دهی اسکریپت در صفحات وب در پایین صفحه بروید .

محل قرار دادن اسکریپت ها در صفحات وب :

به طور کلی ۳ روش برای قرار دادن اسکریپت ها در صفحات وب وجود دارد :

1 . درون محدوده اصلی صفحات HTML ، درقسمت تگ : < body >

اسکریپت های تعریف شده در این قسمت ، به محض بارگذاری و نمایش صفحه اجرا شده و خروجی خود را تولید می کنند . این نوع اسکریپت ها می توان در هر جای محدوده تگ < body > صفحات تعریف کرد . در این نوع اسکریپت ها ، هیچ کنترلی از سوی کاربر برای اجرای آنها وجود ندارد ، مگر اینکه دستورات آن در قالب یک تابع (function) تعریف شده باشند ، که تا زمان فراخوانی آن تابع اجرا نخواهند شد . برای دریافت اطلاعات بیشتر در مورد توابع جاوا اسکریپت ، به قسمت تعریف توابع در جاوا اسکریپت بروید .

مثل : در مثال زیر یک اسکریپت ساده در قسمت تگ < body > صفحه ایجاد شده است . این اسکریپت به محض لود شدن صفحه اجرا شده و خروجی خود را تولید می کند .

Example	
< html > < head >	کد

< title > Title of Page عنوان صفحه < /title > < /head > < body > < script type="text/javascript" > document.write ("This script is placed in the body section . ") < /script > < /body > </html>	
This script is placed in the body section .	خروجی

۲ . درقسمت تگ : < head >

اسکریپت هایي که می خواهیم در هنگام بروز یک رویداد در صفحه مثل کلیک بر روی یک دکمه و ... اجرا شوند ، را می توان در قسمت < head > تعریف کرد . دستورات اسکریپت های این قسمت بایستی در قالب توابع تعریف شده باشند و تا زمانی که از سوی برنامه یا کاربر فراخوانی نشده باشند ، اجرا نخواهند شد . مزیت این روش در این است ، که این اسکریپت ها قبل از اجرا توسط مرورگر لود شده اند . فراخوانی باید توسط یک اسکریپت دیگر در قسمت تگ < body > صورت بگیرد .

مثل : در مثال زیر یک اسکریپت ساده در ۲ حالت درقسمت تگ < head > صفحه تعریف شده است . در حالت اول به دلیل عدم فراخوانی تابع اسکریپت ، آن اسکریپت هیچ گاه اجرا نخواهد شد . اما در حالت دوم تابع hello() توسط رویداد onclick (کلیک) دکمه فرمان فراخوانی شده و خروجی خود را نمایش می دهد . برای اجرای اسکریپت بر روی دکمه فرمان مثال کلیک نمایید :

Example	
حالت ۱ . در این حالت به دلیل عدم فراخوانی اسکریپت هیچ خروجی نداریم .	

< html >
< head >
< title > Title of Page عنوان صفحه < /title >
< script type="text/javascript" >
function hello ( )
{
document.write ("Hello User . This script is placed in the head section .")
}
	کد

< /script > < /head > < body > محتویات صفحه < /body > </html>	
	خروجی

حالت ۲ . در این حالت با فراخوانی اسکریپت توسط رویداد کلیک دکمه فرمان ، دستور آن اجرا می شود .

< html >
< head >
< title > Title of Page عنوان صفحه < /title >
< script type="text/javascript" >
function hello ( )
{
document.write ("Hello User . This script is placed in the head section . ")
}
< /script >
< /head >
< body >
محتویات صفحه
< input value ="to view script Click me" id="Button1" type="button" onclick="hello( )" / >
< /body >
</html>
	کد

	خروجی

۳ . در یک فایل خارجی: JS

در این حالت تمام اسکریپت های مورد نظر را در یک فایل خارجی متنی با پسوند JS ، تعریف کرده و سپس به وسیله تگ < script > در قسمت < head > صفحه ، بین آن فایل و صفحه لینک ایجاد می کنیم .

از این جالت معمولا در مواردی که بخواهیم کدهای اسکریپت را از محتویات صفحات HTML جدا کرده و یا از یک سری دستورات و توابع اسکریپتی مشترک در چند صفحه استفاده کنیم ، کاربرد دارد .

اسکریپت های این حالت باید در قالب توابع مختلف تعریف شده و تا زمانی که از سوی برنامه یا کاربر فراخوانی نشوند ، اجرا نخواهند شد .

مثل : در مثال زیر ابتدا یک اسکریپت در یک فایل خارجی به نام myscript.js تعریف کرده و سپس بین صفحه و آن فایل ارتباط ایجاد کرده ایم . **تابع** ( ) hello2 توسط رویداد کلیک دکمه فرمان فراخوانی و اجرا می شود . برای اجرای آن بر روی دکمه فرمان کلیک کنید :

متن فایل myscript.js

< script type="text/javascript" >
function hello2 ( )
{
document.write ("This script is placed in an external Java Script file . ")
}
< /script >

Example	
< html > < head > < title > Title of Page عنوان صفحه < /title > < script type="text/javascript" src ="../myscript.js" > * ایجاد لینک بین صفحه و فایل اسکریپت * < /script > < /head > < body > محتویات صفحه < input value="to view script Click me" id="btnhello2" type="button" onclick="hello2( )" /> > < /body > </html>	کد
	خروجی

دستورات مقدماتی Java Script

نکات مهم در کد نویسی جاوا اسکریپت

۱. جاوا اسکریپت به بزرگ یا کوچک بودن حروف حساس است (برخلاف HTML) :

در تعریف و نام گذاری توابع و متغیرها در جاوا اسکریپت باید به بزرگ یا کوچک بودن حروف کاملا دقت کرد . **تابع با نام** "MyFunction" با تابع "myfunction" و متغیر با نام "Matn" با متغیر با نام "matn" متفاوت هستند .
همچنین کلیه دستورات جاوا اسکریپت باید به صورت استاندارد تعیین شده ، با حروف بزرگ یا کوچک نوشته شود . **عدم** رعایت این نکته باعث اجرا نشدن دستور و بروز خطا در صفحه می شود . هر یک از دستورات و کلمات کلیدی در جاوا اسکریپت فقط به یک صورت ، که صورت استاندارد است باید نوشته شوند .
توجه : در مثال ها و کدهای بخش آموزش ، شکل صحیح نوشتاری کلیه دستورات نمایش داده شده است .

مثل ۱ : در مثال زیر ۲ متغیر با نام های یکسان ، ولی متفاوت در بزرگ یا کوچک بودن حروف به نام های "Matn" و "matn" ایجاد و مقدار دهی شده اند . خروجی کد نشان می دهد که این دو متغیر کاملا با هم متفاوت هستند و هر یک مقدار مخصوص به خود را دارند :

Example	
<script type="text/javascript"> var matn = "This is a Variable ." ; var Matn = "This is another Variable ." ; document.write (matn) ; document.write (Matn) ; </script>	کد
This is a Variable .This is another Variable .	خروجی

مثل ۲ : شکل صحیح نوشتن متد چاپ خروجی در جاوا اسکریپت به صورت documnet.write **است** . در مثال زیر ابتدا دستور را به شکل نادرست و با حروف بزرگ به صورت Documnet.Write نوسته ایم ، که باعث ایجاد خطا در صفحه شده و خروجی نداریم . اما در حالت دوم آنرا به شکل صحیح نوشته ایم ، که خروجی درست را نیز مشاهده می کنیم :

Example	
حالت اول ، شکل نادرست	
<script type="text/javascript"> var Str = "An Investigation for Development" ; Document.Write (Str) ;	کد

</script>	
	خروجی

حالت دوم ، شکل صحیح

<script type="text/javascript"> var Str = "An Investigation for Development" ; document.write (Str) ; </script>	کد
An Investigation for Development	خروجی

---

۲ . جاوا اسکریپت فواصل خالی اضافی در کد نویسی را نادیده می گیرد:

به کار بردن فاصله خالی اضافی در بین کلمات کد های جاوا اسکریپت ، از سوی مرورگر نادیده گرفته شده و تاثیری ندارد . می توان برای خواناتر شدن برنامه ، بین کلمات فاصله اضافی ایجاد کرد .

نکته : بین دستورات و کلمات کلیدی باید حداقل یک فاصله وجود داشته باشد ، در اینجا منظور از فاصله اضافی ، بیش از یک کاراکتر فاصله است .

مثل : در مثال زیر یک تک کد را در حالت اول بدون فاصله و در حالت دوم با فاصله بین کلمات نوشته ایم . همانطور که در خروجی مشخص است ، این دو قطعه کد کاملا یکسان بوده و هیچ تفاوتی با هم ندارند :

Example

<script type="text/javascript"> var StrName="Developer Studio"; document.write(StrName); </script>	<script type="text/javascript"> var StrName = "Developer Studio" ; document.write (StrName) ; </script>	کد
Developer Studio	Developer Studio	خروجی

---

۳ . نوشتن عبارت های متنی در بیش از یک خط:

در هنگام تعریف و استفاده از عبارت های متنی در دستوراتی نظیر document.write و ... ، می توان ادامه متن را به کمک یک کاراکتر \ به سطر بعدی انتقال داد . این مسئله در زمانی که عبارت های متنی طولانی استفاده می شود ، کاربرد دارد .

مثل : در مثال زیر می خواهیم یک پیام را به کاربر اعلام کنیم . ولی به دلیل طولانی بودن متن پیام تصمیم گرفته ایم ، آن را در بخش کدنویسی در ۲ خط تعریف کنیم ، اما می بینیم که مرورگر در خروجی آنرا در یک خط نشان می دهد :

Example	
<script type="text/javascript"> document.write ("Java Script is a client side language . \ It's codes executes in the computer of visitor ") ; </script>	کد
Java Script is a client side language . It's codes executes in the computer of visitor	خروجی

---

٤ . درج توضیحات (comments) مورد نظر در بخش کد نویسی:

در اسکریپت های نوشته شده به زبان جاوا اسکریپت ، می توان توضیحات مورد نظر را به صورت ویژه ای در درون کدها وارد کرد . این توضیحات به طور کامل از سوی مرورگر نادیده گرفته شده و در خروجی نمایش داده نمی شوند . از توضیحات معمولا برای نشانه گذاری یا ارائه توضیحاتی راجع به کدهای برنامه استفاده می شود .
دو نوع توضیح در جاوا اسکریپت می توان ایجاد کرد :

۱ . توضیحات یک خطی : این توضیات به کمک دو بک اسلش // به صورت زیر وارد می شود . توضیحات ارائه شده به این صورت حداکثر می تواند در یک خط باشد . به مثال زیر دقت کنید . در این مثال خط اول یک comment خط دوم یک دستور چاپ خروجی است . همانطور که در خروجی کد مشخص است ، دستور چاپ توسط مرورگر اجرا شده ولی comment نمایش داده نمی شود :

Example	
<script type="text/javascript"> // this is a one line comment . navigator won't show it . document.write ("How to write a comment") ; </script>	کد
How to write a comment	خروجی

۲ . توضیحات چند خطی : با استفاده از یک نماد */ در ابتدای اولین خط توضیحات و یک نماد /* در آخرین خط توضیحات ، می توان توضیحات چند خطی در اسکریپت ها وارد کرد . از این حالت برای ارائه توضیحات طولانی استفاده می شود . به مثال زیر دقت کنید . در این مثال هم یک دستور و یک comment چند خطی قرار داده شده است . **دستور توسط مرورگر اجراشده** ، ولی comment نمایش داده نمی شود :

Example	
<script type="text/javascript"> /* this is a multi line comment . navigator won't show it . We use it for long comments . It can be several lines */ document.write ("How to write a multi line comment") ; </script>	کد
How to write a multi line comment	خروجی

دستورات مقدماتی Java Script

عملگرهای جاوا اسکریپت :

در این قسمت ، به معرفی و توضیح عملگرهای مورد استفاده در جاوا اسکریپت می پردازیم .

توضیح عملوند : عملوند به متغیری گفته می شود ، که عملگر بر روی آن عملیات انجام می دهد .

۱ . عملگرهای ریاضی:

عملگر	شرح	مثل
+ جمع	دو عملوند خود را با هم جمع می کند .	x = 2 y = 2 x + y = 4
- تفریق	دو عملوند خود از هم کم می کند .	x = 4 y = 2 x - y = 2
* ضرب	دو عملوند خود را در هم ضرب می کند .	x = 2 y = 3 x * y = 6
/ تقسیم	عملوند اول خود را بر عملوند دوم تقسیم می کند .	x = 6 y = 2 x / y = 3
% باقی مانده	باقی مانده حاصل از تقسیم عملوند اول بر عملوند دوم را محاسبه می کند .	x = 8 y = 3 x % y = 2 x = 15 y = 4 x % y = 3 x = 9 y = 3 x % y = 0
++ افزاینده	عملوند خود را یک واحد افزایش می دهد .	x = 7 x++ x = 8
- - کاهنده	عملوند خود را یک واحد کاهش می دهد .	x = 8 x- - x = 7

۲ . عملگرهای انتسابی:

از عملگرهای انتسابی ، برای نسبت دادن مقدار به یک متغیر استفاده می شود .

نکته : برخی از حالت های محاسبات متغیرها مثل x = x + y را می توان به صورت خلاصه تر به صورت x += y **نوشت** .

در جدول زیر انواع حالت های آن آمده است :

عملگر	مثل	برابر است با
= انتساب	x = 5 یا x = y	x = 5 نتیجه : y = x y = 5 var x ; x = 5 (انتساب مقدار ۵ به x)
+= جمع	x += y	x = x + y
- = تفریق	x - = y	x = x - y

ضرب *=	x *= y	x = x * y
تقسیم /=	x /= y	x = x / y
باقی مانده %=	x %= y	x = x % y

۳ . عملگرهای مقایسه ای:

از این عملگرها برای مقایسه یک متغیر با یک مقدار و یا مقایسه ۲ متغیر با هم استفاده می شود .

عملگر	شرح	مثال
= = تساوی	امتحان برابری با یک مقدار یا یک متغیر دیگر	x = = y یا y = = 8
= = = تساوی	امتحان برابری با یک مقدار یا یک متغیر دیگر هم از لحاظ مقدار و هم از لحاظ نوع داده ای	x = = = y یا y = = = "8" مثل: x = 5 ، y = "5" ، اگر x = = = y نتیجه : غلط است
! = عدم تساوی	امتحان عدم برابری با یک مقدار یا یک متغیر دیگر	x ! = y یا y ! = 4 نتیجه : x ! = y اگر ، y = 6 ، x = 5 : مثل درست است
< بزرگتر بودن	امتحان بزرگتر بودن	x > y یا y > 4 نتیجه : x > y اگر ، y = 6 ، x = 5 : مثل غلط است
> کوچکتر بودن	امتحان کوچکتر بودن	x < y یا y < 4 نتیجه : x < y اگر ، y = 6 ، x = 5 : مثل درست است
مساوی یا بزرگتر >= بودن	امتحان مساوی یا بزرگتر بودن	x >= y یا y >= 4 نتیجه : x >= y اگر ، y = 6 ، x = 5 : مثل غلط است
مساوی یا کوچکتر <= بودن	امتحان مساوی یا کوچکتر بودن	x < y یا y < 4 نتیجه : x <= y اگر ، y = 5 ، x = 5 : مثل درست است نتیجه : x <= y اگر ، y = 7 ، x = 5 : مثل غلط است

٤ . عملگرهای منطقی:

از عکلگرهای منطقی برای ترکیب دو یا چند عبارت مقایسه ای یا شرطی با هم و ایجاد یک عبارت واحد استفاده می شود . در جدول زیر انواع عملگرهای منطقی و شرایط درست بودن آنها توضیح داده شده است .

عملگر	شرح	مثال
&& عملگر " و "	این عبارت برای ترکیب دو یا چند عبارت با هم استفاده می شود . نتیجه ترکیب این عملگر فقط زمانی صحیح است ، که تمام عبارات ترکیب شده با هم درست باشند .	مثال : x = 5 , y = 7 , اگر (x < 3 && y > 9) نتیجه : غلط است مثال : x = 5 , y = 7 , اگر (x < 6 && y > 8) نتیجه : درست است
|| عملگر " یا "	این عبارت برای ترکیب دو یا چند عبارت با هم استفاده می شود . نتیجه ترکیب این عملگر در صورت درست بودن حداقل یکی از عبارات ترکیب شده ، درست خواهد بود.	مثال : x = 5 , y = 7 , اگر (x < 3 || y > 4) نتیجه : درست است مثال : x = 5 , y = 7 , اگر (x < 3 || y > 8) نتیجه : غلط است
! عملگر not	این عملگر برای بر عکس کردن درستی یا عدم درستی یک عبارت استفاده می شود . استفاده این عملگر قبل از یک عبارت صحیح باعث نادرست شدن جواب و برعکس خوهد شد .	مثال : x = 5 , y = 5 , اگر ! (x == y) : نتیجه غلط است مثال : x = 5 , y = 7 , اگر ! (x == y) : نتیجه درست است

٥ . عملگر رشته ای : متغیرهای رشته ای متغیر هایی هستند ، که از متن تشکیل شده اند . این متغیرها را همانطور که قبلا اشاره شد ، باید بین دو علامت " " تعریف کرد .

در جاوا اسکریپت می توان دو متغیر رشته ای را با عملگر + به هم اضافه کرد . همچنین برای ایجاد فاصله بین متغیرهای می توان از یک " " به شکلی که در مثال زیر آمده است ، استفاده کرد .

مثل : در مثال زیر دو عبارت رشته ای matn1 و matn2 را در قالب یک متغیر جدید به نام welcome ذخیره کرده ایم :

Example

کد

```
< script type="text/javascript" >
  var matn1 = "Welcome to" ;
  var matn2 = "Developer Studio" ;
  var welcome = matn1 + " " + matn2 ;
  document.write (welcome) ;
< /script >
```

خروجی

```
Welcome to Developer Studio
```

نمایش کاراکترهای خاص در جاوا اسکریپت :

در کد نویسی دستورات جاوا اسکریپت ، از برخی از کاراکترها به منظور ویژه های استفاده می کنیم . برای مثال از کاراکتر " برای شروع یک عبارت متنی در دستوراتی نظیر document.write و ... استفاده می شود . به کار بردن مستقیم چنین کاراکترهای در عبارت های متنی باعث تداخل با کدهای برنامه و بروز خطا و خروجی نامناسب می شود .
برای نمایش چنین کاراکترهایی در عبارت های متنی ، باید از یک کاراکتر \ قبل از کاراکتر مورد نظر استفاده کرد .

مثل : برای مثال می خواهیم در اسکریپت زیر یک پیام به کاربر اعلام کنیم . می خواهیم در متن پیام ، عبارت Developer Studio در بین دو کاراکتر " " قرار بگیرد .در حالت اما به دلیل تداخل این کاراکتر ها با شکل دستوری آنها در دستور document.write ، می بینیم که دارای خروجی نادست بوده و پیام فقط خروجی بر روی صفحه چاپ نمی شود . **در حالت** دوم از یک **قبل از** " استفاده شده و می بینیم که دراری خروجی مورد نظر هستیم و پیام به شکل صحیح نمایش داده شده است :

Example	
حالت اول ، شکل نادرست	
<script type="text/javascript"> var matn = "Hello Welcome to "developer Studio " a website for Developers" ; document.write (matn) ; </script>	کد
	خروجی
حالت دوم ، شکل صحیح	
<script type="text/javascript"> var matn = "Hello Welcome to \"developer Studio\" a website for Developers" ; document.write (matn) ; </script>	کد
Hello Welcome to "developer Studio" a website for Developers	خروجی

ایجاد یک خط جدید در نوشته:

می توان در متن نوشته ی کادر های Pop-Up در جاوا اسکریپت ، نوشته را به سطر پایین انتقال داد . برای این منظور از یک کاراکتر \n استفاده می شود . هر بار استفاده از این کاراکتر باعث انتقال نوشته به یک سطر پایین تر می شود .

مثل : در مثال زیرمتن پیام یک کادر اخطار اسکریپت را در ٤ خط نمایش داده ایم .

Example	

کد	<script type="text/javascript">
	function New_Line ( )
	{
	alert ("Hello \nDear User \n\n Welcome to DeveloperStudio") ;
	}
	</script>
	
	<input type="button" id="Button1" value="Click Me !" onclick="New_Line( )" />

سایر کاراکترها:

سایر کاراکترهایی که برای نمایش آنها باید از روش فوق استفاده کرد ، به همراه توضیح آنها در جدول زیر آمده اند :

شرح	خروجی	نحوه استفاده
علامت نقل قول تکی '	'	\'
علامت نقل قول جفتی "	"	\"
علامت و در انگلیسی &	&	\&
علامت اسلش در متن \	\	\\
رفتن به خط جدید در متن		\n

دستورات مقدماتی Java Script

ساختارهای شرطی در جاوا اسکریپت :

از ساختارهای شرطی در زمانی استفاده می شود که بخواهیم در صورت بر قرار بودن شرط یا شرط هایی ، یکسری دستورات خاص اجرا شده و در صورت عدم بر قراری آنها گروه دیگری از دستورات اجرا شوند .
بر حسب شرایط می توان از یکی از ساختارهای دستوری زیر استفاده کرد :

دستور یا دستورات = Statment * شرط یا شروط = Condition * توجه :

۱) Statment (Condition) if :

از این ساختار در مواقعی که می خواهیم در صورت بر قرار بودن شرط یا شرط هایی یکسری دستورات خاص اجرا شوند ، استفاده می شود . در این حالت در صورت عدم بر قراری شرط های تعیین شده ، هیچ دستوری اجرا نخواهد شد .
شکل کلی استفاده از این ساختار به صورت زیر است :

```
if  ( شرط یا شروط )
 {
   دستورات مورد نظر که در صورت برقرار بودن شرط ها اجرا می شوند
 }
```

مثل : در مثال زیر متغیر عددی IntNum یکبار با عددی بیش از ۱۰ و یکبار با عددی کوچکتر از ۱۰ مقدار دهی شده است . **شرط دستور if** این است ، که در هنگام بزرگتر بودن IntNum **از عدد** ۱۰ پیغام "This Number is bigger than 10" **و** در هنگام کوچکتر بودن آن از عدد ۱۰ ، هیچ خروجی چاپ نشود . به مثال دقت کنید :

Example	
در حالت اول به دلیل بزرگتر بودن عدد IntNum از ۱۰ ، پیغام خروجی بر روی صفحه چاپ می شود .	

کد	< script type="text/javascript" >
	 var IntNum = 18
	 if (IntNum > 10)
	 {
	 document.write (" This Number is bigger than 10 ")
	 }
	< /script >

خروجی	This Number is bigger than 10
در حالت دوم به دلیل کوچکتر بودن عدد IntNum از ۱۰ ، هیچ پیغام خروجی بر روی صفحه چاپ نمی شود .	

کد	< script type="text/javascript" >
	var IntNum = 8
	if (IntNum > 10)
	{
	document.write (" This Number is bigger than 10 ")
	}
	< /script >

۲) if (Condition) Statment 1 else Statment 2 :

از این ساختار در مواقعی استفاده می کنیم که می خواهیم در صورت بر قرار بودن شرط یا شرط هایی ، یکسری دستورات و در صورت عدم بر قراری آن شروط ، گروهی دیگر از دستورات اجرا شوند .
شکل کلی استفاده از این ساختار به صورت زیر است :

```
if ( شرط یا شروط )
 {
  دستوراتی که در صورت بر قرار بودن شرط یا شروط اجرا می شوند
 }
else
 {
  دستوراتی که در صورت عدم بر قراری شرط یا شروط اجرا می شوند
 }
```

مثل : در مثال زیر متغیر عددی IntNum یکبار با عددی بیش از ۱۰ و یکبار با عددی کوچکتر از ۱۰ مقدار دهی شده است . **شرط دستور** if این است ، که در هنگام بزرگتر بودن IntNum **از عدد** ۱۰ پیغام "This Number is bigger than 10" و در هنگام کوچکتر بودن آن از عدد ۱۰ ، پیغام "This Number is smaller than 10" **چاپ‌شود** . به مثال دقت کنید :

Example

در حالت اول به دلیل بزرگتر بودن عدد IntNum از ۱۰ ، پیغام خروجی قسمت if بر روی صفحه چاپ می شود .

کد

```
< script type="text/javascript" >
 var  IntNum  = 18
 if ( IntNum > 10 )
  {
    document.write (" This Number is bigger than 10 ")
  }
 else
  {
    document.write ("This Number is smaller than 10")
  }
< /script >
```

خروجی

```
This Number is bigger than 10
```

در حالت دوم به دلیل کوچکتر بودن عدد IntNum از ۱۰ ، پیغام قسمت else بر روی صفحه چاپ می شود .

کد

```
< script type="text/javascript" >
 var  IntNum  = 8
 if ( IntNum > 10 )
  {
    document.write (" This Number is bigger than 10 ")
  }
 else
  {
    document.write ("This Number is smaller than 10")
  }
< /script >
```

خروجی

```
This Number is smaller than 10
```

***عملگر شرطی :**

دستور if تک شرطی را مانند یک عملگر می توان به صورت زیر نیز نوشت .

در این حالت برنامه شرط معرفی شده در پرانتز را چک کرده و در صورت درست بودن آن شرط ، مقدار ۱ و در صورت درست نبودن آن مقدار ۲ را به متغیر نسبت می دهد .

شکل کلی استفاده از این ساختار به سورت زیر است :

نام متغیر = (شرط)؟ مقدار ۱ : مقدار ۲

variable name = (condition) ? value 1 : value 2 ;

Example	
کد	var name = (x > 10) ? sam : david
توضیح	در مثال بالا اگر مقدار متغیر x از ۱۰ بیشتر باشد ، مقدار متغیر name برابر sam و در صورت کوچکتر بودن مقدار x از ۱۰ ، name برابر david می شود .

---

۳) if (Condition 1) Statment 1 else if (Condition 2) Statment 2 else Statment 3 :

از این ساختار زمانی استفاده می شود که حالت های شروط مورد نظر بیش از یک حالت مختلف است ، که در صورت بر قرار بودن هر گروه از شرط های مورد نظر ، می خواهیم دستورات خاص آن شرط ها اجرا شوند .

در این ساختار ،شرط اول در دستور if ابتدا تعریف شده و هر یک از گروه شرط های دیگر به وسیله یک دستور else if تعیین می شود . در آخر نیز واژه کلیدی else و دستورات مربوط با آن قرار می گیرد ، که در صورت عدم بر قراری تمام گروه شرط هاي تعیین شده ، دستورات قسمت else اجرا می شوند .

در این ساختار چنانچه هر یک از شرط های یک دستور if یا else if درست باشند ، دستورات مربوط به آن اجرا شده و برنامه از کنترل و اجرای سایر شرط های دیگر خودداری می کند . چنانچه هیچ یک از گروه شرط های تعیین شده درست نباشند ، دستورات بخش else در پایان ساختار اجرا می شوند .

شکل کلی استفاده از این ساختار به صورت زیر است :

```
if  ( گروه شرط های شماره ۱ )
 {
   دستوراتی که در صورت بر قرار بودن گروه شرط های ۱ اجرا می شوند
 }
```

```
else if  ( گروه شرط های شماره ۲ )
 {
  دستوراتی که در صورت بر قرار بودن گروه شرط های ۲ اجرا می شوند
 }
else if  ( شرط های شماره ۳ گروه )
 {
  دستوراتی که در صورت بر قرار بودن گروه شرط های ۳ اجرا می شوند
 }
.
.
.
else
 {
  دستوراتی که در صورت عدم بر قراری تمام گروه شرط های فوق اجرا می شوند
 }
```

نکته ۱ : به تعداد مورد نیاز می توان در این ساختار به وسیله دستور else if شرط و دستورات جدید تعریف کرد .

نکته ۲ :استفاده از کروشه در قسمت دستورات شرط if ، فقط در زمانی که دستورات مورد نظر بیش از یک خط هستند ، ضروری است .

نکته ۳ : تعیین قسمت else در ساختار فوق اختیاری بوده و می تواند تعریف **نشود** .

نکته ٤ : در زمانی که تعداد حالت های شروط بسیار زیاد هستند ، بهتر است از ساختار Switch استفاده کرد .

مثل : در مثال زیر یک کادر متن و یک دکمه فرمان قرار داده شده است . کادر متن ورودی را از کاربر دریافت کرده و. با کلیک بر روی دکمه فرمان ، خروجی را در کادر آبی رنگ می بینیم .

شکل عملکرد این مثال به صورت زیر است :

رویداد کلیک دکمه فرمان تابع ifclause ( ) را که در قسمت < head > **قرار** دارد ، را فراخوانی می کند . در این تابع یک **دستور** if سه شرطی قرار داده شده است ، که مقدار متغیر matn را ارزیابی کرده و در صورتی که مقدار آن برابر ۱ **باشد** ، عبارت one ، در صورت ۲ بودن عبارت two ، در صورت ۳ بودن عبارت three و در صورتی که مقدار آن به غیر از یکی از موارد فوق باشد ، عبارت bigger than 3 را نمایش می دهد .

Example
نکات مهم : نکات زیر در مثال با شماره مشخص شده اند . ۱. خاصیت value در یک کادر متن به متن موجود در آن کنترل اشاره می کند . در اینجا ما متن موجود در کنترل txtinput را در متغیر matn ذخیره کرده ایم . ۲. رویداد onclick رویدادی است که در هنگام کلیک بر روی یک کنترل فعال می شود . برای فراخوانی یک تابع توسط یک کنترل ، نام آن تابع

را در رویداد onclick کنترل قرار می دهیم .

کد

```
< html >
  < head >
    < title > عنوان صفحه < /title >
    < script type="text/javascript" >
      function       ifclause ( )            ابتدای تعریف تابع
      {
        var matn ;            تعریف متغیر
1       var matn = txtinput.value ;          قرار دادن متن کادر متن در متغیر
        if (matn == '1')
            document.write ("one")
        else if ( matn =='2')
            document.write ( "two" )
        else if ( matn == '3' )
            document.write ( "three" )
        else
            document.write ( "Bigger than three" )
      }
    < /script >
  < /head >
  < body >
    < input type="text" id="txtinput" / >          تعریف کادر متن
2   < input type="button" id="btnclick" onclick="ifclause( )" value="click me" / >          تعریف دکمه فرمان
    < /body >
< /html >
```

ساختار کنترلی switch :

ار این ساختار در زمانی استفاده می شود که بخواهیم بر حسب مقادیر مختلف یک متغیر یا عبارت خاص ، دستورات متفاوتی اجراشود .

شکل کلی استفاده از این ساختار به صورت زیر است :

```
switch   ( نام یک متغیر یا یک عبارت )
{
 case   مقدار ۱ :
 case 1 دستورات مربوط به
 case   مقدار ۲ :
 case 2 دستورات مربوط به
 .
 .
 .
 .

 case   مقدار n :
 case n دستورات مربوط به
 default :

های فوق ، اجرا خواهند شد  caseدستورات پیش فرض ساختار که در صورت عدم برقراری مقادیر تمام
}
```

نحوه عملکرد این ساختار به شرح زیر است :

ابتدا در پرانتز مقابل واژه کلیدی switch ، یک متغیر یا عبارت اعلام می شود . همچنین در هر دستور case ، یک مقدار مرتبط با آن متغیر نیز تعیین می شود . برنامه مقدار متغیر را (که از قبل توسط یک تابع یا دستور مقدار دهی شده است) را با مقدار تعیین شده برای هر case به ترتیب مقایسه کرده و در صورت برابر بودن آنها ، دستورات آن case و case های بعد از آن را اجرا می کند . در واقع دستورات تا زمان رسیدن به یک دستور break اجرا می شوند .

نکته : برای جلوگیری از اجرای case های بعدی همراه با case ای که اجرا می شود ، باید در پایان دستورات هر case از واژه کلیدی break استفاده کرد . برای دریافت توضیحات بیشتر ، به قسمت توضیح دستور break در پایین صفحه بروید .

قسمت default ، دستورات پیش فرض ساختار را تعیین می کند ، که در صورت عدم برقراری مقادیر تمام case های ساختار ، دستورات آن قسمت اجرا خواهد شد .

نکات مهم :

۱. نوع متغیر تعیین شده در قسمت switch و مقدارهای هر یک از case ها باید با هم یکسان و از یک نوع باشند.
۲. به تعداد مورد نیاز می توان از دستور case درساختار switch استفاده کرد.
۳. تعیین قسمت default درساختار switch ، اختیاری است و می تواند تعیین نشود.

٤. مقدار هیچ دو case متفاوتی ، نباید با هم یکسان باشد.

مثل : در مثال زیر دو کادر متن و یک دکمه فرمان قرار داده شده است . کادر متن اول یک متن یا عدد از کاربر دریافت کرده و با کلیک بر روی دکمه فرمان و فراخوانی تابع hello ( ) ، خروجی در کادر متن دوم نمایش داده می شود .

عملکرد تابع hello ( ) به شرح زیر است :

این تابع مقدار کنترل کادر متن اول یعنی txtinput ، را در متغیر matn ذخیره کرده و سپس در یک ساختار switch ،بر حسب مقادیر مختلف خروجی را در کادر متن دوم یعنی txtresult نمایش می دهد .

توجه : در اینجا به دلیل عدم استفاده از دستور break در پایان دستورات هر case با اجرای هر کدام از case های مثال ، دستورات بقیه case ها نیز اجرا می شود . بنابراین همواره خروجی ما در این حالت خروجی قسمت default است .

Example

کد

```
< input type="text" id="txtinput " / >
< input type="button" name="clickme" onclick="hello( )" value="click me !" / >
< input type="text" id="txtresult" / >
< script type="text/javascript" >
 function hello( )
  {
   var matn = txtinput.value ;
   switch (matn)
   {
    case "1":
    txtresult.value = "one" ;
    case "2":
    txtresult.value = "two";
    case "3":
    txtresult.value = "three";
    default:
    txtresult.value = "bigger than 3";
   }
  }
< /script >
```

دستور break :

همانطور که در قسمت بالا نیز اشاره شد ، چنانچه یک دستور case درساختار switch اجرا شود ، برنامه به طور اتوماتیک دستورات تمام case های بعد از آن را نیز اجرا خواهد کرد ، مگر اینکه به یک دستور break برسد . . برای جلوگیری از این مسئله ، باید در پایان دستورات هر case از دستور break استفاده کرد .

در این حالت در صورت وارد شدن برنامه به یک دستور یک case ، **فقط دستورات آن** case اجرا شده و اجرای بقیه case ها متوقف می شود .

شکل کلی استفاده از دستور break به صورت زیر است :

```
switch   ( نام یک متغیر یا یک عبارت )
{
 case   مقدار ۱ :
 case 1 دستورات مربوط به
 break ;
 case   مقدار ۲ :
 case 2 دستورات مربوط به
 break ;
 .
 .
 .
 .

 case   مقدار n :
 case n دستورات مربوط به
 break ;
 default :

 های فوق ، اجرا خواهند شد caseدستورات پیش فرض ساختار که در صورت عدم برقراری مقادیر تمام
}
```

مثل : در مثال زیر ، مثال قسمت قبل را با دستور break بازنویسی کرده ایم . همانطور که در خروجی مشاهده می کنید ، در این حالت به دلیل استفاده از دستور braek در پایان هر case ، **فقط دستورات آن** case **اجراشده و از** اجرای دستورات case های دیگر جلوگیری می شود . برای مشاهده خروجی در کادر متن txtinput متنی را وارد کرده و سپس بر روی دکمه فرمان کلیک کنید .

Example

کد

```
< input type="text" id="txtinput " / >
< input type="button" name="clickme" onclick="hello( )" value="click me !" / >
< input type="text" id="txtresult" / >
< script type="text/javascript" >
 function hello( )
  {
    var matn = txtinput.value ;
    switch (matn)
    {
     case "1":
```

```
      txtresult.value = "one" ;
      break ;
      case "2":
      txtresult.value = "two";
      break ;
      case "3":
      txtresult.value = "three";
      break ;
      default:
      txtresult.value = "bigger than 3";
    }
  }
< /script >
```

توابع در جاوا اسکریپت

تعریف تابع :

تعریف تابع : یک تابع مجموعه ای واحد از یکسری دستورالعمل است که در هر بار فراخوانی ، کل دستورات درون آن یکبار اجرا می شود .
می دانیم که هر برنامه کامپیوتری ، شامل مجموعه ای از دستوالعمل هاست . از توابع برای شکستن و تقسیم کردن کل برنامه به واحدهای کوچکتر و مستقل استفاده می شود .

قابلیت های تابع :

استفاده از توابع در صفحات و اسکریپت ها ، امکانات زیر را به برنامه نویس می دهد :

۱. دستورات یک تابع (حتی در زمانی که اسکریپت آن در درون صفحه قرار دارد) ، تا زمانی که فراخوانی نشود ، اجرا نخواهد شد . از توابع برای تعریف دستور العمل هایي استفاده می شود که می خواهیم اجرای آنها کنترل شده باشد و در مواقع معینی (مثل وقوع یک رویداد یا ...) انجام شود.
۲. یک تابع را می توان از هر نقطه ای در صفحه فراخوانی کرد.
۳. یک تابع می توان یکسری متغیرها را به عنوان پارامتر ورودی دریافت کرده و همچنین یک مقدار را به عنوان خروجی به نقطه ای که از آن فراخوانی شده سات ، باز گردهند.

محل تعریف توابع :

توابع را می توان در هر جای تگ های اسکریپت < script > ، تعریف کرد . برای آشنایي با مکان تعریف تگ اسکریپت در صفحات HTML ، به قسمت آموزش مکان اسکریپت ها در صفحات وب بروید .
نکته مهم : در قسمت مکان قرار گیری اسکریپت ها در صفحات وب اشاره کردیم ، اسکریپت هایي که در درون قسمت < body > تعریف می شوند ، به محض لود شدن صفحه اجرا خواهند شد . اما چنانچه این اسکریپت ها شامل توابع باشند ، دستورات آن توابع تا زمان فراخوانی تابع اجرا نخواهند شد .

شکل کلی تعریف توابع :

برای تعریف یک تابع از واژه کلیدی function به شکل کلی زیر استفاده می شود :

```
function  نام تابع ( )
{
  دستورات تابع
}
```

مثل : در مثال زیر یک تابع به نام welcome ، تعریف شده است . این تابع تا زمانی که فراخوانی نشود اجرا نشده و خروجی ندارد .

Example	

< script type="text/javascript" >
function welcome( )
{
document.write ("welcome to DeveloperStudio")
}
< /script >
	کد

نحوه فراخوانی تابع :

یک تابع را به شرطی که قبل از آن به طور کامل تعریف شده باشد ، می توان از هر جای صفحه فراخوانی کرد . برای فراخوانی یک تابع ، از نام تابع به همراه یک پرانتز باز و بسته در مقابل نام آن به صورت زیر عمل می شود :

نام تابع ( ) ;

مثل :در مثال زیر تابع welcome را که در مثال قبل نیز استفاده کرده بودیم را توسط یک اسکریپت دیگر فراخوانی کرده ایم .

Example	

< html >
< head >
< title > عنوان صفحه < /title >
< /head >
< body >
< script type="text/javascript" >
function welcome( )
{
document.write ("welcome to DeveloperStudio")
}
< /script >
 	
< script type="text/javascript" >
welcome ( ) ;
</script>
	کد

< /body > < /html >	
welcome to DeveloperStudio	خروجی

نکته : یک تابع را می توان توسط رویدادهای یک کنترل HTML مثل یک دکمه فرمان نیز فراخوانی کرد . در این حالت باید مقدار رویداد مورد نظر را در تگ کنترل ، برابر نام تابع در **ظر گرفت** . به مثال زیر دقت کنید .

مثل : در مثال زیر تابع welcome که در مثال قبل توسط یک اسکریپت دیگر فراخوانی شده بود ، در اینجا توسط رویداد کلیک (onclick) دکمه فرمان btnclick فراخوانی می شود . برای فراخوانی تابع بر روی دکمه فرمان مثال کلیک کنید .

Example	

< html >
< head >
< title > عنوان صفحه < /title >
< /head >
< body >
< script type="text/javascript" >
function welcome( )
{
document.write ("welcome to DeveloperStudio")
}
< /script >
 	
< input type="button" id="Button1" onclick="welcome( )" value=" click me ! " / >
< /body >
< /html >
	کد

نحوه تعریف پارامتر برای یک تابع :

تعریف پارامتر :پارامتر ، یک متغیر است که می توان در هنگام فراخوانی یک تابع ، مقدار آن را به تابع ارجاع داد . به پارامتر ، **آرگومان** نیز می گویند .

یک تابع می تواند ، چندین متغیر را به عنوان پارامتر ورودی دریافت کند . پارامترهای یک تابع را باید در هنگام تعریف تابع ، در پرانتز مقابل نام آن تعیین کرد ، که پارامترها را با کاما از هم جدا می کنیم .

در هنگام فراخوانی یک تابع که دارای پارامتر است ، باید در پرانتز مقابل نام آن ، مقادیر متناظر با پارامترهایش را اعلام کرد . این پارامترها باید از لحاظ تعداد و نوع کاملا یکسان با پارامترهای تعریف شده در تابع باشند .

شکل کلی تعریف پارامتر برای یک تابع بع صورت زیر است :

```
funciton   نام تابع   ( Parametr 1 , Parametr 2 , ... )
 {
   دستورات تابع
 }
```

مثال :
```
function multiple ( var num1 , var num2 )
      {
        دستورات تابع
      }
```

در هنگام فراخوانی یک تابع نیز باید به شکل زیر پارامترهای آن را مقدار دهی کرد . توجه شود که نوع و تعداد متغیرها باید کاملا یکسان با پارامترهای تعریف شده برای تابع باشد ، در غیر این صورت error رخ داده و تابع اجرا نمی شود .

```
function   نام تابع g ( Parametr 1 مقدار , Parametr 2 مقدار , ... )
 {
   دستورات تابع
 }
```

مثل : در مثال زیر تابع multiple را که در بالا تعریف کرده بودیم با تعیین پارامترهای لازم ، فراخوانی کرده ایم :

```
multiple ( 2 , 10 ) ;
```

نکته مهم : پارامتر های یک تابع ، به عنوان متغیرهای محلی برای آن تابع قابل استفاده هستند .

مثل : در مثال زیر تابع multiple با دو پارامتر num1 و num2 تعریف شده است ، که این تابع این دو عدد را در هم ضرب کرده و به عنوان خروجی روی صفحه نمایش می دهد . این تابع در یک اسکریپت دیگر توسط تابعCall_Function فراخوانی می شود . این تابع دو عدد را از دو کادر متن مثال دریافت کرده و به ترتیب آنها را در متغیرهای a , b ذخیره می کند . سپس آن را در هنگام فراخوانی به تابع multiple پاس می دهد . توجه شود که از متغیر های دیگر نیز می توان برای فراخوانی استفاده کرد . برای مشاهده خروجی ۲ عدد را در کادر متن ها وارد کرده و بر روی دکمه فرمان multiple کلیکگ کنید .

Example	
< script type="text/javascript" > function multiple (num1 , num2)	کد

```
  {
    document.write ( num1 * num2 ) ;
  }
< /script >

< script type="text/javascript" >
 function Call_Function ( )
  {
     var a = Text1.value ;
     var b = Text2.value ;
     multiple ( a , b ) ;    فراخوانی تابع با پارامترهای لازم و چاپ خروجی
  }
< /script >
< input type="button" id="Button2" onclick="Call_Function( )" value=" click me ! " />
< input type="text" id="Text1" />
< input type="text" id="Text2" />
```

نحوه تعریف مقدار بازگشتی برای یک تابع :

یک تابع می تواند پس از انجام دستورات در نظر گرفته شده برای آن ، مقداری را به عنوان خروجی به نقطه ای که از آن فراخوانی شده است ، باز گرداند .
برای تعیین مقدار بازگشتی یک تابع از دستور return استفاده کرده ، که مقدار خروجی را درپرانتز مقابل آن دستور به شکل زیر تعریف می کنیم .

```
return  ( مقدار بازگشتی ) ;
```

نکته : مقدار بازگشتی ، می تواند یک رشته یا عدد ، یک متغیر و یا یک عبارت محاسباتی باشد .

مثل : تابع multiple را در مثل قبل با دستور return بازنویسی کرده ایم . در این حالت تابع به جای استفاده از دستور document.write با دستور return ، مقدار را برای چاپ خروجی به تابع Call_Function بر می گرداند :

Example

کد

```
< script type="text/javascript" >
 function multiple ( num1 , num2 )
  {
    return ( num1 * num2 ) ;
```

```
  }
< /script >

< script type="text/javascript" >
 function Call_Function ( )
  {
     var a = Text1.value ;
     var b = Text2.value ;
     document.write (multiple ( a , b )) ;    فراخوانی تابع با پارامترهای لازم
  }
< /script >
< input type="button" id="Button2" onclick="Call_Function( )" value=" click me ! " />
< input type="text" id="Text1" />
< input type="text" id="Text2" />
```

منوهای Pop-Up در جاوا اسکریپت

توسط کادر های Pop-Up در جاوا اسکریپت ، می توان به کاربر پیغام اخطار داد ، از آن تایید گرفت و یا ورودی دریافت کرد . این کادر ها در مواقع مورد نظر ظاهر شده و عملیات تعیین شده برای آنها انجام خواهند داد .
به طور کلی ۳ نوع کادر Pop-Up در جاوا اسکریپت داریم :

۱. کادر پیام یا هشدار(alert Box)
۲. کادر دریافت تایید(confirm Box)
۳. کادر دریافت ورودی(prompt Box)

: در ادامه به معرفی و توضیح هر یک از کادرهای فوق می پردازیم

۱ . کادر پیام یا هشدار(alert Box)

از کادر alert Box :

از کادر alert Box ، برای اعلام یک پیام یا هشدار به کاربر استفاده می شود . این پیام که از قبل باید تعیین شده باشد ،در یک کادر ظاهر شده و تا زمانی که کاربر دکمه Ok را انتخاب نکند ، از بین نمی رود .
شکل کلی تعریف یک کادر منتی به صورت زیر است :

```
alert ( " متن پیام یا هشدار " ) ;
```

مثل : در مثال زیر یک کادر alert Box در تابع Alert_Box تعریف شده است . این تابع توسط دکمه فرمان Alert ! ، فراخوانی شده و کادر پیام خود را ظاهر می کند .

Example	

کد	
	< script type="text/javascript" >
	 function Alert_Box ( )
	 {
	 alert (" Welcome To DeveloperStudio ") ;
	 }
	</script>
	
	<input type="button" id="btnAlert" value=" Alert ! " onclick="Alert_Box( )" />

مثل ۲ : در مثال زیر یک کادر متن و یک دکمه فرمان قرار داده شده است . کاربر باید یک عدد را در کادر متن وارد کرده و بر روی دکمه فرمان کلیک کند . چنانچه عدد ورودی از ۱۰ کمتر باشد ، برنامه عبارت Welcome to DeveloperStdio را در خروجی چاپ کرده و در غیر این صورت یک کادر پیام با متن Number too big را به کاربر نشان می کند .

Example

کد

```
< script type="text/javascript" >
 function Alert_Box2 ( )
  {
    var Num = txtNum.value ;
    if ( Num < 10 )
      document.write ( "Welcome to DeveloperStudio " ) ;
    else
      alert ( " Number too big " ) ;
  }
< /script >

< input type="text" id="Text1" / >
< input type="button" id="btnAlert2" value=" Alert ! " onclick="Alert_Box2()" />
```

نکته : می توان در متن پیام کادرهای جاوا اسکریپت ، به تعداد مورد نیاز خط جدید ایجاد کرد . برای این منظور ، در متن پیام از کاراکتر ' n\ ' به شرحی که در مثال زیر آمده است ، استفاده می شود :

Example

کد

```
< script type="text/javascript" >
 function Alert_Box3 ( )
  {
    alert ( "Hello . Dear User " + '\n' + "Welcome to DeveloperStudio ") ;
}
< /script >

< input type="button" id="btnAlert3" value=" Alert ! with Line Brakes "
onclick="Alert_Box3()" />
```

خروجی

۲ . کادر دریافت تایید: (confirm Box)

از کادر دریافت تایید ، برای اعلام یک پیام به کاربر و دریافت نظر آن مبنی بر قبول یا عدم قبول پیام مورد نظر استفاده می شود .

این کادر دارای ۲ دکمه فرمان OK و Cancel است ، که در صورت انتخاب گزینه OK ، کادر مقدار مثبت (True) و در صورت انتخاب گزینه Cancel ، کادر مقدار منفی (False) را به صفحه بر می گرداند .

شکل کلی تعریف یک کادر تایید به صورت زیر است :

```
confirm ( " متن پیام " ) ;
```

نکته مهم : از مقدار برگشتی یک کادر تایید می توان در برنامه نویسی استفاده کرد . برای این منظور باید مقدار بازگشتی را در یک متغیر به شکل زیر ذخیره کرده و سپس از آن متغیر استفاده کرد . به مثال دقت کنید :

```
نام متغیر  =  confirm ( "متن پیام" ) ;
```

مثل : در مثال زیر یک تابع با یک کادر تایید ، قرار داده شده است . این تابع با دکمه فرمان btnChange ، فراخوانی می شود و کادر تایید خود را نشان می دهد . کادر از کاربر درباره رفتن به صفحه اصلی سایت می پرسد ، که در صورت تایید و فشردن دکمه OK مروگر به صفحه اصلی رفته و در صورت زدن دکمه Cancel یک پیام در خروجی چاپ می کند :

Example

کد

```
< script type="text/javascript" >
 function Change_Page( )
  {
   var x = confirm ( "Do you want go to home page ? " ) ;
   if ( x == true )
    document.URL = "../Default.aspx" ;
   else
    document.write ( " You pressed Cancel ! " ) ;
  }
< /script >

< input type="button" id="btnChange" value=" go to Home Page ?
" onclick="Change_Page( )" />
```

۳ . کادر دریافت ورودی(prompt Box)

از کادر دریافت ورودی ، برای اعلام یک پیام به کاربر و دریافت یک ورودی از وی استفاده می شود . در این حالت یک کادر حاوی پیام مورد نظر ، یک کنترل متنی برای ورود مقدار ، دکمه OK برای تایید و ارسال مقدار ورودی به صفحه و دکمه Cancel برای لغو عملیات کادر ، بر روی صفحه نمایش داده می شود .
همچنین می توان یک مقدار پیش فرض نیز در کادر تعیین کرد ، که همواره به صورت پیش فرض در کنترل متنی کادر نمایش داده خواهد شد . تعیین مقدار پیش فرض اختیاری است .
شکل کلی تعریف یک کادر دریافت ورودی سه صورت زیر است :

```
prompt ( " متن پیام کادر " , " مقدار پیش فرض " ) ;
```

مثل : در مثال زیر یک کادر دریافت ورودی ، برای دریافت نام کاربر در تابع Hello_User تعریف شده است . این تابع با کلیک بر روی دکمه فرمان Enter Name فراخوانی شده و سپس با دریافت نام کاربر یک پیام خوش آمد به کاربر در خروجی اعلام می کند . برای مشاهده خروجی بروی دکمه فرمان Enter Name کلیک کرده و سپس نام خود را در کادر وارد کنید :

Example

کد

```
< script type="text/javascript" >
 function Hello_User( )
  {
    var name = prompt ( "enter your name") ;
    if ( name != null )
     document.write ( "hello dear " + name + " " + "Welcome to DeveloperStudio" ) ;
  }
< /script >

< input type="button" id="btnHello" onclick="Hello_User()" value="Enter Name" />
```

مثل : در مثاب زیر یک کادر دریافت تایید در تابع Change_page تعریف شده است . این کادر یک ورودی ، که نام یکی از بخش های آموزشی سایت DeveloperStudio است ، را از کاربر دریافت کرده و سپس آدرس مرورگر را به آدرس مربوط با آن نام تغییر می دهد . در این کادر مقدار پیش فرض Home Page در نظر گرفته شده است . برای مشاهده خروجی بروی دکمه فرمان کلیک کرده و سپس نام مقصد را انتخاب کنید . در صورت وارد کردن اسم نادرست ، یک کادر هشدار ظاهر خواهد شد :

Example

کد

```
< script type="text/javascript" >
 function Change_Page( )
  {
    var page = prompt ( "Where do you want to go ? " , "Home Page" ) ;
```

```
    switch ( page )
    {
       case "Home Page" :
         document.URL = "../Default.aspx" ;
         break ;
       case "Html" :
         document.URL = "../HTML/introducehtml.aspx" ;
         break ;
       case "CSS" :
         document.URL = "../CSS/CSSIntroduce.aspx" ;
         break ;
       case "Java Script" :
         document.URL = "../JavaScript/Java_Script_Introduce.aspx" ;
         break ;
       case "SQL" :
         document.URL = "../SQL/SQLIntroduce.aspx" ;
         break ;
       default :
         alert ( "Incorrect Name" ) ;
    }
}
< /script>

<input type="button" id="Button2" onclick="Change_Page()" value="Enter Page" />
```

آرایه در جاوا اسکریپت

مفهوم آرایه : آرایه مجموعه ای از متغیرهایی از یک نوع داده ای با نام یکسان است ، که هر کدام از اعضای آن توسط یک **شمارنده (اندیس)** ، از یکدیگر متمایز می شوند .
شمارنده هر عضو آرایه در یک براکت در مقابل نام آن تعیین شده ، که برای مقدار دهی و دسترسی به هر عضو آرایه از اندیس آن استفاده می شود .
برای تعریف یک آرایه ، از واژه کلیدی new Array به شکل کلی زیر استفاده می شود :

var نام آرایه = new Array ( ) ;
مثل : var Cars = new Array ( ) ;

نکات مهم :

نکته ۱ : می توان تعداد اعضای یک آرایه را در زمان تعریف ، در پرانتز جلوی واژه کلیدی new Array ، تعیین کرد :
مثل : آرایه زیر ٤ عضو دارد :

var Cars = new Array (4) ;

نکته ۲ : شماره گذاری اندیس اعضای یک آرایه از صفر شروع شده و برای هر عضو شمارنده یک واحد افزایش می یابد .
مثل : آرایه ای که در مثال قبل ایجاد کردیم ، ٤ عشو زیر را داراست :

Cars [0] , Cars [1] , Cars [2] , Cars [3]

نکته ۳ : برای مقدار دهی یا دسترسی به هر عضو آرایه ، از نام آرایه به همراه شمارنده یا اندیس عضو مورد نظر در براکت مقابل نام آن ، به شکل کلی زیر استفاده می شود :

نام آرایه [اندیس عضو مورد نظر] = مقدار مورد ظر ;
مثل : Cars [0] = "Ford" ;
Cars [1] = "Nissan" ;

مثل : در مثال زیر یک آرایه در یک اسکریپت تعریف شده و اعضای آن مقدار دهی شده اند . در پایان نیز عضو **اول و دوم** آرایه در خروجی بر روی صفحه چاپ شده اند :

Example	
< script type="text/javascript" >	کد

```
var Cars = new Array ( 3 ) ;
Cars [ 0 ] = "Ford" ;
Cars [ 1 ] = "Nissan" ;
Cars [ 2 ] = "Mazda" ;
document.write ( Cars [ 0 ] ) ;
document.write ( Cars [ 1 ] ) ;
< /script >
```

خروجی

```
Ford
Nissan
```

روش های مقدار دهی کلی اعضای یک آرایه :

تمام یا بخش هایي از اعصای یک آرایه را می توان در هنگام تعریف و یا بعد از آن مقدار دهی کرد . به طور کلی ۲ روش برای مقدار دهی اعضای یک آرایه وجود دارد :

روش اول : در روش اول ، هر یک از اعضای آرایه را به صورت تکی مقدار دهی می کنیم . در مثال زیر یک ابتدا آرایه ٤ عضوی تعریف شده و سپس مقدار دهی شده است :

Example

کد

```
< script type="text/javascript" >
var Cars = new Array ( 4 ) ;
Cars [ 0 ] = "Ford" ;
Cars [ 1 ] = "Nissan" ;
Cars [ 2 ] = "Mazda" ;
Cars [ 3 ] = "Volvo" ;
< /script >
```

روش دوم : در روش دوم ، مقادیر مورد نظر برای تمام یا تعدادی از اعضای آرایه را در پرانتز جلوی واژه کلیدی new Array تعیین کرده و هر مقدار را با کاما از هم جدا می کنیم . در این حالت تعداد اعضای آرایه به طور اتوماتیک بر حسب تعداد مقادیر ورودی تعیین می شود . در مثال زیر ، آرایه تعریف ٤ عضو خواهد شد . اعضای این آرایه در مرحله تعریف آرایه تعیین شده اند :

Example

کد

```
< script type="text/javascript" >
var Cars = new Array ( "Ford" , "Nissan" , "Mazda" , "Volvo" ) ;
```

< /script >	

حلقه ها در جاوا اسکریپت :

مفهوم حلقه : از ساختار دستوری حلقه ها در جاوا اسکریپت ، برای اجرای مجوعه ای از دستور العمل ها به تعداد دفعات مورد نیاز یا تا زمانی که یک شرط خاص درست باشد ، استفاده می شود .
در حلقه ، هنگامی که مجموعه دستورات حلقه به طور کامل اجرا می شود ، برنامه دوباره به ابتدای مجموعه دستورات حلقه رفته و در صورت برقرار بودن شرط حلقه ، یکبار دیگر دستورات آن به طور کامل اجرا خواهد کرد .
به طور کلی ۲ نوع حلقه در جاوا اسکریپت داریم :

1. حلقه for : در این نوع حلقه ، مجموعه دستورالعمل ها به تعداد معلوم و مورد نیاز ، تکرار خواهد شد .
2. حلقه while : در این حالت ، دستورالعمل های حلقه تا زمانی که شرط تعیین شده برای آن درست باشد ، مجددا اجرا خواهد شد .

در ادامه به بررسی انواع حلقه های مورد استفاده در جاوا اسکریپت می پردازیم . در لیست زیر حلقه های معرفی شده در این قسمت قرار دارند . برای دریافت اعلاعات درباره هر کدام ، بر روی آن کلیک کنید :

do ... while
for
for ... in
while

۱) حلقه for :

از حلقه for ، زمانی استفاده می شود که می خواهیم مجموعه دستورالعمل های حلقه به تعداد دفعات معینی انجام شود .
این حلقه در هنگام تعریف ۳ پارامتر اصلی دارد :

۱. مقدار اولیه متغیر : به وسیله این مقدار ، مقدار اولیه برای شروع شمارنده حلقه تعیین می شود.
۲. عبارت کنترلی : در این قسمت یک عبارت کنترلی مرتبط با شمارنده حلقه تعیین می شود ، که در هر بار اجرای مجدد حلقه ، شرط عبارت کنترل شده و در صورت برقرار بودن شرط ، دستورات حلقه اجرا می شود.
۳. گام افزایش یا کاهش : در این قسمت ، میزانی که متغیر شمارنده حلقه ، در هر بار اجرای دستورات آن افزایش یا کاهش می یابد را تعیین می کنیم.

نحوه عملکرد این حلقه به صورت زیر است :
در این حلقه ، از یک متغیر به عنوان شمارنده یا کنترل کننده حلقه استفاده می شود . این متغیر در ابتدای اجرای حلقه ، مقدار دهی اولیه شده و اجرای مجدد حلقه با یک عبارت شرطی کنترل شده که در صورت درست بودن شرط ، دستورات حلقه یکبار

اجرا می شود و با هر بار اجرای حلقه متغیر حلقه به اندازه گام تعیین شده ، افزایش یا کاهش می یابد .
شکل کلی تعریف یک حلقه for به صورت زیر است :

```
for  ( گام افزایش یا کاهش  ; تعیین عبارت کنترلی  ; تعیین مقدار اولیه متغیر )
  {
   دستورات بدنه حلقه
  }
```

نکته : از ۳ پارامتر اصلی حلقه for ، فقط تعیین عبارت کنترلی در هنگام تعریف حلقه اجباری بوده و می توان متغیر شمارنده حلقه را قبل از تعریف حلقه مقدار دهی کرد و همچنین گام افزایش یا کاهش را در بدنه دستورات تابع قرار داد . در این صورت می توان جای موارد فوق را در تعریف حلقه خالی گذاشت .

مثل : در مثال زیر یک حلقه ساده در اسکریپت زیر ایجاد شده است . شمارنده این حلقه که متغیری به نام n است با مقدار اولیه ۱ مقدار دهی شده و شرط حلقه کوچکتر یا مساوی بودن شمارنده حلقه از ۵ تعیین شده است . گام افزایش این حلقه نیز ۱+ در نظر گرفته شده است . این حلقه در هر بار اجرا مقدار متغیر n را بر روی صفحه چاپ می کند . به خروجی آن دقت کنید :

Example	

< script type="text/javascript" >
 var n ;
 for (n = 1 ; n <= 5 ; n++)
 {
 document.write ("Line number is " + n + "
") ;
 }
< /script >
	کد

Line number is 1 Line number is 2 Line number is 3 Line number is 4 Line number is 5	خروجی

مثل ۲ : همانطور که گفتیم می توان قسمت پارامترهای مقدار اولیه و گام افزایش یا کاهش را در یک حلقه for خالی گذاشته و مقدار اولیه را قبل از تعریف حلقه و گام حلقه را در درون بلاک کد حلقه تعیین کرد . **مثل‌شماره ۱** را به این صورت نیز می تول نوشت :

Example	

< script type="text/javascript" >
 var n = 1;
 for (; n <= 5 ;)
 {
 document.write ("Line number is " + n + "
") ;
 n++ ;
 }
< /script >
	کد

Line number is 1 Line number is 2 Line number is 3 Line number is 4 Line number is 5	خروجی

نکته و مثال ۳ : گام یک حلقه می تواند منفی یا کاهشی نیز باشد . در مثال زیر شمارنده حلقه با هر بار اجرای حلقه یک واحد کاهش می یابد :

Example	

< script type="text/javascript" >
 var n ;
 for (n = 5 ; n >= 1 ; n--)
 {
 document.write ("Line number is " + n + "
") ;
 }
< /script >
	کد

Line number is 5 Line number is 4 Line number is 3 Line number is 2	خروجی

Line number is 1	

یک برنامه کاربردی :

در این قسمت یک برنامه ساده را با جاوا اسکریپت طراحی کرده ایم . این برنامه از طریق ۲ کادر متن ، دو عدد را به عنوان ورودی دریافت کرده و عدد اول را به توان عدد دوم می رساند .

توضیح : در ابتدا یک متغیر به نام sum برای نگهداری جواب را با مقدار اولیه ۱ تعریف می کنیم . سپس مقدار کادر اول در متغیر n و مقدار کادر دوم را در متغیر i ذخیره می کنیم . از عدد دوم به عنوان شمارنده حلقه استفاده شده که با هر بار اجرای حلقه ، عدد اول یکبار در خود ضرب شده و یک واحد از شمارنده حلقه نیز کم می شود ، تا به ۱ برسد . در این حالت اجرای حلقه متوقف شده و نتیجه خروجی بر روی صفحه چاپ می شود :

Example	

< script type="text/javascript" >
function multiple ( )
{
var sum = 1 ;
var n = Num1.value ;
for (var i = Num2.value ; i > 0 ; i--)
{
sum = sum * n ;
}
 	
document.write (sum) ;
}
< /script >
 	
< input type="text" id="Num1" />
< input type="text" id="Num2" />
< input type="button" id="Btnclick" value=" Click for Multipe" onclick="multiple( )" / >
	کد

Number 1 : Number 2 :	خروجی

حلقه while در جاوا اسکریپت :

از حلقه while در جاوا اسکریپت ، برای اجرای دستورالعمل های مورد نظر تا زمانی که شرط یا شروط تعیین شده برای حلقه درست باشند ، استفاده می شود .
در این حالت ، ابتدا شرط حلقه در مقابل کلمه کلیدی while تعریف می شود . در هر بار اجرای حلقه ، برنامه شرط یا شروط حلقه را چک کرده و در صورت بر قرار بودن آن ، دستورالعمل های حلقه را یکبار اجرا کرده و مجددا به ابتدای حلقه باز می گردد .
در حلقه while نیز می توان از یک متغیر برای کنترل اجرای حلقه استفاده کرد . در صورت استفاده از یک متغیر شمارنده ، باید گام افزایش یا کاهش متغیر در بدنه دستورات حلقه تعریف شود و در صورت عدم استفاده از یک متغیر کنترلی ، باید شرط حلقه در ادامه به نحوی نقض شود ، وگرنه حلقه به صورت بی نهایت ادامه می یابد .
شکل کلی تعریف یک حلقه while به صورت زیر است :

```
while   ( شرط یا شروط حلقه )
  {
    دستوراالعمل های مورد نظر حلقه
  }
```

مثل : مثال اول قسمت آموزش حلقه for را در اینجا با حلقه while باز نویسی کرده ایم . در این حلقه از متغیر n به عنوان متغیر شمارنده و کنترل کننده حلقه استفاده شده است . اجرای حلقه تا زمانی که شرط آن درست است ، ادامه دارد :

Example

کد

```
< script type="text/javascript" >
  var n = 1 ;          تعریف و مقدار دهی متغیر کنترلی حلقه
  while ( n <= 5 )
   {
     document.write ("Line number is " + n + "<br />") ;
     n++ ;          گام افزایشی متغیر کنترلی حلقه
   }
< /script >
```

خروجی

```
Line number is 1
Line number is 2
Line number is 3
Line number is 4
Line number is 5
```

مثل ۲ : در این مثال برنامه کاربردی که یک عدد را به توان عدد دیگری می رساند را که با حلقه for نوشته بودیم ، را با حلقه while باز نویسی کرده ایم . به تفاوت های این دو ساختار دقت کنید :

Example

کد

```
< script type="text/javascript" >
 function multiple ( )
  {
    var sum = 1 ;
    var n = Num1.value ;
    var i = Num2.value ;
    while ( i > 0 )
     {
       sum = sum * n ;
       i-- ;
     }

    document.write ( sum ) ;
  }
< /script >

< input type="text" id="Num1" />
< input type="text" id="Num2" />
< input type="button" id="Btnclick" value=" Click for Multipe" onclick="multiple( )" / >
```

خروجی

Number 1 : Number 2 :

حلقه do ... while :

ساختار حلقه do ... while ، دقیقا همانند حلقه while است ، با این تفاوت که شرط حلقه do ... while در انتهای حلقه تعریف و کنترل می شود . به عبارت دیگر در این حلقه ابتدا یکبار دستورات حلقه اجرا شده و در آخر شرط حلقه برای اجرای مجدد ، کنترل می شود که در صورت درست بودن یکبار دیگر دستورات آن خواهد شد .

مزیت این حلقه نسبت به حلقه while این است که ، در حلقه while در صورت عدم برقراری شرط حلقه دستورات آن هیچگاه اجرا نخواهد شد . اما در حلقه do ... while ، حتی در صورت غلط بودن و عدم برقراری شرط حلقه ، دستورات آن حداقل یکبار اجرا خواهد شد .

شکل کلی تعریف یک حلقه do ... while به صورت زیر است :

```
do
 {
  دستورات حلقه
 }
while  ( شرط یا شروط حلقه )
```

مثل : مثال اول قسمت آموزش حلقه while را در اینجا با حلقه do ... while باز نویسی کرده ایم . در این حلقه از متغیر n به عنوان متغیر شمارنده و کنترل کننده حلقه استفاده شده است . اجرای حلقه تا زمانی که شرط آن درست است ، ادامه دارد :

Example

کد

```
< script type="text/javascript" >
 var n = 1 ;          تعریف و مقدار دهی متغیر کنترلی حلقه
 do
  {
    document.write ("Line number is " + n + "<br />") ;
    n++ ;          گام افزایشی متغیر کنترلی حلقه
  }
 while ( n <= 5 )
< /script >
```

خروجی

```
Line number is 1
Line number is 2
Line number is 3
Line number is 4
Line number is 5
```

مثل : در مثال زیر یک حلقه do ... while تعریف شده که شرط اجرای دستورات آن کوچکتر بودن متغیر c از ۵ است . اما قبل از حلقه متغیر c با مقدار ۸ مقدار دهی شده است . می بینیم که با وجود اشتباه بودن و عدم برقراری شرط حلقه دستورات آن حداقل یکبار اجرا شده و خروجی تولید کرده است ، ولی سری دوم اجرای حلقه به دلیل عدم برقراری شرط آن اجرا نشده است :

Example

کد

```
< script type="text/javascript" >
 var c = 8 ;          تعریف و مقدار دهی متغیر کنترلی حلقه
 do
  {
    document.write ("Line number is " + c + "<br />") ;
```

c++ ;
}
while (c <= 5)
< /script >	

Line number is 8	خروجی

حلقه for...in در جاوا اسکریپت :

از حلقه for ... in در جاوا اسکریپت ، برای حرکت در درون اعضای یک آرایه یا مجموعه خواص یک شی استفاده می شود .
به ازای خواندن هر یک از اعضا آرایه یا یکی از خواص شی مورد نظر ، یکبار دستورات درون حلقه اجرا خواهد شد .
تعداد دفعات تکرار دستورات حلقه ، برابر با تعداد اعضای آرایه و یا تعداد خواص شی مورد نظر است . در این حلقه معمولا از یک متغیر به عنوان شمارنده یا اندیس آرایه استفاده می شود .
شکل کلی تعریف یک حلقه for...in به صورت زیر است :

```
for  ( متغیر  in  نام یک آرایه / مجموعه خواص یک شی )
 {
   دستورات بدنه حلقه
 }
```

مثل : در مثال زیر ابتدا یک آرایه به نام Lesson برای نگهداری نام دروس کامپیوتر با 0 عضو ایجاد و مقدار دهی شده است . وظیفه حلقه for...in حرکت در درون اعضای آرایه Lesson و چاپ نام تک تک آنها به عنوان خروجی است . به کد مثال دقت کنید :

Example

کد

```
< script type="text/javascript" >
 var n = 0 ;
 var i = 1 ;
 var Lessons = new Array( 5 ) ;
 Lessons[0] = "HTML" ;
 Lessons[1] = "CSS" ;
 Lessons[2] = "Visual Basic" ;
 Lessons[3] = "Java Script" ;
 Lessons[4] = "ASP.NET" ;
 for ( n  in  Lessons )
  {
   document.write ( "Lesson " + i + " = " + Lessons [n] + "<br / >") ;
   i++ ;
  }
< /script >
```

خروجی

```
Lesson 1 = HTML
```

Lesson 2 = CSS Lesson 3 = Visual Basic Lesson 4 = Java Script Lesson 5 = ASP.NET	

for...in و حلقه continue یک مثال مرتبط با دستور :

مثل : فرض کنید که در مثال بالا فقط می خواهیم نام دروسی از آرایه Lesson چاپ شود ، که شماره اندیس آنها در آرایه **زوج باشد** . بنابراین در بدنه دستورات حلقه یک دستور continue با این شرط که باقی مانده حاصل از تقسیم شماره اندیس آرایه بر ۲ **برابر صفر** نباشد را به قبل از دستور چاپ حلقه اضافه می کنیم . در این صورت نام اعضایی از آرایه که شماره اندیس آنها فرد است چاپ نمی شود و حلقه به اندیس بعدی می رود :

Example

کد

```
< script type="text/javascript" >
  var n = 0 ;
  var i = 1 ;
  var Lessons = new Array( 5 ) ;
  Lessons[0] = "HTML" ;
  Lessons[1] = "CSS" ;
  Lessons[2] = "Visual Basic" ;
  Lessons[3] = "Java Script" ;
  Lessons[4] = "ASP.NET" ;
  for ( n   in   Lessons )
   {
     if ( n % 2 != 0 ) continue ;
     document.write ( "Lesson " + i + " = " + Lessons [n] + "<br / >") ;
     i++ ;
   }
< /script >
```

خروجی

```
Lesson 1 = HTML
Lesson 2 = Visual Basic
Lesson 3 = ASP.NET
```

دستورات break و continue

دستور break :

از دستور break برای خروج کامل از ادامه اجرای دستورات یک حلقه در صورت بر قرار بودن شرط تعیین شده برای آن استفاده می شود .
دستور break را باید در بدنه دستورات یک حلقه تعریف کرد . در هر بار اجرای حلقه ، برنامه با رسیدن به شرط دستور break ، آنرا چک کرده و در صورت برقراری شرط از ادامه اجرای دستورات حلقه به طور کامل جلوگیری کرده و به طور کامل از حلقه خارج می شود .

شکل کلی تعریف یک دستور break به صورت زیر است :
نکته : عملکرد و استفاده از دستور break در تمام حلقه ها یکسان است . در مثال زیر فرض می کنیم ، حلقه ما while است :

```
while ( شرط حلقه )
 {
  دستورات حلقه
  if ( شرط حلقه )   break ;
  ادامه دستورات حلقه
 }
```

مثل : مثال چاپ شماره خطوط را که در حلقه های قبل به کار برده بودیم را در این قسمت ، با دستور break باز نویسی کرده ایم . در این حلقه شرط دستور break ، برابر شدن متغیر شمارنده حلقه یعنی n با مقدار ۳ است ، که در هنگامی که n برابر ۳ می شود ، برنامه از اجرای ادامه دستورات حلقه جلوگیری کرده و از حلقه خارج می شود . بنابرین شماره خطوط تاشماره ۳ چاپ‌شده و ٤ و ٥ چاپ نمی شود :

Example	
نکته : توجه شود که دستورات قبل از دستور break در بدنه حلقه ، به طور کامل اجرا می شوند و تاثیر دستور break بر دستورات بعد از خود می باشد .	
کد	

```
< script type="text/javascript" >
 var n ;
 for ( n = 1 ; n <= 5 ; n++ )
  {
    document.write ("Line number is " + n + "<br />") ;
    if ( n == 3 ) break ;
```

} < /script >	
Line number is 1 Line number is 2 Line number is 3	خروجی

دستور continue :

از دستور continue ، برای خارج شدن از ادامه اجرای یکبار دستورات حلقه و پرش به گام بعدی حلقه استفاده می شود .
دستور continue را باید در بدنه دستورات حلقه تعریف کرد . در هر بار اجرای حلقه ، برنامه با رسیدن به شرط تعیین شده برای دستور continue آن دستور را بررسی کرده و در صورت درست بودن ، از ادامه اجرای دستورات حلقه در آن مرحله جلوگیری کرده و مجددا به ابتدای حلقه باز می گردد . در این حالت ، گام حرکت حلقه یکبار اجرا خواهد شد .

شکل کلی تعریف یک دستور continue ، به صورت زیر است .
نکته : عملکرد و استفاده از دستور continue در تمام حلقه ها یکسان است . در این مثال ، ما فرض کرده ایم که حلقه while است :

```
while ( شرط حلقه )
 {
  دستورات حلقه
  if ( شرط حلقه )  continue ;
  ادامه دستورات حلقه
 }
```

مثال : مثال چاپ شماره خطوط را که در حلقه های قبل به کار برده بودیم را در این قسمت ، با دستور continue باز نویسی کرده ایم . در این حلقه شرط دستور continue ، برابر شدن متغیر شمارنده حلقه یعنی n با مقدار ۳ است ، که در هنگامی که n برابر ۳ می شود ، برنامه از اجرای ادامه دستورات حلقه در مرحله ای که n = ۳ است جلوگیری کرده و به گام بعدی حلقه یعنی n = ٤ پرش می کند . بنابرین شماره خطوط تاشماره ۲ چاپ‌شده وشماره ۳ چاپ نشده و به ادامه اجرای حلقه در خط ٤ پرش می شود .

Example	
نکته : توجه شود که دستورات قبل از دستور continue در بدنه حلقه ، به طور کامل اجرا می شوند و تاثیر دستور continue بر دستورات بعد از خود می باشد .	
< script type="text/javascript" > var n ;	کد

```
for ( n = 1 ; n <= 5 ; n++ )
  {
    if ( n == 3 ) continue ;
    document.write ("Line number is " + n + "<br />") ;
  }
< /script >
```

خروجی

```
Line number is 1
Line number is 2
Line number is 4
Line number is 5
```

ساختار دستوری try ... catch :

در برنامه نویسی و طراحی صفحات وب ، گاهی اوقات ممکن است خطاهایی رخ دهد . این خطاها می تواند دلایل مختلفی داشته باشد ، از قبیل :

- اشتباه تایپی در متن دستورات صفحه از سوی طراح صفحه.
- واردنمودن اطلاعات اشتباه یا نادرست از سوی کاربر.
- انواع خطاهای ممکن زمان اجرای مرورگر.

اگر کاربر که در حال مشاهده یک صفحه است ، در هنگام کار با خطا مواجه شود و عملا راهی برای رفع آن اشکال برایش وجود نداشته باشد ، مطمئنا از ادامه مشاهده و کار با صفحه صرف نظر خواهد کرد . این وظیفه برنامه نویس است که با برنامه ریزی صحیح ، خطاهای احتمالی را پیدا کرده و راه حل های مناسب را برای آنها طراحی کند .

از ساختار کنترلی try ... catch در جاوا اسکریپت ، برای پیدا کردن خطاها و error های احتمالی و تعیین اقدامات اصلاحی در صورت بروز خطا ، استفاده می شود .
این ساختار از ۲ بخش try و catch تشکیل شده است . کد اصلی برنامه که می خواهیم اجرا شود و احتمال می دهیم دارای خطا باشد ، را در قسمت try قرار داده و اقدامات اصلاحی را که می خواهیم در صورت بروز خطا انجام شود ، را در قسمت catch می گذاریم .
برنامه در هنگام رسیدن به ساختار ، دستورات قسمت try را انجام می دهد و در صورت مواجه با خطا در دستورات ، بخش catch را اجرا خواهد کرد .
نکته مهم : در صورت عدم برخورد با خطا ، دستورات قسمت catch به هیچ وجه اجرا نخواهد شد .
نکته مهم ۲ : در قسمت catch ، می توان در پرانتز مقابل واژه کلیدی catch ، یک عبارت یا متغیر تعریف کرد ، که این متغیر خصوصیات error به وجود آمده را در خود نگهداری کرد . یکی از این خواص ، شرح یا description خطا است ، که می توان به شکلی که در مثال ۲ نشان داده شده است ، به شرح خطا دسترسی داشته و آنرا به کاربر اطلاع داد .
شکل کلی تعریف یک ساختار try ... catch به صورت زیر است :

```
try
 {
  دستورات مورد نظر برای اجرا که احتمال خطا دارد
 }
catch ( نام یک متغیر )
 {
  اقدامات اصلاحی مورد نظر در صورت وقوع خطا
 }
```

مثل ۱ : در مثال زیر ، یک تابع به نام show_error **قرار داده‌شده است** . این تابع توسط دکمه فرمان click me فراخوانی شده و قصد دارد تا یک پیغام خوش آمد به کاربر اعلام کند . اما در متن دستور یک اشتباه تایپی وجود دارد و آن اینکه به جای **عبارت** documnet.write ، نوشته شده است document.wriet ، به همین دلیل پیغام خروجی چاپ نشده و در صفحه error رخ می دهد . اگر دقت کنید در نوار پایین مرورگر (Status Bar) **علامت** Error on page قرار گرفته است . برای مشاهده error ، بر روی آیکون خطا کلیک کنید . در اینجا به دلیل عدم پیش بینی خطا هیچ واکنشی از سوی مرورگر **انجام** نمی شود :

Example

کد

```
< script type="text/javascript" >
 function show_error()
  {
   document.wriet ( "Welcome User!" ) ;
  }
< /script >

< input type="button" id="Button1" value="click me !" onclick="show_error()" />
```

مثل ۱ : در مثال زیر ، تابع show_error2 **قرار داده‌شده است** . این تابع نیز همانند تابع قبلی می خواهد یک پیام خوش آمد به کاربر اعلام کند . این تابع نیز دارای اشتباه تایپی در دستور document.write است ، با این تفاوت که با ساختار try ... catch قرار داده شده ، تعیین شده است در صورت بروز خطا یک پیام هشدار به کاربر اعلام شود . توسط متغیر err که در پرانتز مقابل واژه کلیدی catch تعریف شده است ، متن پیام error در خاصیت description ذخیره شده و سپس به کاربر اعلام می شود . به کد مثال دقت کنید :

Example

کد

```
< script type="text/javascript" >
 function show_error2 ( )
  {
   try
    {
     document.wriet ( "Welcome User!" ) ;
    }
   catch ( err )
    {
     alert ( "There was an error on this page . \n\n" + "Error Description : "
+ err.description +"\n\nClick Ok for continue" ) ;
```

```
    }
  }
< /script >

< input type="button" id="btnclick2" value="click me ! to see error report" onclick="show_error(
)" />
```

دستور throw :

توسط دستور throw در جاوا اسکریپت می توان یک خطایابی کامل تر را انجام داد . با استفاده از این دستور به همراه ساختار try ... catch ، می توان روند اجرای برنامه و بروز خطا را کاملا تحت کنترل داشت و یک پیغام خطا دقیق طراحی کرد .

نکته : دستور throw به تنهایي کاربردی نداشته و باید آنرا با ساختار دستوری try ... catch به کار برد .

مثل : در مثال زیر سعی شده است ، تا چگونگی استفاده از یک دستور throw را در قالب ساختار try ... catch توضیح داد . در مثال زیر یک اسکریپت ساده برای دریافت ورودی از کاربر طراحی شده است . تابع Enter_Num که توسط دکمه فرمان Enter Number فراخوانی می شود ، در ابتدا یک کادر متن برای دریافت ورودی از کاربر ، نمایش می دهد . مقدار دریافتی از کاربر در متغیر Num ذخیره می شود . سپس در یک ساختار try ... catch ، مقدار دریافتی از کاربر بررسی می شود .

در حالت اول ، چنانچه کاربر مقداری را در کادر وارد نکرده و آنرا خالی ارسال کرده باشد ، برنامه خطای ۱ را شناسایي و یک پیام هشدار مبنی بر وارد کردن عدد نمایش داده و سپس مجددا تابع Enter_Num را برای دریافت مقدار صحیح اجرا می کند . در حالت دوم اگر کاربر عددی بزرگتر از ۱۰۰ را وارد کرده باشد ، برنامه خطای دوم را شناسایي کرده و یک پیغام هشدار مبنی بر بزرگ بودن عدد وارده را نمایش داده و مجددا تابع Enter_Num را اجرا می کند . به کد مثال و نحوه استفاده از دستور throw دقت کنید :

Example

کد

```
< script type="text/javascript" >
 function Enter_Num ( )
  {
   var Num = prompt ( "Enter a number please : " , "" ) ;
   try
    {
     if ( !Num )
       throw "Error1"
     else if ( Num > 100 )
       throw "Error2"
    }
```

```
    catch ( er )
      {
      if ( er == "Error1" )
       {
          alert ( "Plese enter a number !" ) ;
          Enter_Num ( ) ;
        }
      if ( er == "Error2" )
       {
          alert ( "Number too big . Enter a smaller number !" ) ;
          Enter_Num( ) ;
        }
      }
  }
< /script >

< input type="button" id="BtnEnter" value="Enter Number " onclick="Enter_Num( )" />
```

ساختار دستوری try ... catch

در برنامه نویسی و طراحی صفحات وب ، گاهی اوقات ممکن است خطاهایي رخ دهد . این خطاها می تواند دلایل مختلفی داشته باشد ، از قبیل :

- اشتباه تایپی در متن دستورات صفحه از سوی طراح صفحه.
- واردنمودن اطلاعات اشتباه یا نادرست از سوی کاربر.
- انواع خطاهای ممکن زمان اجرای مرورگر.

اگر کاربر که در حال مشاهده یک صفحه است ، در هنگام کار با خطا مواجه شود و عملا راهی برای رفع آن اشکال برایش وجود نداشته باشد ، مطمًئنا از ادامه مشاهده و کار با صفحه صرف نظر خواهد کرد . این وظیفه برنامه نویس است که با برنامه ریزی صحیح ، خطاهای احتمالی را پیدا کرده و راه حل های مناسب را برای آنها طراحی کند .

از ساختار کنترلی try ... catch در جاوا اسکریپت ، برای پیدا کردن خطاها و error های احتمالی و تعیین اقدامات اصلاحی در صورت بروز خطا ، استفاده می شود .
این ساختار از ۲ بخش try و catch تشکیل شده است . کد اصلی برنامه که می خواهیم اجرا شود و احتمال می دهیم دارای خطا باشد ، را در قسمت try قرار داده و اقدامات اصلاحی را که می خواهیم در صورت بروز خطا انجام شود ، را در قسمت catch می گذاریم .
برنامه در هنگام رسیدن به ساختار ، دستورات قسمت try را انجام می دهد و در صورت مواجه با خطا در دستورات ، بخش catch را اجرا خواهد کرد .
نکته مهم : در صورت عدم برخورد با خطا ، دستورات قسمت catch به هیچ وجه اجرا نخواهد شد .
نکته مهم ۲ : در قسمت catch ، می توان در پرانتز مقابل واژه کلیدی catch ، یک عبارت یا متغیر تعریف کرد ، که این متغیر خصوصیات error به وجود آمده را در خود نگهداری کرد . یکی از این خواص ، شرح یا description خطا است ، که می توان به شکلی که در مثال ۲ نشان داده شده است ، به شرح خطا دسترسی داشته و آنرا به کاربر اطلاع داد .
شکل کلی تعریف یک ساختار try ... catch به صورت زیر است :

```
try
 {
  دستورات مورد نظر برای اجرا که احتمال خطا دارد
 }
catch ( نام یک متغیر )
 {
  اقدامات اصلاحی مورد نظر در صورت وقوع خطا
 }
```

مثل ۱ : در مثال زیر ، یک تابع به نام show_error **قرار داده‌شده است** . این تابع توسط دکمه فرمان click me فراخوانی شده و قصد دارد تا یک پیغام خوش آمد به کاربر اعلام کند . اما در متن دستور یک اشتباه تایپی وجود دارد و آن اینکه به جای **عبارت** documnet.write ، نوشته شده است document.wriet ، به همین دلیل پیغام خروجی چاپ نشده و در صفحه error رخ می دهد . **اگر دقت** کنید در نوار پایین مرورگر (Status Bar) **علامت** Error on page قرار گرفته است . برای مشاهده error ، بر روی آیکون خطا کلیک کنید . در اینجا به دلیل عدم پیش بینی خطا هیچ واکنشی از سوی مرورگر انجام نمی شود :

Example

کد

```
< script type="text/javascript" >
 function show_error()
  {
   document.wriet ( "Welcome User!" ) ;
  }
< /script >

< input type="button" id="Button1" value="click me !" onclick="show_error()" />
```

مثل ۱ : در مثال زیر ، تابع show_error2 **قرار داده‌شده است** . این تابع نیز همانند تابع قبلی می خواهد یک پیام خوش آمد به کاربر اعلام کند . این تابع نیز دارای اشتباه تایپی در دستور document.write است ، با این تفاوت که با ساختار try ... catch قرار داده شده ، تعیین شده است در صورت بروز خطا یک پیام هشدار به کاربر اعلام شود . توسط متغیر err که در **پرانتز** مقابل واژه کلیدی catch تعریف شده است ، متن پیام error در خاصیت description ذخیره شده و سپس به کاربر اعلام می شود . به کد مثال دقت کنید :

Example

کد

```
< script type="text/javascript" >
 function show_error2 ( )
  {
   try
    {
     document.wriet ( "Welcome User!" ) ;
    }
   catch ( err )
    {
     alert ( "There was an error on this page . \n\n" + "Error Description : "
+ err.description +"\n\nClick Ok for continue" ) ;
```

```
    }
  }
< /script >

< input type="button" id="btnclick2" value="click me ! to see error report" onclick="show_error(
)" />
```

دستور throw :

توسط دستور throw در جاوا اسکریپت می توان یک خطایابی کامل تر را انجام داد . با استفاده از این دستور به همراه ساختار try ... catch ، می توان روند اجرای برنامه و بروز خطا را کاملا تحت کنترل داشت و یک پیغام خطا دقیق طراحی کرد .
نکته : دستور throw به تنهایي کاربردی نداشته و باید آنرا با ساختار دستوری try ... catch به کار برد .

در مثال زیر سعی شده است ، تا چگونگی استفاده از یک دستور throw را در قالب ساختار try ... catch توضیح داد .
مثل : در مثال زیر یک اسکریپت ساده برای دریافت ورودی از کاربر طراحی شده است . تابع Enter_Num که توسط دکمه فرمان Enter Number فراخوانی می شود ، در ابتدا یک کادر متن برای دریافت ورودی از کاربر ، نمایش می دهد . مقدار دریافتی از کاربر در متغیر Num ذخیره می شود . سپس در یک ساختار try ... catch ، مقدار دریافتی از کاربر بررسی می شود .
در حالت اول ، چنانچه کاربر مقداری را در کادر وارد نکرده و آنرا خالی ارسال کرده باشد ، برنامه خطای ۱ را شناسایي و یک پیام هشدار مبنی بر وارد کردن عدد نمایش داده و سپس مجددا تابع Enter_Num را برای دریافت مقدار صحیح اجرا می کند . در حالت دوم اگر کاربر عددی بزرگتر از ۱۰۰ را وارد کرده باشد ، برنامه خطای دوم را شناسایي کرده و یک پیغام هشدار مبنی بر بزرگ بودن عدد وارده را نمایش داده و مجددا تابع Enter_Num را اجرا می کند . به کد مثال و نحوه استفاده از دستور throw دقت کنید :

Example

کد

```
< script type="text/javascript" >
 function Enter_Num ( )
  {
    var Num = prompt ( "Enter a number please : " , "" ) ;
    try
     {
       if ( !Num )
         throw "Error1"
       else if ( Num > 100 )
         throw "Error2"
     }
```

```
    catch ( er )
      {
      if ( er == "Error1" )
       {
          alert ( "Plese enter a number !" ) ;
          Enter_Num ( ) ;
       }
      if ( er == "Error2" )
       {
          alert ( "Number too big . Enter a smaller number !" ) ;
          Enter_Num( ) ;
       }
      }
  }
< /script >

< input type="button" id="BtnEnter" value="Enter Number " onclick="Enter_Num( )" />
```

ساختار دستوری onerror :

استفاده ازساختار onerror ، روش قدیمی خطا یابی در صفحات وب در زبان جاوا اسکریپت است .
در صفحه قبل ، نحوه استفاده از ساختار دستوری try ... catch را توضیح دادیم . از ساختار دستوری onerror نیز برای خطایابی در صفحات وب ، ولی با روشی متفاوت استفاده می شود .
رویداد onerror ، هر زمان که خطایي در یک اسکریپت در صفحه به وجود بیاید ، تحریک شده و اتفاق می افتد . برای استفاده از رویداد onerror ، برنامه نویس باید تابعی را طراحی کند که خطا به وجود آمده در صفحه را مدیریت (Handle) کند . رویداد onerror در زمان بروز خطا ، تابع مدیریت خطا (Event Handler Function) را فراخوانی خواهد کرد ، که در این صورت دستورات پیش بینی شده برای مواجه با خطا اجرا خواهد شد .
تابع مدیریت خطا با ۳ پارامتر (آرگومان) ، به شرح زیر فراخوانی خواهد شد :

۱. msg : متن پیام خطایي که توسط مرورگر تولید شده و شامل توضیحاتی راجع به error رخ داده است ، را ارائه می دهد.
۲. url : مسیر کامل صفحه ای که خطا در آن اتفاق افتاده را شامل می شود.
۳. l : شماره خط کدی که خطا در آن اتفاق افتاده است ، را نگهداری می کند.

شکل کلی تعریف یک ساختار onerror ، به همراه تابع مدیریت خطا به صورت زیر است :

```
onerror  =  نام تابع مدیریت خطا

function   نام تابع مدیریت خطا ( )
  {
    دستورات مدیریت خطا
    return false Or true
  }
```

نکته مهم : تابع مدیریت خطا دارای یک مقدار بازگشتی است ، که توسط طراح و به وسیله دستور return تعیین می شود . این مقدار می تواند یکی از ۲ حالت True یا False باشد .این مقدار تعیین می کند که آیا مرروگر در هنگام بروز خطا ، علامت خطا را در نوار پایین مرورگر (Status Bar) نشان داده و گزارش استاندارد راجع به خطا به وجود آمده اعلام کند یا خیر . در صورت تعیین و باز گرداندن مقدار True ، مرررورگر در هنگام وقوع یک error ، در نوار Status Bar علامت و گزارش خطا را اعلام نمی کند ولی در صورت بازگرداندن مقدار False ، مرورگر در نوار پایین علامت خطا را نمایش داده و یک گزارش کامل از خطا را به صورت استاندارد اعلام می کند .

مثل : در مثال زیر یک تابع مدیریت خطا با نام ErrorHandler برای واکنش در زمان بروز خطا طراحی شده است . این تابع قرار است در هنکام بروز خطا یک کادر پیام حاوی توضیح نوع خطا ، آدرس کامل صفحه ای که خطا در آن روي داده و شماره خط کد اشتباه را به کاربر اعلام کند .

از طرف دیگر در تابع PageEroor ، یک اشتباه تایپی وجود دارد و به جای عبارت alert نوشته شده است alertt . این اشتباه باعث بروز خطا در صفحه شده که در نتیجه تابع مدیریت خطا فراخوانی شده و پیغام خود را نمایش می دهد . همچنین به علت باز گرداندن مقدار true توسط تابع مدیریت خطا ، مرورگر علامت خطا را در نوار Status Bar نشان نمی دهد :

Example

```
<script type="text/javascript">
  onerror = ErrorHandler
  function ErrorHandler ( msg , url , l )
    {
      alert ( "Error Discription : " + msg + "\nPage URL : " + url + "\nLine Number : " + l ) ;
      return true ;
    }
</script>

<script type="text/javascript">
  function PageError ( )
    {
      alertt ( "developer" ) ;
    }
</script>

<input type="button" id="BtnError" value="Click to view error report" onclick="PageError()" />
```

کد

مبحث شی گرایي در جاوا اسکریپت

مقدمه ای بر مبحث شی گرایي

جاوا اسکریپت ، یک زبان برنامه نویسی شی گرا یا (Object Oriented Programming) **است** . این زبان به برنامه نویس ، قابلیت استفاده از اشیای پیش ساخته و یا تعریف و ایجاد اشیای جدید مورد نیاز خود را می دهد .

توجه : در این قسمت ، مروری کوتاه بر مفهوم برنامه نویسی شی گرا در جاوا اسکریپت و نحوه استفاده از آن خواهیم داشت . در قسمت های بعدی به معرفی انواع اشیا پیش ساخته در Java Script می پردازیم .

مفهوم کلاس :

کلاس (class) ، اساسی ترین مفهوم شی گرایي است . یک کلاس یک الگو یا چهارچوب از پیش تعریف شده است که اشیا (objects) از روی آن ساخته می شوند . هر کلاس مجموعه ای از خواص (Properties) و متدها (Methods) **است** ، که کلاس آنها را به اشیایي که از رویش ساخته می شوند ، به ارث می دهد .

یک کلاس همانند یک دستورالعمل یا نقشه برای ایجاد اشیای جدید مشتق شده از آن است . برای مثال موجودیت دانشجو در مجموعه دانشگاه یک کلاس است ، که هر دانشجو یک شی ساخته شده از روی آن است .

مثل ۲ : متغیرهای متنی یا string ، یکی از کلاس های پیش ساخته در جاوااسکریپت است . **درقسمت بعد با تمام خواص و** متدهای آن آشنا خواهید شد .

مفهوم شی :

هر شی ، یک نمونه ساخته شده از روی کلاس مادر است . شی خصوصیات و متدهای خود را از کلاس مادر به ارث می برد .

هر یک از خصوصیات یک شی ، در برگیرنده مقدار مختص به آن خاصیت برای آن شی است . به عبارت دیگر ، از هر یک از خواص یک کلاس ، یک نسخه یا کپی منحصر به فرد به هر شی اختصاص می یابد ، که مقدار آن با مقدارش برای سایر اشیای آن کلاس ، متمایز و منحصر به فرد است .

مثل : برای مثال ، هر دانشجو که از روی کلاس دانشجو ایجاد می شود ، دارای خاصیت های نام ، نام خوانادگی ، شماره دانشجويي و ... غیره مخصوص به خود است . گرچه مقدار این خواص در نمونه های مختلف دانشجوها ممکن است در برخی از آنها مثل نام یکسان و در برخی مثل شماره دانشجويي یکتا باشد ، ولی مقدار آن برای هر دانشجو جدا و متمایز است .

مفهوم متد :

متدها ، عملیات یا رفتارهایي است که اشیا یک کلاس می توانند انجام دهند .

مثل : برای مثال هر دانشجو که از روی کلاس دانشجو ساخته می شود ، دارای عملیات های یکسانی مثل ثبت نام ، انتخاب واحد و ... **است** .

نحوه تعریف یک کلاس و ایجاد یک شی جدید در جاوا اسکریپت :

در اینجا ، به معرفی یکی از روش های تعریف یک کلاس و ایجاد شی جدید از روی آن می پردازیم . سپس با نحوه دسترسی به خواص . متدهای اشیا ، آشنا خواهیم شد .
برای ایجاد یک کلاس ، از واژه کلیدی function به شکل کلی زیر استفاده می شود :

```
function نام کلاس ( Parameter1 , Parameter2 , ... )
 {
   this.۱ نام خاصیت = Parameter1 ;
   this.۲ نام خاصیت = Parameter2 ;
   .
   .
   .
 }
```

توضیح : ابتدا واژه کلیدی function را نوشته ، پس از آن نام کلاس ، که در سطح برنامه برای شناسایي و دسترسی به کلاس مورد نظر از آن استفاده می شود را تعیین می کنیم . به تعداد خواص مورد نظر برای اشیای کلاس ، در پرانتز جلوی نام کلاس ، پارامتر تعریف کرده ، تا در هنگام فراخوانی کلاس برای ایجاد شی جدید در برگیرنده مقادیر لازم برای خواص شی باشند .
سپس در بدنه دستورات کلاس برای تعریف و مقدار دهی هر یک از خواص اشیای کلاس ، از وایه کلیدی this به شکل کلی زیر استفاده می شود :

```
this.نام خاصیت = نام پارامتر
```

واژه کلیدی this ، در هر لحظه به شی جاری از کلاس که در آن لحظه در حال ساخت یا ویرایش است ، اشاره می کند .
پس از تعریف ساختار کلاس ، برای ایجاد یک شی جدید از حالت کلی زیر استفاده می شود :

```
var نام شی جدید = function نام کلاس ( Parameter1 Value , Parameter2 Value , ... ) ;
```

برای دسترسی و مقدار دهی هر یک از خواص یک شی از روش کلی زیر استفاده می شود :

```
نام شی . نام خاصیت
```

مثل : در مثال زیر یک کلاس به نام student ایجاد کرده ایم . هر شی این کلاس ۳ خاصیت نام (Name) ، نام خوانادگی (Family) و شماره دانشجویي (ID) را خواهد داشت . در هنگام فراخوانی کلاس برای ایجاد یک شی جدید ، مقادیر این ۳ خاصیت به ترتیب به پارامترهای f , n و i ارسال خواهند شد .
در مثال ، پس از تعریف کلاس به ایجاد یک شی جدید به نام St1 و مقدار دهی خواص آن پرداخته ایم . سپس توسط سه دستور چاپ در خروجی مشخصات کامل دانشجو جدید را در خروجی چاپ کرده ایم . در انتها هم شماره دانشجویي ، دانشجو مذکور را دوباره مقدار دهی کرده و مقدار جدید آن را چاپ کرده ایم :

Example	

<script type ="text/javascript">
function Student (n , f , i)
{
this.Name = n ;
this.Family = f ;
this.ID = i ;
}
var St1 = new Student ("Mehrdad" , "Fattahi" , 122092) ;
document.write ("Name :" + St1.Name +"
");
document.write ("Family : " + St1.Family +"
");
document.write ("Studnet ID : " + St1 .ID +"
");
St1.ID = 299302 ;
document.write ("New Student ID : " + St1.ID) ;
</script>
	کد

Name :Mehrdad Family : Fattahi Studnet ID : 122092 New Student ID : 299302	خروجی

شی string :

شی string یکی از اشیای پیش ساخته در جاوا اسکریپت است . این شی برای دستکاری و انجام عملیات بر روی داده های متنی استفاده می شود .

هر متغیر از نوع متنی ، نمونه ای از شی string **است** .

در این صفحه به معرفی خواص شی string و در صفحه بعدی به معرفی متدهای آن خواهیم پرداخت .

خواص شی string	متدهای شی string

خواص شی string :

در جدول زیر ، مجموعه خواص شی string ارائه شده است . برای دریافت اطلاعات درباره هر کدام ، بر روی نام آن کلیک کنید :

نام خاصیت	شرح
constructor	**نام تابع‌سازنده‌**شی مورد نظر را بر می گرداند .
length	تعداد کاراکترهای یک عبارت متنی را بر می گرداند .
prototype	به برنامه نویس امکان اضافه کردن خواص و متدهای جدید را به شی می دهد .

خاصیت constructor :

این خاصیت ، نام تابع سازنده شی مورد نظر را بر می گرداند . تغبع سازنده ، تابعی است که در هنگام تعریف اولیه ، شی را به **وجود آورده است** . شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	object.constructor * object = نام شی مورد نظر

مثل : در مثال زیر با استفاده از تابع string ( ) یک شی string جدید به نام txtname ایجاد کرده ایم . به وسیله دستور document.write ، نوع تابع سازنده شی را نشان داده ایم :

نکته : توجه شود شکل صحیح نوشتاری تابع String با S بزرگ است .

Example	

< script type="text/javascript" >
  var txtname = new String ( ) ;
  document.write (txtname.constructor) ;
< /script >
	کد

function String() { [native code] }	خروجی

خاصیت length :

این خاصیت ، تعداد کاراکترها (حروف) عبارت متنی را بر می گرداند . فاصله بین حروف نیز یک کاراکتر محسوب می شود . شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	object.length

مثل : در مثال زیر یک متغیر رشته ای را تعریف و سپس به وسیله دستور document.write ،طول آن را نشان داده ایم :

Example	

<script type="text/javascript">
  var name = "Developer Studio" ;
  document.write (name.length) ;
</script>
	کد

16	خروجی

خاصیت prototype :

به وسیله این خاصیت می توان خواص و متدهای جدید مورد نظر خود را به شی اضافه کرد . **پس از** تعریف خاصیت یا متد ، می توان از آن همانند خواص ذاتی شی استفاده کرد . شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	object.prototype.name = value * object = نام شی مورد نظر name = نام خاصیت یا متد جدید value = مقدار اولیه خاصیت جدید یا نام متد

مثل : در مثال زیر ابتدا یک کلاس به نام Student که دارای ۳ خاصیت نام (Name) ، نام خوانادگی (Family) و شماره دانشجویی (ID) است را ایجاد کرده و یک شی جدید به نام St 2 از روی آن ساخته ایم .**سپس** به کلاس یک خاصیت جدید به نام Major اضافه کرده ایم ، که قرار است رشته تحصیلی را برای دانشجو نگهداری کند . درپایان ، خاصیت جدید را برای شی St2 مقدار دهی کرده و آن را در خروجی نشان داده ایم :

Example

کد

```
<script type="text/javascript">
 function Student ( n , f , i )
  {
    this.Name = n ;
    this.Family = f ;
    this.ID = i ;
  }
 var St2 = new   Student ( "Mehrdad" , "Fattahi" , 122092 ) ;
 Student.prototype.Major = null ;        ایجاد خاصیت جدید برای کلاس به نام Major
 St2.Major = "Soft Ware";            مقدار دهی خاصیت جدید برای شی St2
 document.write ( St2.Major ) ;
</script>
```

خروجی

```
Soft Ware
```

شی string :

در این بخش به معرفی متدهای شی string می پردازیم . در قسمت قبل خواص این شی را بررسی کردیم .

متدهای شی string :

متدهای شی string در جدول زیر معرفی شده اند . برای دریافت اطلاعات راجع به هر کدام بر روی نام آن کلیک کنید :

نام خاصیت	شرح
big ( )	برای نمایش متن یک متغیر رشته ای با اندازه بزرگتر استفاده می شود .
blink ( )	برای نمایش متن یک متغیر رشته ای به صورت چشمک زن استفاده می شود .
bold ( )	برای نمایش متن یک متغیر رشته ای به صورت توپر (درشت) استفاده می شود .
charAt ( )	برای نمایش مقدار یک حرف (کاراکتر) مورد نظر در یک متغیر رشته ای اسفاده می شود .
charCodeAt ( )	برای نمایش کد اسکی یک حرف (کاراکتر) مورد نظر در یک متغیر رشته ای اسفاده می شود .
concat ( )	از این متد برای چسباندن و اضافه کردن دو یا چند متغیر رشته ای به هم استفاده می شود .
fixed ( )	این متد ، برای نمایش متن متغیر رشته ای با قلم ی (font) شبیه نوشته های تلگراف استفاده می شود .
fontcolor ( )	از این متد ، برای نمایش متن یک متغیر رشته ای به یک رنگ دلخواه استفاده می شود .
fontsize ( )	از این متد ، برای نمایش یک متغیر رشته ای در یک اندازه خاص استفاده می شود .
indexOf ( )	این متد ، شماره مکان قرار گیری اولین نمونه یک حرف یا کلمه را در یک متغیر متنی را بر می گرداند .

italics ( )	از این متد ، برای نمایش متن یک متغیر رشته ای به صورت کج استفاده می شود .
lastIndexOf ( )	این متد ، شماره مکان قرار گیری آخرین نمونه یک حرف یا کلمه را در یک متغیر متنی را بر می گرداند .
link ( )	از این متد برای تبدیل متن یک متغیر رشته ای به یک پیوند (HyperLink) استفاده می شود .
match ( )	ازاین متد ، برای جستجوی یک حرف یا کلمه در یک متغیر متنی استفاده می شود .
replace ( )	از این متد برای جایگزینی یک حرف یا کلمه خاص در یک متغیر متنی و جایگزینی آن با یک مقدار جدید استفاده می شود .
search ( )	از این متد برای جستجو یک حرف یا کلمه خاص در یک متغیر متنی استفاده می شود .
slice ( )	از این متد برای جستجو یک حرف یا کلمه خاص در یک متغیر متنی استفاده می شود .
small ( )	از این متد برای متن یک متغیر متنی با اندازه ای کوچکتر از حد معمول استفاده می شود .
splite ( )	از این متد برای تقسیم کردن یک متغیر متنی به آرایه ای از کاراکترها استفاده می شود .
strike ( )	از این متد برای نماژش متن یک متغیر متنی با یک خط کشیده شده بر روی آن استفاده می شود .
sub ( )	از این متد برای نمایش یک متن به صورت اندیس استفاده می شود .
substr ( )	از این متد برای برش تعداد معینی از کاراکترهای یک متغیر متنی استفاده می شود .
substring ( )	از این متد برای برش تعداد معینی از کاراکترهای یک متغیر متنی بین دو نقطه مشخص استفاده می شود .
sup ( )	از این متد برای نمایش یک متن به صورت زیر نویس استفاده می شود .
toLowerCase ( )	از این متد برای نمایش متن یک متغیر رشته ای با حروف کوچک استفاده می شود .
toUpperCase ( )	از این متد برای نمایش متن یک متغیر رشته ای با حروف بزرگ استفاده می شود .

اشیای پیش ساخته در Java Script

شی Date :

از این شی برای دستکاری و کار با تاریخ و زمان استفاده می شود . به طور کلی هر متغیری از نوع تاریخ و زمان نمونه ای از شی Date خواهد بود .

خواص شی Date متدهای شی Date

نحوه تعریف یک متغیر جدید از نوع Date :

برای تعریف یک متغیر جدید از نوع تاریخ و زمان از تابع Date ، به صورت کلی زیر استفاده می شود :

```
var  نام متغیر  =  new  Date () ;
مثال :  var  NowTime  =  new  Date () ;
```

نکته ۱ : هر متغیری از نوع Date ، که به روش فوق ایجاد شود ، زمان و تاریخ جاری سیستم در لحظه ایجاد خود را ، به عنوان مقدار پیش فرض در درون خود نگهداری می کند . این مقدار شامل مخخف نام روز جاری، مخفف نام ماه جاری ، شماره روز جاری در ماه ، سال جاری ، ساعت دقیق که به صورت ساعت ، دقیقه و ثانیه است و فرمت ساعت خواهد بود .

مثل : در مثال زیر یک متغیر به نام NowTime را به روش اشاره شده ایجاد و مقدار دهی کرده ایم . سپس به وسیله دستور document.write مقدار آن را بر روی صفحه نمایش داده ایم . این متغیر هر بار که صفحه مجددا بار گذاری شده و یا Refresh می شود ، مقدار آن دوباره به تاریخ و ساعت جاری سیستم Set شده و تغییر می کند . بنابراین مقدار آن ثابت نیست و با هر بار لود شدن صفحه و و اجرای مجدد اسکریت ، ساعت و تاریخ جدید جایگزین مقدار قبلی می شود . برای درک بهتر هر چند لحظه یکبار صفحه را Refresh کرده و به مقدار خروجی دقت کنید :

Example	

< script type="text/javascript" >
  var NowTime = new Date ( ) ;
  document.write (NowTime) ;
< /script >
	کد

Fri Oct 19 2012 00:01:35 GMT+0330 (Iran Standard Time)	خروجی

نکته ۲ : همچنین می توان در هنگام تعریف یک متغیر جدید از نوع Date ، یک تاریخ مورد نظر را به عنوان تاریخ ذخیره شده در آن متغیر را در پرانتز جلوی تابع Date تعریف کرد . در این صورت سیستم به صورت اتوماتیک ، تاریخ تعیین شده

را به فرمت مورد استفاده در جاوا اسکریپت تبدیل کرده و در متغیر ذخیره خواهد کرد .

مثل : در مثال زیر ابتدا یک متغیر به نام MyDateرا ایجاد کرده و یک تاریخ را نیز به عنوان مقدار برای آن در نظر گرفته ایم . سپس آن متغیر را بر روی صفحه در خروجی چاپ کرده ایم . همانطور که می بینید ، مقدار خروجی برابر با مقدار تعیین شده است و ارتباطی با تاریخ و ساعت جاری سیستم ندارد :

Example	
< script type="text/javascript" > var MyDate = new Date ("12/05/2004") ; document.write (MyDate) ; < /script >	کد
Sun Dec 05 2004 00:00:00 GMT+0330 (Iran Standard Time)	خروجی

خواص شی Date :

خاصیت constructor : این خاصیت ، نام تابع سازنده شی مورد نظر را بر می گرداند . تغبع سازنده ، تابعی است که در هنگام تعریف اولیه ، شی را به وجود آورده است . شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	object.constructor * object = نام شی مورد نظر

مثل : در مثال زیر با استفاده از تابع Date ( ) یک شی MyDate2 جدید به نام txtname ایجاد کرده ایم . به وسیله دستور document.write ، نوع تابع سازنده شی را نشان داده ایم :

نکته : توجه شود شکل صحیح نوشتاری تابع Date با D بزرگ است .

Example	
< script type="text/javascript" > var MyDate2 = new Date ( ) ; document.write (MyDate2.constructor) ; < /script >	کد
function Date() { [native code] }	خروجی

شی Date :

در این بخش به معرفی متدهای شی Date می پردازیم . در قسمت قبل خواص این شی را بررسی کردیم .

متدهای شی Date :

متدهای شی Date در جدول زیر معرفی شده اند . برای دریافت اطلاعات راجع به هر کدام بر روی نام آن کلیک کنید :

نام متد	شرح
getDate ( )	این متد ، شماره روز جاری در ماه را بر می گرداند .
getDay ( )	این متد ، شماره روز جاری در هفته را بر می گرداند .
getMonth ( )	این متد ، شماره ماه جاری را بر می گرداند .
getYear ( )	این متد ، شماره سال جاری را بر می گرداند .
getFullYear ( )	این متد نیز ، شماره سال جاری را بر می گرداند .
getHours ( )	این متد مقدار ساعت را بر می گرداند .
getMinutes ( )	این متد شماره دقیقه را در یک متغیر زمانی بر می گرادند .
getSeconds ( )	این متد شماره ثانیه ها را در یک متغیر زمانی بر می گرادند .
getMilliseconds ( )	این متد شماره میلی ثانیه ها را در یک متغیر زمانی بر می گرادند .
getTime ( )	این متد تعداد ثانیه های سپری شده از تاریخ اول ژانویه سال ۱۹۷۰ تا به امروز را بر می گرداند
getTimezoneOffset ( )	این متد اختلاف زمانی بین زمان جاری سیتسم را با ساعت بین المللی گرینویچ را بر حسب دقیقه بر می گرداند .

parse ( )	این متد یک تاریخ را به صورت رشته ای دریافت کرده و تعداد میلی ثانیه های سپری شده از زمان اول ژانویه سال ۱۹۷۰ تا به امروز را بر می گرداند
setDate ( )	این متد برای تغییر مقدار عددی تاریخ روز در یک متغیر زمانی استفاده می شود .
setMonth ( )	این متد برای تغییر شماره ماه و تاریخ روز در یک متغیر زمانی استفاده می شود .
setFullYear( )	این متد برای تغییر مقدار سال در یک متغیر زمانی استفاده می شود .
setYear( )	این متد برای تغییر مقدار سال در یک متغیر زمانی استفاده می شود .
setHours( )	این متد برای تغییر مقدار عددی ساعت در یک متغیر زمانی استفاده می شود .
setMinutes( )	این متد برای تغییر مقدار عددی دقیقه در یک متغیر زمانی استفاده می شود .
setSeconds( )	این متد برای تغییر مقدار عددی ثانیه در یک متغیر زمانی استفاده می شود .
setMilliseconds( )	این متد برای تغییر مقدار عددی میلی ثانیه در یک متغیر زمانی استفاده می شود .
setTime( )	این متد برای محاسبه و ایجاد یک تاریخ جدید ، به وسیله اضافه یا کم کردن تعداد معینی میلی ثانیه به تاریخ اول ژانویه سال ۱۹۷۰ استفاده می شود .
toString( )	از این متد برای تبدیل یک متغیر زمانی به یک عبارت رشته ای استفاده می شود .

شی Math - انجام امور ریاضی در جاوا اسکریپت

شی (Math) در جاوا اسکریپت :

از شی Math در جاوا اسکریپت برای انجام امور ریاضی استفاده می شود . این شی شامل تعداد زیادی متد و تابع برای انجام کارهای ریاضی می باشد . همچنین اینن شی دارای تعداد ثابت عددی مثل عدد P است که می توانید از آنها در محاسبات خود استفاده نمایید .

شکل کلی تعریف و استفاده از یک ثابت عددی یا متد شی Math در جاوا اسکریپت به صورت زیر است :

Syntax	Math.نام ثابت عددی ;
	یا
	Math.نام متد ( ) ;
	مثال : var MyNum = Math.PI ; // عدد پی (۳.۱٤) را به متغیر نسبت می دهد .
	مثال : var MyNum = Math.sqrt(9) ; // جزر عدد داخل پرانتز را حساب کرده و به متغیر نسبت می دهد

ثابت های عددی شی Math :

همانطور که اشاره شد شی Math ، دارای تعدادی ثابت عددی مثل عدد P است که از آنها می توانید در انجام امور ریاضی و محاسبات خود استفاده نمایید . در لیست زیر این ثابت ها قرار داده شده اند . برای دریافت توضیحات بیشتر و مثال های عملی بر روی نام هر ثابت کلیک نمایید :

ثابت عددی	مقدار حدودی	کاربرد
E	2.718	این ثابت عددی مقدار عدد حقیقی e را بر می گرداند .
LN2	0.693	این ثابت عددی ، مقدار لگاریتم طبیعی ۲ را بر می گرداند .
LN10	2.302	این ثابت عددی ، مقدار لگاریتم طبیعی ۱۰ را بر می گرداند .
LOG2E	1.442	این ثابت عددی ، مقدار لگاریتم عدد e را بر مبنای ۲ بر می گرداند .
LOG10E	0.434	این ثابت عددی ، مقدار لگاریتم عدد e را بر مبنای ۱۰ بر می گرداند .

این ثابت عددی ، مقدار عدد P را بر می گرداند .	3.14	PI

متدها و تابع های عددی شی Math :

پس از آشنایی با ثابت های عددی شی Math ، در این قسمت به معرفی و توضیح تابع ها و متدهای ریاضی این شی می پردازیم که می توانید از آنها در انجام امور ریاضی و محاسبات خود استفاده نمایید . در لیست زیر این تابع ها قرار داده شده اند . برای دریافت توضیحات بیشتر و مثال های عملی بر روی نام هر تابع کلیک نمایید :

کاربرد	نام تابع
این تابع قدر مطلق عددی x را بر می گرداند .	abs (x)
این تابع آرک کوسینوس عدد x را بر می گرداند .	acos (x)
این تابع آرک سینوس عدد x را بر می گرداند .	asin (x)
این تابع آرک تانژانت عدد x را بر می گرداند .	atan (x)
این تابع عدد x رو به بالا گرد کرده و نزدیک ترین عدد صحیح بدان را بر می گرداند .	ceil (x)
این تابع کوسینوس x را بر می گرداند .	cos(x)
این تابع مقدار عدد e به توان x را بر می گرداند .	exp(x)
این تابع عدد x را رو به پایین گرد کرده و نزدیک ترین عدد صحیح بدان را بر می گرداند .	floor(x)
این تابع لگاریتم x را بر مبنای عدد e بر می گرداند .	log(x)
این تابع بزرگترین عددی را که به عنوان پارامتر به آن ارسال شده است را بر می گرداند .	max(a,b, .. ,x,y,z)
این تابع کوچکترین عددی را که به عنوان پارامتر به آن ارسال شده است را بر می گرداند .	min(a,b, .. ,x,y,z)

pow(a,b)	این تابع عدد a را به توان b رسانده و نتیجه را بر می گرداند .
random()	این تابع یک عدد تصادفی بین ۰ و ۱ را انتخاب کرده و به عنوان خروجی بر می گرداند .
round(x)	این تابع عدد x را گرد کرده و به نزدیکترین عدد صحیح بدان تبدیل می کند .
sin(x)	این تابع مقدار سینوس x را بر می گرداند .
tan(x)	این تابع مقدار تانژانت x را بر می گرداند .

شی (Number) در جاوا اسکریپت :

در هنگام کار با اسکریپت های جاوا اسکریپت ، به کار با اعداد نیاز داریم . جاوا اسکریپت بر خلاف سایر زبان های برنامه نویسی معمول ، یک زبان با انواع داده ای خاص نیست . یعنی چه ؟

برای مثال در زبان C ، انواع داده ای مثل integer , short , long و یا float را برای اعداد داریم . اما در جاوا اسکریپت این گونه نیست و همه اعداد در یک فرمت ٦٤ بیتی (٨ بایت) و بر مبنای ١٠ ذخیره می شوند .

بنابراین در جاوا اسکریپت ، فقط یک نوع داده عددی داریم و برای تعریف آنها نیاز نیست نوع خاص ذکر شود . اعدد می توانند به صورت صحیح و یا اعشاری و ... نوشته و استفاده شوند . شکل کلی تعریف یک عدد در javascript به صورت زیر است :

Syntax	var نام متغیر = مقدار ; مثال ها : var MyNum = 14 ; var MyDem = 36.785 ;

تعریف اعداد بسیار بزرگ و یا بسیار کوچک :

برای تعریف اعداد بسیار بزرگ و یا بسیار کوچک ، می توانید از نشان علمی e استفاده نمایید . کاربرد آن را در مثال های زیر نمایش داده ایم :

Syntax	var MyNum = 123e5 ; معادل : ١٢٣٠٠٠٠٠ var MyDem = 123e-5 ; معادل : ٠.٠٠١٢٣

شی Boolean (مقدار true یا false)

شی (Boolean) در جاوا اسکریپت :

شی Boolean در جاوا اسکریپت ، همانند سایر زبان های برنامه نویسی دیگر می تواند دارای یکی از مقدار صحیح true و یا غلط false باشد .

از این شی برای تشخیص صحیح یا غلط بودن یک مقدار استفاده می شود .

شکل کلی تعریف و استفاده از یک شی Boolean در جاوا اسکریپت به صورت زیر است :

Syntax	var نام متغیر = new Boolean( ) ; مثال : var MyBool = new Boolean( ) ;

مقدار دهی اولیه یک شی Boolean :

پس از اینکه یک شی از نوع Boolean را تعریف کردید ، چنانچه مقداری را به آن نسبت ندهید ، به صورت پیش فرض دارای مقدار صحیح یا true خواهد بود . اما با نسبت دادن یکی از مقادیر زیر ، می توانید آن را منفی یا false نمایید :

- 0
- -0
- null
- ""
- false
- undefined
- NaN

نکته : توجه نمایید به کار بردن مقدار "false" اشتباه بوده و باعث مثبت شدن مقدار متغیر می شود . باید بدون " " استفاده شود .

متد شی Boolean :

در لیست زیر متدهای شی Boolean قرار دارد . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام متد	کاربرد
toString ( )	این متد مقدار شی Boolean را به متن string تبدیل کرده و آن را به خروجی بر می گرداند .
valueOf ( )	این متد مقدار اولیه شی Boolean را بر می گرداند .

شی Boolean - متد ( ) valueOf

شی (Boolean) - متد ( ) valueOf :

این متد مقدار یک شی یا متغیر Boolean را در خروجی بر می گرداند .

بر حسب مقدار متغیر Boolean خروجی این متد نیز متفاوت خواهد بود . اگر مقدار متغیر مثبت (true) باشد ، خروجی آن true و چنانچه مقدار آن منفی (false) باشد ، خروجیش false خواهد بود .

شکل کلی استفاده از این متد به صورت زیر است :

Syntax	نام متغیر.valueOf( ) ; مثل : MyBool.valueOf( ) ;

مثل : در مثال زیر دو متغیر از نوع Boolean را تعریف کرده و به هر کدام مقداری متفاوت داده ایم . سپس به وسیله متد ( ) valueOf ، مقدار آنها را در خروجی چاپ کرده ایم :

Example	

کد	<script type = " text/javascript " >
	var MyNum1 = new Boolean(1);
	var MyNum2 = new Boolean(0);
	document.write("مقدار متغیر ۱ = " + MyNum1.valueOf() + "
	");
	document.write("مقدار متغیر ۲ = " + MyNum2.valueOf());
	</script>

خروجی	مقدار متغیر ۱ = true

	مقدار متغیر ۲ = false

شی - (Boolean) متد ( ) toString :

این متد مقدار یک شی یا متغیر Boolean را به متن (String) تبدیل کرده و در خروجی بر می گرداند .

بر حسب مقدار متغیر Boolean خروجی این متد نیز متفاوت خواهد بود . اگر مقدار متغیر مثبت (true) باشد ، خروجی آن true و چنانچه مقدار آن منفی (false) باشد ، خروجیش false خواهد بود .

شکل کلی استفاده از این متد به صورت زیر است :

Syntax	نام متغیر.toString( ) ; مثل : MyBool.toString( ) ;

مثل : در مثال زیر دو متغیر از نوع Boolean را تعریف کرده و به هر کدام مقداری متفاوت داده ایم . سپس به وسیله متد ( ) toString ، مقدار آنها را در خروجی چاپ کرده ایم :

Example	

<script type = " text/javascript " >
var MyNum1 = new Boolean(1);
var MyNum2 = new Boolean(0);
document.write("مقدار متغیر ۱ = " + MyNum1.toString() + "
");
document.write("مقدار متغیر ۲ = " + MyNum2.toString());
</script>
	کد

مقدار متغیر ۱ = true مقدار متغیر ۲ = false	خروجی

شی RegExp - عملیات جستجو متن در جاوا اسکریپت

شی (RegExp) در جاوا اسکریپت:

یکی از مهمترین نیازها و کارهایی که ممکن است در یک صفحه وب بخواهید انجام دهید ، جستجو برای کلمات یا عبارت خاص در متن صفحه است . فرض کنید که در یک صفحه وب می خواهید به دنبال واژه CSS بپردازید و ببینید آیا این کلمه در صفحه وجود دارد یا خیر .

شی RegExp به طراح امکان ایجاد کدها و ساختارهایی را برای جستجو دنبال کلمات و عبارات در صفحات وب را می دهد .

RegExp مخفف عبارت Regular Expression است . RegExp شی است که یک الگو از کاراکترها را تعیین می کند . یک الگو ساده می تواند فقط یک کاراکتر باشد و الگوهای پیچیده تر ، شامل تعداد بیشتری کاراکتر هستند .

نتیجه گیری : شی RegExp برای انجام عملیات جستجو برای کلمات و یا کاراکترهای مورد نظر در متن یک صفحه استفاده می شود .

شکل کلی تعریف و استفاده از یک شی RegExp در جاوا اسکریپت به صورت زیر است :

Syntax	var patt = new RegExp (pattern , modifiers) ; یا به صورت ساده شده : var patt = /pattern/modifiers ;

اما کد بالا چه مفهومی داشته و طرز استفاده از آن به چه صورت است ؟ برای توضیح آن ، هر یک از کلمات موجود در کد را توضیح داده ام .

کلمه	توضیح
patt	یک نام دلخواه است که برای شی خودمان از نوع RegExp تعیین کرده ایم . مثل یه نام برای هر نوع متغیر دیگر .
new RegExp	نام متد یا تابعی است که یک نمونه جدید از شی RegExp را ایجاد می نماید .
pattern	pattern الگو کاراکترها ، یا همان عبارتی است که در متن می خواهیم به دنبال آن بگردیم . این الگو می توانید شامل یک کاراکتر بوده و یا یک کلمه و عبارت باشد .
modifiers	modifiers یک کاراکتر است که تعیین می کند عملیات جستجو به چه صورت انجام شود . مثلا چنانچه در این قسمت کاراکتر i را قرار دهیم ، عملیات جستجو به صورت case sensetive (حساس به حروف بزرگ و کوچک) انجام می شود .

جند مثال برای آشنایی با کارکرد RegExp : در این بخش سعی می کنیم با ارائه چند مثال کاربردی و کدهای آنها ، نحوه کار و استفاداه ار این شی را به شما آموزش دهیم .

مثل ۱ : جستجو برای یک عبارت یک کلمه خاص در یک متن .

در مثال اول یک متغیر متنی به نام Str1 تعریف کرده ایم که شامل یک متن است . به وسیله ساخت و به کار گیری یک شی RegExp در درون این متغیر به دنبال واژه Developer پرداخته ایم . کد مثال به دنبال این کلمه در متغیر Str1 پرداخته و سپس آن را در خروجی نمایش داده است :

کد	< script type="text/javascript" >
	var Str1 = "Welcome to Developer1.ir";
	var patt1 = /Developer/i;
	document.write(Str1.match(patt1));
	< /script >

خروجی	Developer

مثل ۲ : جستجو برای یک عبارت یک کلمه خاص در یک متن و یافتن موارد تکراری.

در مثال دوم یک متن طولانی تر را در متغیر خودمان قرار داده ایم . در این عبارت ، به دنباب کلمه is پرداخته ایم. این کلمه در متن چند بار تکرار شده است . بنابراین با به کار بردن یک Modifire از نوع g در شی RegExp ، باعث شده ایم تا شی نمونه های تکراری را یافته و همه آنها را نمایش دهد :

نکته مهم : توجه داشته باشید به دلیل اینکه از Modifire نوع i در این مثال استفاده نکرده ایم ، مورد اول تکرار Is ، به حساب نیامده و در خروجی نمایش داده نشده است . در مثل سوم Modifire نوع i را نیز به کار برده ایم . خروجی ها را با هم مقایسه نمایید .

کد	< script type="text/javascript" >
	var Str2 = "Is more than one is in this ?";
	var patt2 = /is/g;
	document.write(Str2.match(patt2));
	< /script >

خروجی	is,is

مثل‌شماره ۳ : استفاده همزمان از دو Modifire نوع i , g	

< script type="text/javascript" >
  var Str3 = "Is more than one is in this ?";
  var patt3 = /is/**gi**;
  document.write(Str3.match(patt3));
< /script >
	کد

Is,is,is	خروجی

شی Navigator در جاوا اسکریپت - (تشخیص نوع مرورگر)

شی مرورگر (navigator) در جاوا اسکریپت:

شی Navigator ، حاوی اطلاعاتی راجع به نوع مرورگر مورد استفاده کاربر است . این اطلاعات درباره مرورگری است ، که کاربر در آن لحظه در حال مشاهده صفحه با آن می باشد . این شی حاوی اطلاعاتی همچون نام مرورگر مورد استفاده ، نسخه (ورژن) ، زبان مورد استفاده مرورگر و ... می باشد . این اطلاعات در موارد مختلفی می تواند مورد استفاده قرار بگیرد ، که در ادامه به معرفی آنها خواهیم پرداخت .

این شی توسط واژه کلیدی navigator در سطح برنامه های اسکریپتی شناخته شده و حالت کلی استفاده آن به صورت زیر است :

Syntax	navigator.نام خاصیت

در ادامه به معرفی و تشریح خواص و رویدادهای مهم و پر کاربرد این شی در Java Script می پردازم .

تشخیص نوع مرورگر با استفاده از شی: navigator

یکی از مهمترین کاربردهای شی navigator در جاوا اسکریپت ، تشخیص نوع مرورگر مورد استفاده کاربر در هنگام مشاهده صفحه و نسخه (ورژن) آن است . این مسئله در چند مورد کاربرد دارد ، که به یکی از آنها اشاره می کنم .

مرورگر های وب و همچنین زبان های برنامه نویسی وب از جمله زبان اسکریپتی Java Script در طول زمان تغییرات زیادی داشته اند و ممکن است برخی از مرورگرها و یا یک ورژن خاص آنها از برخی از دستورات Java Script به درستی پشتیبانی نکنند . بنابراین یک طراح وب حرفه ای باید این مسئله را مد نظر داشته و برای آن تدابیر لازم را اتخاذ کند . برای مثال ممکن است یک اسکریپت در مرورگر IE ٦ به صورت صحیح اجرا شود ، ولی در مرورگر IE ٥ به صورت صحیح اجرا نشود و یا مثلا مرورگر FireFox از یکسری دستورات خاص پشتیبانی نکند ، در حالی که همان کد در IE به صورت صحیح اجرا شود . موارد اختلاف کارایی زیادی بین مرورگرها وجود دارد و باید همواره این نکته را به خاطر داشته باشید .

در این صورت باید طراح در زمان نوشتن اسکریپت هایی که امکان بروز خطا ، عدم پشتیبانی و یا پشتیبانی نادرست در آنها وجود دارد ، دستورات متناسب با هر نوع مرورگر خاص را در اسکریپت تعیین کرده و با استفاده ازدستورات شرطی و یا دستور Switch ، پس از تشخیص نوع مرورگر و ورژن آن ، دستورات مرتبط با آن نوع مرورگر خاص اجرا شود . در این حالت ، سازگاری برنامه با انواع دستورات اسکریپتی تضمین می شود .

خاصیت appName و appVersion شی Navigator ، به ترتیب نام و ورژن مرورگر مورد استفاده را مشخص می کنند . در بخش زیر خواص مهم شی Navigator را نشان داده ایم .

خواص مهم شی navigator :

در لیست زیر خواص مهم و پر کاربرد شی Navigator قرار دارد . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام خاصیت	شرح
appCodeName	این خاصیت نام تعیین شده ویژه (اسم رمز) مربوط به نسخه مرورگر مورد استفاده را بر می گرداند . مثال : مثلا نام ویژه نسخه Mozilla، IE7 است .
appName	این خاصیت نام کامل مرورگر مورد استفاده را بر می گرداند .
appVersion	این خاصیت پلتفرم و ورژن مرورگر مورد استفاده را بر می گرداند .
browserLanguage	این خاصیت زبان مرورگر مورد استفاده را بر می گرداند .
cookieEnabled	این خاصیت یک مقدار Boolen را بر می گرداند ، که مشخص می کند آیا مرورگر مورد استفاده از قابلیت cookie پشتیانی می کند ، یا خیر .
cpuClass	این خاصیت کلاس CPU مورد استفاده در کامپیوتر کاربر استفاده کننده از مرورگر را بر می گرداند .
onLine	این خاصیت یک مقدار Boolen را بر می گرداند ، که مشخص می کند آیا مرورگر در حالت OnLine است یا خیر .
platform	این خاصیت پلتفرم سیستم عامل کامپیوتر کاربر استفاده کننده از مرورگر را بر می گرداند .
SystemLanguage	این خاصیت زبان سیستم عامل کامپیوتر کاربر استفاده کننده از مرورگر را بر می گرداند .

شی Navigator - خاصیت appName

این خاصیت ، نام کامل مرورگر مورد استفاده را بر می گرداند . همانطور که در قسمت معرفی شی Navigator اشاره کردم ، مهمترین کاربرد این خاصیت تشخیص نوع مرورگر است . تشخیص نوع مرورگر می تواند کاربردهای زیادی داشته باشد ، که به برخی از آنها اشاره خواهم کرد .

مرورگر های وب و همچنین زبان های برنامه نویسی وب از جمله زبان اسکریپتی Java Script در طول زمان تغییرات زیادی داشته اند و ممکن است برخی از مرورگرها و یا یک ورژن خاص آنها از برخی از دستورات Java Script به درستی پشتیبانی نکنند . بنابراین یک طراح وب حرفه ای باید این مسئله را مد نظر داشته و برای آن تدابیر لازم را اتخاذ کند . برای مثال ممکن است یک اسکریپت در مرورگر IE ٦ به صورت صحیح اجرا شود ، ولی در مرورگر IE ٥ به صورت صحیح اجرا نشود و یا مثلا مرورگر FireFox از یکسری دستورات خاص پشتیبانی نکند ، در حالی که همان کد در IE به صورت صحیح اجرا شود . **موارد اختلاف** کارایی زیادی بین مرورگرها وجود دارد و باید همواره این نکته را به خاطر داشته باشید . در این صورت باید طراح در زمان نوشتن اسکریپت هایی که امکان بروز خطا ، عدم پشتیبانی و یا پشتیبانی نادرست در آنها وجود دارد ، دستورات متناسب با هر نوع مرورگر خاص را در اسکریپت تعیین کرده و با استفاده از دستورات شرطی و یا **دستور** Switch ، پس از تشخیص نوع مرورگر و ورژن آن ، دستورات مرتبط با آن نوع مرورگر خاص اجرا شود . در این حالت ، سازگاری برنامه با انواع دستورات اسکریپتی تضمین می شود .

از دیگر کاربردهای تشخیص نوع مرورگر می توان به نصب کردن برنامه های کاربردی (Plug In) **مثل** Flash Palyer یا Real Player بر روی مرورگر اشاره کرد . ممکن است یک سایت دانلود این برنامه ها ، نسخه های متفاوتی از یک نرم افزار را برای نصب بر روی مرورگر های مختلف تولید کرده باشد ، که برای هر مرورگر باید نسخه متناسب را نصب کرد . در این صورت سایت مذکور می تواند با تشخیص نوع مرورگر شما ، نسخه مخصوص به آن مرورگر را برای نصب در اختیار شما قرار بدهد ، تا در این صورت مشکل عدم کارایی نرم افزار پیش نیاید و کاربردهای دیگر .

شکل کلی استفاده از این خاصیت به صورت زیر است :

Syntax	navigator.appName

نکته : خروجی این خاصیت یک مقدار رشته ای (String) **است** .

مثل ١ : در مثال زیر با استفاده از خاصیت appName ، نام کامل مرورگر مورد استفاده کاربر را نشان داده ایم . نام نمایش داده شده در خروجی بستگی به مرورگری دارد که شما هم اکنون ، در حال مشاهده صفحه با آن هستید (به دلیل اینکه خروجی های مثال ها به صورت اکتیو توسط مرورگر تولید می شود و از قبل طراحی شده نیست) :

Example

<script type ="text/javascript"> document.write ("Full name of your browser is " + navigator.appName) ; </script>	کد
Full name of your browser is **Netscape**	خروجی

مثل ۲ : کاربرد تشخیص نوع مرورگر :

در مثال زیر با استفاده از یک اسکریپت ، ابتدا نوع مرورگر را تشخیص داده و بر حسب مقدار آن ، در یک دستور Switch ، کد قابل اجرا بر ۳ مرورگر Oprer , Firefox و IE را تعیین کرده ایم . در این اسکریپت ابتدا برنامه نوع مرورگر مورد استفاده کاربر را توسط خاصیت appName شی Navigator تشخیص داده و بر حسب آن دستور مناسب را اجرا خواهد کرد . نتیجه خروجی بستگی به مرورگری دارد که شما هم اکنون ، در حال مشاهده صفحه با آن هستید .

راهنمایی : خاصیت systemLanguge یکی از خواص شی Navoigator است ، که در ادامه آن را به طور کامل تر توضیح می دهم . این خاصیت زبان سیستم عامل کامپیوتر اجرا کننده مرورگر را بر می گرداند . اما این خاصیت فقط توسط مرورگر Intenet Explorer پشتیبانی شده و مرورگرهای Opera و FireFox از آن پشتیبانی نمی کنند . در اسکریپت زیر ، هدف این است که فقط زمانی که کاربر با مرورگر IE صفحه را مشاهده می کند ، زبان سیستم عامل را ببیند و در صورتی که از مرورگر های Firefox یا Opera استفاده می کند ، پیام عدم پشتیبانی مرورگر از آن دستور را اعلام کند .

راهنمایی ۲ : نام مرورگر Firefox توسط خاصیت appName واژه Netscape است .

Example	
<script type ="text/javascript"> switch (navigator.appName) { case " Microsoft Internet Explorer " : document.write (navigator.systemLanguage) ; break; case " Netscape " : document.write (" Your Browser Dosen't Support navigator.systemLanguage Property ! ") ; break ; case " Opera " : document.write (" Your Browser Dosen't Support navigator.systemLanguage	کد

Property ! ") ; } </script>	
Your Browser Dosen’t Support navigator.systemLanguage Property !	خروجی

شی Navigator - خاصیت های appCodeName و appVersion

خاصیت appCodeName شی Navigator -نام ویژه مرورگر :

این خاصیت نام ویژه (نام کد) مرورگر مورد استفاده را بر می گرداند .
نام کد ، نامی خاص است که از سوی شرکت سازنده برای نام گذاری و مشخص کردن یک ورژن از مرورگر تعیین می شود .
این نام بیشتر یک اصطلاح عامیانه است و کاربرد خاصی ندارد . مثلا نام ویژه مرورگرهای Mozilla , Internet Explorer است . شکل کلی استفاده از این خاصیت به صورت زیر است :

Syntax	navigator.appCodeName

نکته : خروجی این خاصیت یک مقدار رشته ای (String) است .

مثال ۱ : در مثال زیر با استفاده از خاصیت appCodeName ، نام کد مرورگر مورد استفاده کاربر را نشان داده ایم . نام نمایش داده شده در خروجی بستگی به مرورگری دارد که شما هم اکنون ، در حال مشاهده صفحه با آن هستید (به دلیل اینکه خروجی های مثال ها به صورت اکتیو توسط مرورگر تولید می شود و از قبل طراحی شده نیست) :

Example	
<script type ="text/javascript"> document.write ("Code name of your browser is " + navigator.appCodeName) ; </script>	کد
Code name of your browser is **Mozilla**	خروجی

خاصیت appVersion شی Navigator - ورژن مرورگر:

این خاصیت ، ورژن و پلتفرم کامل مرورگر مورد استفاده را بر می گرداند . پلتفرم شامل اطلاعاتی از جمله سیست عامل سازگار با مرورگر ، زبان مورد استفاده و ... می باشد .
همانند بخش قبل که در مورد خاصیت appName (نام مرورگر) ، اشاره کردم ، تعیین نوع و ورژن مرورگر می تواند کاربرهای مختلفی داشته باشد . در ادامه به ۲ خاصیت این روش اشاره می کنم .
مرورگر های وب و همچنین زبان های برنامه نویسی وب از جمله زبان اسکریپتی Java Script در طول زمان تغییرات

زیادی داشته اند و ممکن است برخی از مرورگرها و یا یک ورژن خاص آنها از برخی از دستورات Java Script به درستی پشتیبانی نکنند . بنابراین یک طراح وب حرفه ای باید این مسئله را مد نظر داشته و برای آن تدابیر لازم را اتخاذ کند . برای مثال ممکن است یک اسکریپت در مرورگر IE ٦ به صورت صحیح اجرا شود ، ولی در مرورگر IE ٥ به صورت صحیح اجرا نشود و یا مثلا مرورگر FireFox از یکسری دستورات خاص پشتیبانی نکند ، در حالی که همان کد در IE به صورت صحیح اجرا شود . موارد اختلاف کارایی زیادی بین مرورگرها وجود دارد و باید همواره این نکته را به خاطر داشته باشید . در این صورت باید طراح در زمان نوشتن اسکریپت هایی که امکان بروز خطا ، عدم پشتیبانی و یا پشتیبانی نادرست در آنها وجود دارد ، دستورات متناسب با هر نوع مرورگر خاص را در اسکریپت تعیین کرده و با استفاده از دستورات شرطی و یا دستور Switch ، پس از تشخیص نوع مرورگر و ورژن آن ، دستورات مرتبط با آن نوع مرورگر خاص اجرا شود . در این حالت ، سازگاری برنامه با انواع دستورات اسکریپتی تضمین می شود .

از دیگر کاربردهای تشخیص نوع و ورژن مرورگر می توان به نصب کردن برنامه های کاربردی (Plug In) مثل Flash Palyer یا Real Player بر روی مرورگر اشاره کرد . ممکن است یک سایت دانلود این برنامه ها ، نسخه های متفاوتی از یک نرم افزار را برای نصب بر روی مرورگر های مختلف تولید کرده باشد ، که برای هر مرورگر باید نسخه متناسب را نصب کرد . در این صورت سایت مذکور می تواند با تشخیص نوع مرورگر شما ، نسخه مخصوص به آن مرورگر را برای نصب در اختیار شما قرار بدهد ، تا در این صورت مشکل عدم کارایی نرم افزار پیش نیاید و کاربردهای دیگر .

شکل کلی استفاده از این خاصیت به صورت زیر است :

Syntax	navigator.appVersion

نکته : خروجی این خاصیت یک مقدار رشته ای (String) است .

مثل ۱ : در مثال زیر با استفاده از خاصیت appVersion ، ورژن کامل مرورگر مورد استفاده کاربر را نشان داده ایم . ورژن نمایش داده شده در خروجی بستگی به مرورگری دارد که شما هم اکنون ، در حال مشاهده صفحه با آن هستید (به دلیل اینکه خروجی های مثال ها به صورت اکتیو توسط مرورگر تولید می شود و از قبل طراحی شده نیست) :

Example	
<script type ="text/javascript"> document.write ("Full Version of your browser is " + navigator.appVersion) ; </script>	کد
Full Version of your browser is **5.0 (Windows NT 6.1) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.52 Safari/537.17**	خروجی

شی Navigator - خاصیت cookieEnabled

خاصیت cookieEnabled :

این خاصیت یک مقدار Boolen (نوع True یا False) را بر می گرداند ، که تعیین می کند آیا قابلیت ایجاد و خواندن کوکی (cookie) در مرورگر شما فعال است یا خیر .

کوکی (cookie) متغیری است که بر روی کامپیوتر کاربر ذخیره شده و هر بار که کاربر آن صفحه را باز کند ، مقدار آن توسط مرورگر خوانده شده و می تواند مورد استفاده قرار بگیرد . از کوکی ها معمولا برای ثبت و ذخیره کردن اطلاعاتی همچون نام کاربری و رمز ورود به سایت ها و سایر اطلاعات شخصی کاربران استفاده می شود . در بخش های بعدی به طور کامل تر به بحث درباره نحوه ایجاد و خواندن کوکی ها خواهیم پرداخت .

شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	navigator.cookieEnabled

نکته : خروجی این خاصیت یک مقدار Boolen (نوع True یا False) **است** . اگر مقدار بازگشتی True **باشد ، مرورگر** قابلیت ایجاد و خواندن کوکی را داراست و در صورت برگشت مقدار False ، آن قابلیت را نداشته و یا قابلیت غیر فعال شده **است** .

مثل ۱ : در مثال زیر با استفاده از خاصیت cookiedEnabled ، فعال بودن قابلیت ایجاد و خواندن کوکی در مرورگر شما را در خروجی نمایش داده ایم . این قابلیت در مرورگر شما فعال است یا خیر ؟!

Example

کد

```
<script type ="text/javascript">
 if ( navigator.cookieEnabled == true )
  document.write ( " cookie is < b > Enabled < /b > in your browser " ) ;
 else if ( navigator.cookieEnabled == false )
  document.write ( " cookie is < b > Not Enabled < /b > in your browser " );
</script>
```

خروجی

cookie is **Enabled** in your browser

کاربرد خاصیت cookieEnabled :

مهمترین کاربرد این خاصیت این است که شما می توانید ابتدا چک کنید که آیا مرورگر کاربر قابلیت ایجاد و خواندن کوکی را دارد یا خیر . سپس در صورت فعال بودن ، دستورات اسکریپتی مربوط به خواندن و ایجاد کوکی اجرا شود . در این صورت قابلیت اطمینان و کارایی برنامه بالاتر می رود .

در مثال زیر نحوه استفاده از این کاربرد را در عمل نشان داده ایم :

مثل ۱ : در مثال زیر با استفاده از خاصیت cookiedEnabled ، ابتدا فعال ایجاد و خواندن کوکی را در مرورگر شما بررسی کرده ایم . سپس با توجه به خروجی ، در صورت فعال بودن این قابلیت یکسری دستور ایجاد کوکی اجرا خواهد شد و در صورت فعال نبودن آن پیام عدم پشتیبانی مرورگر شما از کوکی اعلام خواهد شد . در بخش های بعد نحوه ایجاد و خواندن کوکی را به طور کامل شرح خواهیم داد :

Example

کد

```
<script type ="text/javascript">
 if ( navigator.cookieEnabled == true )
 {
   function setCookie(c_name,value,expiredays)
   {
    var exdate=new Date()exdate.setDate(exdate.getDate()+expiredays)
    document.cookie=c_name+ "=" +escape(value)+
    ((expiredays==null) ? "" : ";expires="+exdate.toGMTString())
  }
 }
 else if ( navigator.cookieEnabled == false )
  document.write ( " cookie is < b > Not Enabled < /b > in your browser " );
</script>
```

شی Navigator - خواص systemLanguage و browserLanguage

خاصیت browserLanguage - زبان مرورگر :

این خاصیت زبان پیش فرض و مورد استفاده مرورگر را بر می گرداند .

برای مثال مقدار این خاصیت در زبان انگلیسی (آمریکا) en-us و برای زبان فارسی fa **است** .

شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	navigator.browserLanguage

نکته : خروجی این خاصیت یک مقدار رشته ای (String) **است** .

مثل ۱ : در مثال زیر با استفاده از خاصیت browserLanguage ، زبان مورد استفاده مرورگر را به کاربر نمایش داده ایم . مقدار خروجی این مثال بستگی به زبان مورد استفاده مرور گر شما دارد :

Example	
<script type ="text/javascript"> document.write ("Default language of your browser is " + navigator.browserLanguage) ; </script>	کد
	خروجی

مثل ۲ : در مثال زیر با استفاده از خاصیت browserLanguage ، ابتدا زبان مورد استفاده مرورگر را تشخیص داده **و بر** حسب آن پیام مناسب با آن زبان را در خروجی نشان داده ایم . به این صورت که اگر زبان مورد استفاده مرور گر انگلیسی باشد ، پیام به زبان انگلیسی و در صورتی که زبان آن فارسی باشد ، پیام به زبان فارسی چاپ خواهد شد :

Example	
<script type ="text/javascript"> if (navigator.browserLanguage == "en-us") document.write (" welcome to DeveloperStudio ") ; else if (navigator.browserLanguage == " fa ") document.write (" خوش آمدید به سایت DeveloperStudio . ");	کد

</script>	
	خروجی

خاصیت systemLanguage – زبان سیستم عامل مرورگر :

این خاصیت زبان پیش فرض و مورد استفاده سیستم عامل اجرا کننده مرورگر را بر می گرداند .

برای مثال مقدار این خاصیت در زبان انگلیسی (آمریکا) en-us و برای زبان فارسی fa **است** .

شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	navigator.systemLanguage

نکته : خروجی این خاصیت یک مقدار رشته ای (String) **است** .

مثل ۱ : در مثال زیر با استفاده از خاصیت systemLanguage ، زبان مورد استفاده سیستم عامل مرورگر را به کاربر نمایش داده ایم . مقدار خروجی این مثال بستگی به زبان مورد استفاده سیستم عامل مرور گر شما دارد :

Example	
<script type ="text/javascript"> document.write ("Default oprating system language of your browser is " + navigator.systemLanguage) ; </script>	کد
	خروجی

شی Navigator - خواص online , platform , cpuClass

خاصیت onLine وضعیت مرورگر:

این خاصیت یک مقدار Boolen (نوع true یا false) را برمی گرداند ، که تعیین می کند که آیا سیستم عامل مرورگر به اینترنت متصل است و یا در حالت offline قرار دارد .

حالت offline : مرورگرهای جدید از جمله ورژن های IE ۴ و بالاتر دارای قابلیتی به نام work offline هستند . بر حسب این قابلیت کاربر می تواند بدون اتصال به شبکه اینترنت ، صفحات وبی که بر روی هارد دیسک کامپیوتر خود ذخیره کرده و یا در history مرورگر قرار دارند ، را باز و مشاهده کرده و بین صفحات حرکت کند . در این حالت می گویند که سیستم در حالت offline قرار دارد .

در مرورگر IE شما می توانید از طریق منوی Standard و زیر منوی Tools ، گزینه work offline را انتخاب کنید تا مرورگر در حالت offline قرار بگیرد . در حالت offline چنانچه بخواهید صفحه جدیدی که ذخیره نشده را باز کنید ، مرورگر به شما پیام می دهد که باید به اینترنت متصل شوید .

شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	navigator.onLine

نکته : خروجی این خاصیت یک مقدار Boolen (نوع true یا false) است . مفهوم خروجی این خاصیت یکی از ۲ حالت زیر را می تواند داشته باشد :

خروجی های خاصیت onLine		
خروجی	وضعیت مرورگر	شرح
true	online	مرورگر به اینترنت متصل است .
false	offline	مرورگر در حالت work offline قرار داشته و به اینترنت متصل نیست .

مثال ۱ : در مثال زیر با استفاده از خاصیت onLine ، وضعیت کاری مرورگر را به کاربر نمایش داده ایم . مقدار خروجی این مثال بستگی به وضعیت ارتباطی مرورگر مورد استفاده شما دارد :

Example	

کد	<script type ="text/javascript">
	  if (navigator.onLine == true)
	  document.write (" You navigator is in **online** mode ")
	  else
	  document.write(" You navigator is in **offline** mode ")
	</script>

خروجی	You navigator is in **offline** mode

خاصیت cpuClass نوع CPU کامپیوتر مرورگر :

این خاصیت کلاس CPU کامپیوتر اجرا کننده مرورگر را بر می گرداند .

شکل کلی استفاده از این خاصیت به شرح زیر است :

Syntax	navigator.cpuClass

نکته : خروجی این خاصیت یک مقدار رشته ای (String) **است** .

به طور کلی ۲ کلاس برای CPU ها داریم ، که خروجی این خاصیت می تواند یکی از ۲ مقدار زیر باشد :

خروجی های خاصیت cpuClass		
خروجی	کلاس CPU	شرح
x86	X86	کلاس CPU کامپیوتر اجرا کننده مرورگر X86 **است** .
Alpha	Alpha	کلاس CPU کامپیوتر اجرا کننده مرورگر Alpha **است** .

مثل ۱ : در مثال زیر با استفاده از خاصیت cpuClass ، کلاس کامپیوتر شما را نمایش داده ایم . کلاس CPU کامپیوتر شما چیشت ؟؟!

Example	
کد	<script type ="text/javascript"> document.write (navigator.cpuClass) </script>
خروجی	undefined

اشیای پیشرفته جاوا اسکریپت - شی Window

شی window :

شی window بالاترین شی در رده بندی عناصر صفحات HTML در مدل DOM است . هر پنجره مرورگر در واقع یک نسخه از شی window است . این شی به ازای هر نمونه از تگ های < body > یا < frameset > ساخته می شود . در لیست زیر مجموعه خواص و رویدادهای مهم و پرکاربرد شی window آمده است . برای دریافت اطلاعات بیشتر راجع به هر کدام بر روی نام آن کلیک کنید :

خواص شی window :

نام خاصیت	شرح
closed	مشخص می کند که آیا پنجره مورد نظر بسته است یا خیر .
defaultStatus	این خاصیت برای تعیین متن نمایش داده شده در نوار statusbar پنجره به کار می رود .
length	این خاصیت تعداد قاب ها موجود در پنجره را بر می گرداند .
name	این خاصیت برای تعیین یک نام برای پنجره مورد نظر استفاده می شود .
opener	این خاصیت به پنجره ای که باعث باز شدن پنجره جاری شده است ، اشاره می کند .
self	این خاصیت برای اشاره به پنجره ای که هم اکنون در آن هستیم به کار می رود .
status	این خاصیت برای تعیین متن نمایش داده شده در نوار statsbar به کار می رود .
top	این خاصیت به پنجره مادر پنجره جاری اشاره می کند .

خواص شی window

خاصیت defaultStatus :

این خاصیت متنی که در نوار Statusbar پنجره مروگر نمایش داده می شود را تعیین و نگهداری می کند . نوار Statusbar در پایین پنجره مرورگر قرار دارد و معمولا برای اطلاع رسانی عملکرد مرورگر به کاربر ، از آن استفاده می شود .

Syntax	window.defaultStatus = " متن مورد نظر " * window = نام پنجره مورد نظر

مثل : در مثال زیر با استفاده از یک دستور ساده متن نوار Statusbar مرورگر را در هنگام لود صفحه به مقدار مورد نظر تغییر داده ایم . به پایین پنجره مرورگر دقت کنید . این مقدار را در آینده دوباره می توان تغییر داد :

نکته : خاصیت self در مثال زیر به پنجره جاری که هم اکنون در آن هستیم اشاره می کند .

Example

کد

```
<script type ="text/javascript">
  window.self.defaultStatus = "www.DeveloperStudio.ir :: An Investigation for Development" ;
</script>
```

مثل : در مثال زیر یک کادر متن (TextBox) ساده و یک دکمه فرمان را برای تعیین متن نوار StatusBar بر روی فرم قرار داده ایم . نحوه عمکرد این مثال به این صورت است که کاربر بایستی متنی را در درون کادر متن وارد کرده و سپس بر روی دکمه فرمان change defaultStatus کلیک نماید . در این صورت تابع show_msg اجراشده و متن نوار StatusBar را به متن وارد شده از کاربر تغییر خواهد داد :

نکته : خاصیت self در مثال زیر به پنجره جاری که هم اکنون در آن هستیم اشاره می کند .

Example

کد

```
<script type ="text/javascript">
<input type="text" id="TxtMsg" size="25" />
<input type="button" id="BtnMsg" value="change defaultStatus"
onclick="show_msg( )" />

 function show_msg ( )
  {
    TxtMsg = document.getElementById ( "TxtMsg" );
```

var Msg = TxtMsg.value ; window.self.defaultStatus = Msg ; } </script>	
	خروجی

خواص شی window

خاصیت length :

این خاصیت تعداد قاب ها یا frame های موجود در پنجره را نمایش می دهد . همانطور که در HTML آموختید ، قاب یا frame با تگ < iframe > ایجاد شده و هر قاب می تواند یک صفحه را در درون خود جای دهد . در حالت عادی تعداد frame های موجود در یک صفحه صفر است .

Syntax	window.length * window = نام پنجره مورد نظر

نکته : نکته جالبی که در هنگام کار با این خاصیت با آن روبرو شدم این بود که چنانچه دستور window.length را در یک script به کار ببریم ، این دستور فقط تعداد frame هایی که با تگ < iframe > قبل از آن script ایجاد شده اند را نشان می دهد و frame های بعد از خود را نمی شناسد . برای درک بهتر به دو مثال این صفحه دقت کنید . در مثال اول چون هیچ frame قبل از script آن ایجاد نشده است ، مقدار خروجی دستور window.length صفر است . اما در مثال دوم قبل از script آن یک frame ایجاد کرده ایم و می بینیم که خروجی script در این حالت ۱ است .

مثال ۱ : در مثال زیر با استفاده از یک scriptساده ، تعداد frame های صفحه را در خروجی چاپ کرده ایم . همانطور که میبینید ، به دلیل اینکه قبل از script این مثال هیچ frame ی ایجاد نشده است ، مقدار خروجی صفر است .

Example	
<script type ="text/javascript"> document.write (window.length) ; </script>	کد
0	خروجی

مثال ۲ : در مثال زیر ابتدا یک قاب یا frame را ابتدا ایجاد کرده ایم و مجددا تعداد frame های موجود در صفحه را در خروجی چاپ کرده ایم . در این حالت می بینیم که خروجی مثال ۱ است :

Example	
<iframe src="Dom_Introduce.aspx" style="width: 500px; height: 200px "></iframe>	کد

```
<script type ="text/javascript">
  document.write ( "<br /> " + window.length ) ;
</script>
```

1

خواص شی window

خاصیت name :

این خاصیت تعیین کننده و نگهدارنده نام پنجره مورد نظر است . نام یک پنجره باید از نوع متن یا text تعیین شود . نام پنجره برای دستیابی به آن پنجره در script ها استفاده می شود و ربطی به نام فایل صفحه ندارد و مقدار آن توسط برنامه نویس می تواند تعیین گردد . به صورت پیش فرض پنجره ها بدون نام هستند ، یعنی نامی برای آنها تعیین نشده است و چنانچه از سوی برنامه نویس یا طراح نیز نامی برای آن در نظر گرفته نشود ، مقدار آن خالی خواهد ماند .

Syntax	window.name = " نام مورد نظر " ;

مثل : در مثال زیر توسط یک اسکریپت ساده برای پنجره فعلی که در درون آن هستیم ، یک نام را تعیین کرده و سپس مقدار آن را در خروجی چاپ کرده ایم :

Example	
<script type ="text/javascript"> window.name = "MyWindow" ; document.write (window.name) ; </script>	کد
MyWindow	خروجی

خاصیت opener :

این خاصیت به پنجره ای که موجب ایجاد یا باز شدن پنجره فعلی شده است ، اشاره می کند . صفحه ای که به یک پنجره لینک داده است ، در صورت کلیک بر روی آن لینک و رفتن به صفحه جدید ، به عنوان باز کننده یا opener صفحه جدید ، محسوب می شود .

Syntax	window.opener ;

خاصیت self :

این خاصیت به پنجره ای که هم اکنون در آن هستیم ، اشاره می کند . این خاصیت نیز باید با یک خاصیت یا متد دوم برای آگاهی از خواص پنجره فعلی استفاده شود و به تنهایي کارایی چندانی ندارد . به کار بردن این خاصیت برای اشاره به پنجره فعلی در بیشتر موارد ضروری نیست و صرفا ذکر واژه window به تنهایي کافی است اما برای اطمینان بیشتر ، بهتر است به کار رود .

Syntax	window.self ;

مثل : در مثال زیر توسط یک اسکریپت ساده ابتدا به پنجره جاری که درون آن هستیم اشاره کرده و سپس آدرس کامل آن را توسط خاصیت location در خروجی چاپ کرده ایم :

Example	
<script type ="text/javascript"> document.write (window.self.location) ; </script>	کد
http://localhost:6094/Developer%20v.4/JavaScript/Window/name.aspx	خرو

خواص شی window

خاصیت status :

این خاصیت متنی که در نوار Statusbar پنجره مروگر نمایش داده می شود را تعیین و نگهداری می کند . نوار Statusbar در پایین پنجره مرورگر قرار دارد و معمولا برای اطلاع رسانی عملکرد مرورگر به کاربر ، از آن استفاده می شود .
عملکرد این خاصیت کاملا شبیه خاصیت defaultStatus است ، با این تفاوت که خاصیت defaultStatus معمولا برای تعیین یک متن پیش فرض برای statusBar به کار می رود و در هنگام لود شدن صفحه آن متن در statusbar نمایش داده می شود . ولی متنی که برای خاصیت status تعیین می شود در هنگام لود صفحه و اجرای تازه آن ، در statusbar نمایش داده نمی شود .

Syntax	window.status = " متن مورد ظر " * window = نام پنجره مورد ظر

مثل : در مثال زیر یک کادر متن (TextBox) ساده و یک دکمه فرمان را برای تعیین متن نوار StatusBar بر روی فرم قرار داده ایم . نحوه عمکرد این مثال به این صورت است که کاربر بایستی متنی را در درون کادر متن وارد کرده و سپس بر روی دکمه فرمان change statusbar Text کلیک نماید . در این صورت تابع show_msg اجراشده و متن نوار StatusBar را به متن وارد شده از کاربر تغییر خواهد داد :
نکته : خاصیت self در مثال زیر به پنجره جاری که هم اکنون در آن هستیم اشاره می کند .

Example

کد

```
<script type ="text/javascript">
<input type="text" id="TxtMsg" size="25" />
<input type="button" id="BtnMsg" value="change statusbar Text"
onclick="show_msg( )" />

 function show_msg ( )
  {
    TxtMsg = document.getElementById ( "TxtMsg" );
    var Msg = TxtMsg.value ;
    window.self.status = Msg ;
  }
```

</script>	
	خروجی

خاصیت top :

این خاصیت به بالاترین پنجره یا پنجره مادر پنجره جاری ، اشاره می کند .

Syntax	window.top

شی History (مشخصات صفحات مرورشده)

شی (History) در جاوا اسکریپت:

شی History به طور اتوماتیک با باز شدن یک صفحه ، توسط موتور جاوا اسکریپت مرورگر (Java Script runtime engine) ایجاد می شود . این شی حاوی اطلاعاتی درباره URL هایی که کاربر توسط مرورگر مشاهده کرده است ، می باشد . به عبارت دیگر آدرس صفحاتی که کاربر توسط مرورگر مشاهده می کند ، همانند یک آرایه در شی History ذخیره می شود .

این اطلاعات درباره صفحه قبلی و یا صفحات قبلی مشاهده شده ، صفحه بعدی یا صفحات بعدی مشاهده شده (البته در صورتی که کاربر از مسیر رفته به عقب بر گشته باشد) می باشد . از این شی برای دسترسی به صفحات قبلی و یا بعدی مرور شده استفاده می شود .

این شی یکی از زیر مجموعه های شی window است و باید به شکل کلی زیر به کار رود . البته بدون به کار بردن واژه window ، قابلیت دسترسی مستقیم به شی نیز وجود دارد :

Syntax	window.history.نام خاصیت یا history.نام خاصیت

در ادامه به معرفی و تشریح خواص و رویدادهای مهم و پر کاربرد این شی در Java Script می پردازم .

خواص مهم شی History :

در لیست زیر تنها خاصیت شی History قرار دارد . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام خاصیت	شرح
length	این خاصیت تعداد صفحاتی که URL و مشخصات آنها در شی History ذخیره شده است را مشخص می کند .

رویداد های مهم شی History :

در لیست زیر رویدادهای شی History **قرار دارد** . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام رویداد	کاربرد
back ( )	این متد صفحه قبلی مشاهده شده که آدرس آن در شی History ذخیره شده ، را باز می کند .
forward ( )	این متد صفحه بعدی مشاهده شده که آدرس آن در شی History ذخیره شده ، را باز می کند .
go ( )	این متد صفحه ای که شماره اندیس یا URL آن در پرانتز جلوی آن تعیین شده را باز می کند .

شی History - خاصیت length

این خاصیت ، تعداد صفحاتی که **آدرس** URL آنها در شی History ذخیره شده است را بر می گرداند .
هنگامی که یک پنجره را باز می کنید ، مقدار این خاصیت ۰ است ، یعنی مقدار آن از صفر شروع می شود . با حرکت در بین صفحات ، به ازای هر صفحه جدید باز شده ، یک واحد به این خاصیت اضافه می شود .
همانطور که در بخش معرفی شی History گفتم ، آدرس صفحاتی که توسط مرورگر مشاهده می کنید ، به صورت یک آرایه در شی History ذخیره می شود . به عبارت دیگر می توان گفت اندازه این خاصیت برابر با اعضا آرایه فوق است .

شکل کلی استفاده از این خاصیت به صورت زیر است :

Syntax	history.length

نکته ۱ : خروجی این خاصیت یک مقدار عددی (Integer) **است** .
نکته ۲ : مقدار خاصیت length **در مرورگر** Firefox **از** ۱ شروع می شود .

مثل ۱ : در مثال زیر با استفاده از خاصیت length ، تعداد صفحاتی که آدرس (URL) آنها در شی History **مرورگرشما قرار دارد را در** خروجی نشان داده ایم . (خروجی مثال به صورت اکتیو توسط مرورگر تولید شده و از پیش طراحی شده نیست) :

Example	
<script type ="text/javascript"> document.write (history.length) ; </script>	کد
1	خروجی

شی History - متدهای back , forward

متد back شی History :

این متد ، صفحه قبلی مشاهده شده در مرورگر را باز می کند . آدرس (URL) این صفحه در عنصر قبلی ذخیره شده در شی History قرار دارد .

عملکرد این متد همانند این است که دکمه back مرورگر را کلیک کرده و یا متد ( ) go شی History را با مقدار ۱- اجرا کنید .

شکل کلی استفاده از این متد به صورت زیر است :

Syntax	history.back ( )

مثال ۱ : در مثال زیر از متد back در تابع backward استفاده کرده ایم . این تابع با کلیک بر روی دکمه بازگشت به صفحه قبلی فراخوانی شده و مرورگر را به صفحه قبلی مشاهده شده می برد . برای رفتن به صفحه قبل بر روی دکمه فرمان بازگشت به صفحه قبلی در قسمت خروجی کلیک کنید :

Example	

<script type = " text/javascript " >
function backward ( )
{
history.back ( ) ;
}
</script>
 	
< input type = " button " name = " PrPage " value = "بازگشت به صفحه قبل"
onclick = " backward ( ) " />
	کد

	خروجی

متد forward شی History :

این متد ، صفحه بعدی مشاهده شده در مرورگر را باز می کند . آدرس (URL) این صفحه در عنصر بعدی ذخیره شده در شی History قرار دارد . البته این متد زمانی اجرا می شود که کاربر از مسیر رفته به عقب برگشته باشد . به عبارت دیگر آدرس صفحه ای در عنصر بعدی حافظه شی History ذخیره شده باشد . در غیر این صورت نیز با فراخوانی این متد خطایی رخ نداده و مرورگر در همان صفحه باقی می ماند .

عملکرد این متد همانند این است که کاربر دکمه forward مرورگر را کلیک کرده و یا متد ( ) go شی History را با مقدار ۱+ اجرا کنید .

شکل کلی استفاده از این متد به صورت زیر است :

Syntax	history.forward ( )

مثال ۱ : در مثال زیر از متد forward در تابع fwd استفاده کرده ایم . این تابع با کلیک بر روی دکمه در شی History رفتن به صفحه بعدی فراخوانی شده و مرورگر را به صفحه بعدی مشاهده شده می برد . برای رفتن به صفحه بعد بر روی دکمه فرمان رفتن به صفحه بعدی در قسمت خروجی کلیک کنید :

Example	

<script type = " text/javascript " >
function fwd ( )
{
history.forward ( ) ;
}
</script>
 	
< input type = " button " name = " FwPage " value = "رفتن به صفحه بعد"
onclick = " fwd ( ) " />
	کد

	خروجی

شی History - متد go

متد go شی History :

این متد یک صفحه که در لیست صفحات شی History مرورگر ذخیره شده است را باز می کند .
همانطور که در معرفی شی History گفتیم ، آدرس URL صفحاتی که شما در یک پنجره مرورگر مشاهده می کنید ، در این شی ذخیره می شود . از این متد برای رفتن به یکی از صفحات مشاهده شده که در شی History قرار دارد ، استفاده می شود .
این متد دارای یک پارامتر اجباری است ، که باید به وسیله آن آدرس URL صفحه و یا شماره آن در لیست شی History را به این متد ارسال کنید . نوع این پارامتر می تواند عددی (Integer) و یا متن (String) باشد .

شکل کلی استفاده از این متد به صورت زیر است :

Syntax	history.go (VLocation *) * VLocation : یک پارامتر عددی یا متن که آدرس صفحه مورد نظر را مشخص می کند

راهنمایی به کار بردن پارمتر : شما می توانید شماره صفحه مورد نظر خود در لیست صفحات شی History و یا آدرس دقیق آن را به این شی توسط پارامتر ارسال کنید . در مثال های زیر استفاده از هر دو حالت این پارامتر را به شما نمایش می دهیم .
اما معنای به کار بردن اعداد توسط این پارامتر به صورت زیر است :

- عدد ۱ : به معنای باز کردن صفحه بعدی مشاهده شده است (البته متد در این حالت زمانی کار می کند ، که کاربر از مسیر رفته به عقب بازگشت کرده باشد.)
- عدد ۲ : به معنای باز کردن ۲ صفحه بعد مشاهده شده نسبت به صفحه جاری است و برای اعداد ۳ و ۴ و ... نیز به همین منوال است.
- عدد ۱ - (منفی :(به معنای باز کردن صفحه قبلی مشاهده شده است.
- اعداد ۲- و ۳- و ... : به معنای باز کردن ۲ صفحه قبل مشاهده شده است و برای اعداد ۳- و ۴- و... وضع به همین منوال است.

نکته مهم : در صورتی که صفحه تعیین شده در متد ( ) go ، قبلا در پنجره جاری مرورگر باز و مشاهده نشده باشد و به عبارت دیگر آن صفحه در شی History مرورگر وجود نداشته باشد ، در هنگام اجرای این متد خطایی رخ نمی دهد و مرورگر در صفحه جاری باقی می ماند .

مثال ۱ : در مثال زیر متد ( ) go شی History را در تابع PrPage به کار برده ایم . با اجرای این متد ، مرورگر به صفحه قبلی مشاهده شده می رود . برای اجرای این متد بر روی دکمه فرمان Previous Page در خروجی مثال کلیک کنید :

Example

<script type = " text/javascript " >
function PrPage ( )
{
history.go (-1) ;
}
</script>
 	
< input type = " button " name = " BtnPrPage " value ="Previous Page"
onclick = " PrPage ( ) " />
	کد

	خروجی

مثل ۲ : در مثال زیر متد go ( ) شی History را در تابع UrlDefine به کار برده ایم . در این مثال آدرس صفحه ای که می خواهیم مرورگر آن را باز کند را به صورت کامل در داخل پرانتز جلوی متد go ( ) توسط پارامتر مربوط قرار داده ایم . با اجرای این متد ، مرورگر صفحه تعیین شده را در صورتی که شما آن را قبلا در همین پنجره مشاهده کرده باشید ، باز می کند . برای اجرای این متد بر روی دکمه فرمان Go Page در خروجی مثال کلیک کنید :

Example	

<script type = " text/javascript " >
function UrlDefine ( )
{
history.go (http://www.developerstudio.ir/JavaScript/Histoty.aspx) ;
}
</script>
 	
< input type = " button " name = " BtnUrl " value ="Go Page" onclick
= " UrlDefine ( ) " />
	کد

	خروجی

شی Location در جاوا اسکریپت - URL صفحه

شی location در جاوا اسکریپت :

شی location یکی از اشیای زیر مجموعه شی window بودها و شالمل اطلاعاتی درباره آدرس (URL) صفحه جاری می باشد . این اطلاعات از قبیل آدرس کامل صفحه ، شماره پورت (port) و نام هاست (hostname) و یا پروتکل مورد استفاده در آن است

همچنین به وسیله این متدهای این شی می توانید آدرس صفحه جاری را تغییر داده ، یک صفحه جدید را باز نموده و یا صفحه جدید را Reload کنید .

این شی توسط واژه کلیدی locaiton در سطح برنامه های اسکریپتی شناخته شده و حالت کلی استفاده آن به صورت زیر است .

البته می توان این شی را بدون استفاده از واژه کلیدی window نیز به کار برد :

Syntax	window.location.نام خاصیت یا location.نام خاصیت

در ادامه به معرفی و تشریح خواص و رویدادهای مهم و پر کاربرد این شی در Java Script می پردازم .

خواص مهم شی location :

در لیست زیر خواص مهم و پر کاربرد شی location قرار دارد . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام خاصیت	شرح
host	این خاصیت نام هاست و شماره پورت (port) آدرس (URL) صفحه جاری را نمایش داده و به وسیله آن می توانید مقدار نام هاست و شماره پورت صفحه را نیز تنظیم کنید .
hostname	این خاصیت نام هاست آدرس (URL) صفحه جاری را نمایش داده و به وسیله آن می توانید مقدار نام هاست صفحه را نیز تنظیم کنید .

href	این خاصیت آدرس کامل (URL) صفحه جاری را نمایش داده و به وسیله آن می توانید مقدار آدرس صفحه را نیز تنظیم کنید .
pathname	این خاصیت مسیر آدرس (URL) صفحه جاری را نمایش داده و به وسیله آن می توانید مقدار مسیر صفحه را نیز تنظیم کنید .
port	این خاصیت شماره پورت آدرس (URL) صفحه جاری را نمایش داده و به وسیله آن می توانید مقدار شماره صفحه را نیز تنظیم کنید .

رویدادهای مهم شی location :

در لیست زیر رویدادهای شی location قرار دارد . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام رویداد	شرح
assign ( )	این متد یک صفحه جدید ، که آدرس آن را در پرانتز مقابل آن تعیین کرده اید ، باز می کند .
reload ( )	این متد یک صفحه جدید ، که آدرس آن را در پرانتز مقابل آن تعیین کرده اید ، باز می کند .
replace ( )	این متد یک صفحه جدید ، که آدرس آن را در پرانتز مقابل آن تعیین کرده اید ، باز می کند .

خواص شی Location

خاصیت hash :

این خاصیت آدرس موجود بعد از علامت # در آدرس کامل صفحه را بر می گرداند . همان طور که در بخش آموزش تگ link در قسمت آموزش HTML دیدید چنانچه در یک صفحه یک لنگر (ancher) ایجاد نماییم ، در صورت پرش و رفتن به آدرس آن لنگر ،نام کامل لنگر در انتهای آدرس صفحه و بعد از علامت # قرار می گیرد. این خاصیت چنانچه نام لنگری در انتهای آدرس صفحه وجود داشته باشد ، مقدار آن را بر می گرداند . یا به عبارت دیگر کلیه حروف و اعداد بعد از علامت # در انتهای آدرس صفحه را بر می گرداند .

Syntax	location.hash ;

مثل : در مثال زیر آدرس یک صفحه فرضی را قرار داده ایم . چنانچه خاصیت hash شی location آن را فراخوانی کنیم ، خروجی زیر به دست خواهد آمد :

Example	
http://www.developerstudio.ir/JavaScript/Window/name.aspx#op <script type ="text/javascript"> document.write (location.hash) ; </script>	کد
#op	خروجی

خاصیت host :

این خاصیت نام هاست (host) وشماره پورت (port number) صفحه جاری را بر می گرداند. خروجی این خاصیت از نوع متن میباشد .

Syntax	location.host ;

مثل : در مثال زیر توسط یک اسکریپت طراحی شده و با استفاده از خاصیت host شی location نام هاست و شماره پورت سایت و صفحه جاری را نمایش داده ایم :

Example	
<script type ="text/javascript"> document.write (location.host) ; </script>	کد
localhost:6094	خروجی

خاصیت hostname :

این خاصیت نام هاست (host) صفحه جاری را بر می گرداند. خروجی این خاصیت از نوع متن میباشد .

Syntax	location.hostname ;

مثل : در مثال زیر توسط یک اسکریپت طراحی شده و با استفاده از خاصیت hostname شی location نام هاست صفحه جاری را نمایش داده ایم :

Example	
<script type ="text/javascript"> document.write (location.hostname) ; </script>	کد
localhost	خروجی

خاصیت pathname :

این خاصیت مسیر کامل دسترسی به صفحه را به کاربر نمایش میدهد. در واقع این خاصیت نحوه پوشه بندی و قرار گیری فایل ها بر روی سرور را نمایان می سازد.از مقدار این خاصیته میتوان در موارد مختلفی استفاده کرد.

Syntax	location.pathname ;

مثل : در مثال زیر توسط یک اسکریپت طراحی شده و با استفاده از خاصیت pathname شی location مسیر کامل دسترسی به صفحه جاری را نمایش داده ایم :

Example	
<script type ="text/javascript"> document.write (location.pathname) ; </script>	کد
/Developer%20v.4/JavaScript/location-pr.aspx	خروجی

خاصیت port :

این خاصیت شماره پورت (port number) سرور جاری صفحه را نمایش می دهد.از مقدار port صفحه می توان برای مقاصد مختلف استفاده کرد. مقدار پیش فرض این خاصیته برای پروتکل HTTP عدد ۸۰ و برای پروتکل FTP عدد ۲۱ است . همچنین مقدار این خاصیت می تواند در سرور های محلی (local) مقاوت باشد.

Syntax	location.port ;

مثل : در مثال زیر توسط یک اسکریپت طراحی شده و با استفاده از خاصیت port شی location شماره Port صفحه جاری را نمایش داده ایم . همانطور که می بینید به علت استفاده از پروتکل HTTP در نمایش این سایت ، مقدار Port آن برابر با ۸۰ که شماره Port پیش فرض این پروتکل است ، می باشد :

Example

کد	<script type ="text/javascript"> document.write (location.port) ; </script>
خروجی	6094

خاصیت protocol :

این خاصیت نوع پروتکل واقعی صفحه جاری را بر میگرداند . پروتکل شیوه ارسال و دریافت اطلاعات بین سرور و کامپیوتر کاربر است . معمولا صفحات وب با استفاده از پروتکل HTTP و فایلهای آپلود یا دانلود شده با استفاده از پروتکل FTP بین کامپیوترها جابجا می شوند.

Syntax	location.protocol ;

مثل : در مثال زیر توسط یک اسکریپت طراحی شده و با استفاده از خاصیت protocol شی location نوع protocol صفحه جاری را نمایش داده ایم . همانطور که می بینید پروتکل صفحه جاری HTTP می باشد .

Example	
کد	<script type ="text/javascript"> document.write (location.protocol) ; </script>
خروجی	http:

خاصیت search :

این خاصیت کلیه حروف و ارقام موجود بعد از علامت ؟ در آدرس صفحه را بر می گرداند . معمولا پارا متر هایی که به صفحات وب ارسال می شوند، بعد از علامت ؟ در انتهای آدرس صفحه قرار می گیرند. خاصیته Search این پارا مترها و

مقدارهایشان را در صورت وجود به صورت متن بر می گرداند. چنانچه هیچ پارامتری به صفحه ارسال نشود، مقدار باز گشتی Null خواهد بود.

Syntax	location.serach ;

مثل : در مثال زیر آدرس یک صفحه فرضی را قرار داده ایم . چنانچه خاصیت search شی location آن را فراخوانی کنیم ، خروجی زیر به دست خواهد آمد :

Example	
http://www.persianstat.com/Results.aspx?id=102287&mode=1 <script type ="text/javascript"> document.write (location.search) ; </script>	کد
?id=102287&mode=1	خروجی

متدهای شی Location

متد assign ( ) :

این متد یک آدرس URL را به عنوان پارامتر دریافت کرده و در هنگام اجرا ، صفحه را به آن آدرس انتقال می دهد . از این متد می توان برای انتقال اتوماتیک صفحه به یک آدرس دیگر استفاده کرد .

Syntax	location.assign (آدرس صفحه مورد نظر) ;

مثل : در مثال زیر آدرس صفحه Home Page سایت را به عنوان پارامتر به متد assign شی Location داده ایم . در هنگامی که کاربر بر روی دکمه فرمان قرار داده شده کلیک نماید ، صفحه به صفحه Home Page سایت منتقل خواهد شد :

Example	

<script type ="text/javascript">
function Go_HomePage ( )
{
location.assign (" http://www.developerstudio.ir/default.aspx ");
}
</script>
 	
< input type="button" name="Btn1" onclick="Go_HomePage( )"
value="Go to Home Page" / >
	کد

	خروجی

متد reload ( ) :

اجرای این متد باعث فراخوانی و لود شدن مجدد صفحه می شود . از این متد می توان برای Refresh و لود کردن اتوماتیک صفحه استفاده کرد . این متد پارامتری دریافت نمی کند .

Syntax	location.reload ( ) ;

مثل : در مثال زیر آدرس یک تابع را ایجاد کرده ایم که در صورت کلیک کاربر بر روی دکمه فرمان قرار داده شده ، اجرا می شود و باعث Refresh و لود شدن مجدد صفحه می گردد :

Example	

<script type ="text/javascript">
function Go_HomePage ( )
{
location.reload ( );
}
</script>
 	
< input type="button" name="Btn2" onclick="Relaod_Page()"
value="Refresh Page" / >
	کد

	خروجی

متد replace ( ) :

این متد آدرس یک صفحه را به صورت پارامتر از کاربر دریافت کرده و صفحه جاری را با آن صفحه انتقال می دهد . از این متد می توان برای انتقال اتوماتیک و برنامه نویسی شده به صفحات دیگر استفاده کرد . عملکرد این متد همانند متد assign ( ) است .

Syntax	location.replace (آدرس صفحه مورد نظر) ;

اعتبار سنجی داده های ورودی در فرم های HTML

اعتبار سنجی داده های ورودی در فرم های HTML :

یکی از نیازهای مهم طراحان وب در هنگام طراحی فرم های HTML ، امکان اعتبار سنجی داده هایی است که کاربر در فیلدهای فرم وارد می نماید . یعنی چه ؟

یعنی که برنامه ابتدا داده های ورودی کاربر را با معیارهای مورد نظر طراح مقایسه کرده و در صورت دارا بودن شرایط درست آنها را به سرور ارسال می کند .

این کار بسیار مهم و حیاتی است . چرا که ما نیاز داریم تا اطلاعات صحیح را از کاربران دریافت نماییم . به چند مورد از این اعتبار سنجی ها اشاره می کنیم :

- در برخی از فیلدها کاربر حتما باید مقداری را وارد نموده و آن را خالی رها نکند . مثلا فیلد نام در فرم های ثبت نام که حتما باید پر شوند.
- بررسی اینکه شماره تلفن یا آدرس ایمیلی که کاربر در فیلدهای مربوطه وارد نموده ، با فرمت واقعی آنها سازگار **باشند** . البته نمی توان این مسئله را بررسی کرد که آیا ایمیل یا تلفن داده شده واقعی هستند یا خیر . فقط می توان رعایت فرمت صحیح توسط کاربر را بررسی کرد . مثلا کاربر آدرس ایمیل را به صورت some@xyz.abc **وارد نموده باشد**.
- یا مثلا بررسی اینکه کاربر در یک فیلد عددی که مربوط به سن است ، عدد وارد نماید و نه متن.

مواردی شبیه موارد فوق ، در طراحی فرم های تحت وب مورد نیاز هستند . **در** این بخش قصد داریم تا شما را با برخی از این عملیات ها آشنا سازیم :

۱ . کنترل خالی نماندن یک فیلد (Require Filed) :

همانطور که گفتم یکی از نیازهای طراح فرم های HTML ، کنترل این مسئله است که کاربر در فیلدهایی اجباری حتما مقداری را وارد نماید و آن را خالی رها نکند . مثل فیلد نام در یک فرم ثبت نام که بایستی حتما پر شود . در مثال کد زیر راه کنترل این مسئله را به شما نمایش داده ایم .

مثل : در مثال زیر یک فرم ساده داریم که کاربر بایستی نام خود را در آن وارد نماید . در صورتی که کاربر نام خود را در کادر وارد نکرده و آن را خالی رها نماید، به هنگام ارسال فرم یک پیام هشدار به وی ارسال می شود که از وی می خواهد مقداری را در کادر وارد نماید . در غیر اینصورت فرم به صورت درست ارسال می شود :

```
function validateForm()
  {
    var x=document.forms["myForm"]["fname"].value;
    if (x==null || x=="")
      {
        alert(" Enter your name please ! ");                    //کد head صفحه قرار گیرد
اسکریپت که باید در قسمت
        return false;
      }                                                                                    کد
  }

< form name="myForm" action="demo_form.asp" onsubmit="return validateForm()"
method="post" >
  First name: < input type="text" name="fname" >
                                  // کد فرم مثال
  < input type="submit" value="Submit" >
< /form >
```

توضیح کد مثال : در مثال فوق ، کاربر باید نام خود را در کادر متنی به نام fname وارد نماید . در زمانی که کاربر فرم را submit می کند ، فرم تابع ( ) validateform را اجرا می کند . این تابع مقدار کادر متن fname **موجود در فرم** myForm را در متغیر x می ریزد . سپس مقدار آن را بررسی نموده و در صورت خالی نبودن مقدار آن ، اجازه submit فرم را می دهد . در صورتی که نام وارد نشده باشد و مقدارش خالی باشد ، پیام هشدار صادر می شود .

۲ . بررسی فرمت صحیح ایمیل های وارد شده (Email Validation) :

در فرم های وب معمولا کادری وجود دارد که کاربر باید آدرس ایمیل خود را در آن وارد نماید . برنامه نویس یا طراح وب نیاز خواهد داشت ، بتواند مقدار وارد شده در کادر متن مربوط به ایمیل را بررسی کرده و چنانچه با فرمت واقعی یک ایمیل سازگار **باشد ، آن را** قبول کرده و در سیستم ثبت نماید .

در اینجا یک مسئله وجود دارد . ما نمی توانیم بررسی کنیم که آیا ایمیل وارد شده توسط کاربر ، ایمیل واقعی وی می باشد و یا اینکه اصلا وجود خارجی دارد یا خیر . بلکه فقط می توانیم بسنجیم که آیا ایمیل وارد شده با فرمت یک ایمیل واقعی مطابقت دارد یا خیر .

برای این منظور باید مقدار وارد شده برای ایمیل دارای یک علامت @ و یک علامت نقطه (.) **باشد** . همچنین باید علامت

@ اولین کاراکتر ایمیل نبوده و همواره قبل از کاراکتر (.) ، قرار بگیرد . در نهایت هم باید حداقل ۲ کاراکتر بعد از کاراکتر نقطه (.) ، وجود داشته باشد .

مثل : در مثال زیر کدی را قرار داده ایم که مقدار وارد شده در کادر متن در نظر گرفته شده برای ایمیل را بررسی کرده و در صورت صحیح بودن فرمت آن ، فرم را submit می کند . در غیر اینصورت یک پیام هشدار به کاربر نمایش می دهد تا ایمیل وارده را تصحیح نماید .

کد

```
function validateForm()
  {
    var x=document.forms["myForm"]["email"].value;
    var atpos=x.indexOf("@");
    var dotpos=x.lastIndexOf(".");
    if (atpos<1 || dotpos < atpos+2 || dotpos+2 > =x.length )
      {
        alert("Not a valid e-mail address");
        return false;
      }
  }

< form name="myForm" action="demo_form.asp" onsubmit="return validateForm();"
method="post" >
  Email:
  < input type="text" name="email" >
  < input type="submit" value="Submit" >
< /form >
```

شی Screen در جاوا اسکریپت - صفحه نمایش

شی Screen در جاوا اسکریپت:

شی Screen یکی از اشیای زیر مجموعه شی window بوده و شامل اطلاعاتی درباره صفحه نمایش کاربر از قبیل اندازه ، عمق رنگ و ... می باشد و توسط کلیه مرورگرهای مطرح پشتیبانی می شود .
این شی توسط واژه کلیدی Screen در سطح برنامه های اسکریپتی شناخته شده و حالت کلی استفاده آن به صورت زیر است .
البته می توان این شی را بدون استفاده از واژه کلیدی window نیز به کار برد :

Syntax	window.Screen.نام خاصیت یا Screen.نام خاصیت

در ادامه به معرفی و تشریح خواص و رویدادهای مهم و پر کاربرد این شی در Java Script می پردازم .

خواص مهم شی Screen :

در لیست زیر خواص مهم و پر کاربرد شی Screen قرار دارد . برای دریافت اطلاعات بیشتر و مثال های عملی بر روی نام آنها کلیک کنید :

نام خاصیت	شرح
availHeight	این خاصیت ارتفاع صفحه نمایش کاربر را نمایش می دهد . این اندازه شامل اندازه منوی taskbar ویندوز نمی شود .
availWidth	این خاصیت پهنای صفحه نمایش کاربر را نمایش می دهد . این اندازه شامل اندازه منوی taskbar ویندوز نمی شود .
height	این خاصیت ارتفاع واقعی صفحه نمایش کاربر را نمایش می دهد . این اندازه شامل اندازه منوی taskbar ویندوز می شود .
width	این خاصیت پهنای واقعی صفحه نمایش کاربر را نمایش می دهد . این اندازه شامل اندازه منوی

taskbar ویندوز می شود .	
این خاصیت توانایی نمایش عمق رنگ در صفحه نمایش کاربر را با واحد bit نمایش می دهد .	colorDepth

رویدادهای زمانی در جاوا اسکریپت

رویدادهای زمانی در جاوا اسکریپت:

به وسیله تابع های خاصی در جاوا اسکریپت ، می توانید کاری کنید که کدهای شما در دوره های زمانی معین تکرار و اجرا شوند و یا یک کد و تابع پس از گذشت زمان معینی از لود شدن صفحه اجرا شود .

به این توابع در جاوا اسکریپت ، تابع های زمانی می گویند .

استفاده از تابع های زمانی در جاوا اسکریپت بسیار ساده است . دو تابع اصلی برای کار با رویدادهای زمانی در جاوا اسکریپت **عبارتند از** :

۱. setInterval ( ) : این تابع پس از گذشت یک مدت زمان معین ، به صورت دوره ای و متناوب ، یک تابع تعیین شده برای آن را اجرا می نماید.
۲. setTimeout ( ) : این تابع پس از گذشت مدت زمان تعیین شده برای آن پس از لود شدن صفحه ، فقط یکبار کد تابع مربوط به خود را اجرا می کند.

setInterval ( ) و setTimeout ( ) ، متدهای شی HTML DOM Window هستند .

متد setInterval :

تابع setInterval ( ) ، به صورت متناوب و دوره ای ، پس از گذشت مدت زمان تعیین شده برای آن ، یک تابع یا کد را اجرا می کند و تا زمانی که به وسیله برنامه متوقف نشود ، به این کار ادامه خواهد داد .

شکل کلی استفاده از این متد به صورت زیر است :

Syntax	setInterval (" javascript function name " , milliseconds) ; مثال : setInterval (" MyFunction " , 1000) ;

در جدول زیر به توضیح موارد syntax پرداخته ایم :

توضیح هر یک از موارد syntax	
پارامتر	توضیح
" javascript function name "	این پارامتر تعیین کننده نام تابعی است که این متد آن را به صورت متناوب و دوره ای اجرا خواهد کرد . این نام باید در بین دو علامت " " قرار بگیرد .
milliseconds	این پارامتر عددی تعیین کننده مدت زمانی است که متد setInterval ( ) ، **تابع خود را** پس از گذشت آن به صورت دوره ای اجرا می کند . واحد این پارامتر بر حسب میلی ثانیه است .

مثال : در مثال زیر با به کار بردن متد ( ) setInterval ، یک ساعت ساده دیجیتال ساخته شده است . به کد مثال دقت نمایید :

کد

```
<p>  بر روی دکمه فرمان زیر کلیک نمایید تا ساعت شروع به کار کند</p>
<button onclick="myFunction( )"> نمایش ساعت </button>
<script type="text/javascript">
   function myFunction( )
      {
         setInterval("myTimer( )", 1000);
      }
</script>
<script type="text/javascript">
   function myTimer( )
      {
         var d = new Date( );
         var t = d.toLocaleTimeString( );
         document.getElementById("demo").innerHTML = t;
      }
</script>

<p id="demo">  </p>
```

خروجی

مشاهده خروجی مثال

چگونه اجرای یک تابع را متوقف سازیم ؟ - تابع: clearInterval :

در هنگام کار با تابع ( ) setInterval ، متوجه شدید که این تابع به صورت متناوب و دوره ای ، یک کد یا تابع را پس از گذشت مدت زمان تعیین شده برای آن ، اجرا می کند . اگر کاربر یا برنامه این تابع را متوقف نسازد ، تا زمانی صفحه یا برنامه تحت وب در حال اجراست ، این تابع نیز اجرا شده و به کار خود ادامه خواهد داد .

اما زمانی می رسد که شاید شما بخواهید پروسه این تابع را متوقف سازید .

برای اینکه بتوانید سیر عملیات یک تابع ( ) setInterval را متوقف سازید ، باید یک متغیر سراسری (Global) را در صفحه تعریف کرده و تابع ( ) setIntetval خود را در آن بریزید . به عبارت دیگر ، به شکل زیر تابع ( ) setInterval را در یک متغیر خاص تعریف می کنید .

Syntax	یک متغیر = setInterval (" javascript function name " , milliseconds) ; مثال : MyStop = setInterval (" MyFunction " , 1000) ;

پس از اینکه تابع ( ) setInterval موردنظر خود را در یک متغیر تعریف کرده اید ، هر زمان که بخواهید آن تابع را متوقف سازید ، باید متغیر مرتبط به آن را به عنوان یک پارامتر ، همانند syntax جدول زیر ، به تابع ( ) clearInterval ، ارسال نمایید . در این صورت تابع از تکرار و تناوب باز خواهد ماند .

Syntax	clearInterval (نام متغیر تابع مورد نظر) ; مثال : clearInterval (MyStop) ;

با ارائه یک مثال ، سعی کرده ایم تا نحوه متوقف سازی یک تابع ( ) setInterval را با استفاده از تابع ( ) clearInterval ، نمایش دهیم .

برای این منظور کد مثال قبل را تکرار کرده ایم ، ولی اینبار کد توقف را نیز به آن اضافه کرده ایم . برای توقف ساعت در حال کار بر روی دکمه فرمان Stop کلیک نمایید . به کد مثال دقت نمایید :

کد

```
<p>  بر روی دکمه فرمان زیر کلیک نمایید تا ساعت شروع به کار کند
سپس در صورتی که خواستید ساعت متوقف شود ، بر روی دکمه فرمان توقف ساعت کلیک نمایید</p>
<button onclick="myFunction( )"> نمایش ساعت </button>
<button onclick="stopTimer( )"> توقف‌ساعت </button>
<script type="text/javascript">
      var MyStop ;
   function myFunction( )
      {
        MyStop = setInterval("myTimer( )", 1000);
      }
</script>
<script type="text/javascript">
   function myTimer( )
      {
         var d = new Date( );
         var t = d.toLocaleTimeString( );
         document.getElementById("demo").innerHTML = t;
      }
</script>

<script type="text/javascript">
   function stopTimer ( )
      {
         clearIntreval ( MyStop ) ;
      }
```

```
</script>

<p id="demo">   </p>
```