# CSS3 ANIMATION


Alireza Hoseinzadeh

Alireza.Hoseinzadeh@outlook.com

# فهرست مطالب

# پیشگفتار

Cascading Style Sheet یا به طور خلاصه CSS ابزاری است برای طراحان و توسعه دهندگان وب که در کنار زبانهای نشانه گذاری مثل HTML برای ایجاد و توسعه صفحات وب مورد استفاده قرار می گیرد. زبان نشانه گذاری HTML با استفاده از پاراگراف ها، لیست ها و سایر عناصر HTML ساختار اصلی یک صفحه وب را ایجاد می کند. وظیفه CSS سبک دهی به عناصر موجود در این ساختار و ایجاد جلوه های بصری زیبا برای صفحات وب می باشد.

CSS اطلاعات مورد نیاز برای کنترل جنبه های بصری یک صفحه وب را در اختیار مرورگر وب قرار می دهد؛ اطلاعاتی مثل موقعیت عناصر HTML در صفحه، نحوه نمایش متون، رنگ ها، تصاویر ، پس زمینه عناصر و ... . هدف اصلی CSS ساده کردن روند طراحی، صفحه آرایی و سبک دهی به صفحات وب می باشد.

تا قبل از CSS3 از خاصیت های CSS برای طراحی و تغییر محتوای ثابت (static) یک سند HTML استفاده می شد و برای ایجاد Animation فقط می توانستید از Javascript و Flash استفاده کنید. اما با معرفی CSS3 و ماژول های Transforms, Transitionsو Animations می توان با استفاده از CSS روی عناصر HTML ، Animation ایجاد کرد. این ماژول ها در تمامی مرورگرهای مدرن پشتیبانی می شوند.

حداقل مهارت مورد نیاز برای شروع کار با CSS3 Animation آشنایی نسبی با زبان نشانه گذاری HTML و CSS2.1 می باشد و در این کتاب فرض بر این است که شما خواننده گرامی آشنایی اولیه با مفاهیم وب و HTML و CSS2.1 را دارید. کتاب حاضر با توجه به مطالب بیان شده قصد آموزش ماژول های Transforms, Transitions, Animation و Gradients را به شما خواننده گرامی دارد.

علیرضا حسین زاده

Webography.ir
Alireza.Hoseinzadeh@Outlook.Com

# مقدمه ای بر CSS3

سازمان W3C یا World Wide Web Consortium متولی اصلی استانداردسازی وب و تکنولوژی های وابسته به وب می باشد.
توسعه استاندارد CSS در این سازمان و توسط گروه CSSWG یا CSS Working Group صورت می گیرد که یک زیر گروه در سازمان W3C می باشد.

پس از اتمام توسعه CSS2.1 توسط CSSWG ، برای توسعه استانداردهای بعدی، CSS به ماژول های متفاوتی تقسیم و دسته بندی شد.
در این کتاب ماژول های Animation , Transitions, Transforms و Gradients مطرح شده ومورد بررسی قرار می گیرند.
تمامی این ماژول ها بخشی از استاندارد CSS3 می باشند و در استانداردهای قبلی CSS وجود نداشتند.

همزمان با توسعه CSS3 توسط CSSWG توسعه دهندگان مرورگرهای اینترنتی به ابداعاتی در زمینه CSS دست زدند. بیشتر خاصیت های تعریف شده در CSS3 اولین بار توسط شرکت های Apple, Google و Mozilla پیشنهاد شدند. ایجاد و استفاده از این خاصیت های جدید توسط این شرکت ها قبل از استانداردسازی آنها توسط W3C می تواند باعث تداخل با استانداردهای رسمی W3C شود. برای حل این مشکل هر یک از شرکت های توسعه دهنده مرورگر از پیشوندی مخصوص به مرورگر خود قبل از نام خاصیت، برای توسعه و معرفی خاصیت های جدید خود، تا پیش از استانداردسازی آنها توسط W3C استفاده می کنند. به این پیشوندها در اصطلاح CSS Vendor Prefix گفته می شود.

## CSS Vendor Prefixes

سازمان W3C برای اینکه به توسعه دهندگان مرورگر اجازه ابداع و معرفی خاصیت های جدید در CSS را بدهد، به مرورگرها اجازه استفاده از یک پیشوند منحصر به فرد برای نامگذاری خاصیت های جدید و آزمایشی را داد. این پیشوندها عبارتند از :

| Prefix | Browser |
|---|---|
| -moz- | Firefox |
| -webkit- | Safari / Chrome / Konqueror |
| -o- | Opera |
| -ms- | Internet Explorer 9+ |

هر مرورگری که بخواهد یک خاصیت جدید را معرفی و پشتیبانی کند، باید قبل از نام خاصیت از پیشوند مخصوص خود استفاده کند.
توجه کنید که این خاصیت ها تا زمانیکه توسط W3C تایید و معرفی نشوند استاندارد نیستند.

برای پشتیبانی از خاصیت های جدید CSS3 در مرورگرهای قدیمی علاوه بر حالت استاندارد دستورات باید از حالت پیشوندی دستورات نیز استفاده کرد. خاصیت استاندارد W3C باید در انتهای دستورات ذکر شود.

مثال :

```
.rotate{
        -moz-transform: rotate(45deg);
        -webkit-transform: rotate(45deg);
        -o-transform: rotate(45deg);
        -ms-transform: rotate(45deg);
```

```
        transform: rotate(45deg);
}
```

در این حالت مرورگرها تنها دستوری را که برایشان معتبر و تعریف شده است ، اعمال کرده و باقی دستورات را نادیده خواهند گرفت. به غیر ازدستور Gradient ، در سایر ماژول های CSS3 نام و مقدار خاصیت ها در هر دوحالت استاندارد و Prefix کاملا یکسان می باشند و در حالت Prefix نیز باید مشابه حالت استاندارد رفتار کرده و فقط پیشوندهای مخصوص هر مرورگر استفاده کنیم.

در بیشتر مثال های این کتاب برای جلوگیری از تکرار و طولانی شدن دستورات فقط حالت استاندارد خاصیت ها استفاده شده است. برای پشتیبانی کامل از مثال ها در تمامی مرورگرهاشما می توانید حالت Prefix دستورات را نیز قبل از حالت استاندارد استفاده کنید.

# فصل اول

# 2D Transforms

در CSS3 چندین خاصیت مختلف برای تغیر شکل عناصر موجود در صفحه وب ارائه شده است. با استفاده از این خاصیت ها میتوان زاویه و جهت قرار گیری عناصر موجود در صفحه، اندازه عناصر و ... را در فضای دو بعدی و سه بعدی تغییر داده و دستکاری کرد.

خاصیت اصلی استفاده شده برای انجام تغییرات ذکر شده خاصیت transform می باشد. با استفاده از برخی از توابع و اختصاص مقادیر به این توابع می توان نحوه و میزان تغییر شکل عناصر را مشخص کرد. برای مثال دستور زیر عنصر مورد نظر را به اندازه ۱۰ درجه در جهت عقربه های ساعت می چرخاند:

```
transform : rotate(10deg);
```

قاعده کلی برای استفاده از خاصیت transform به شکل زیر می باشد :

```
transform : function(value);
```

توابع قابل استفاده در این خاصیت در ادامه فصل بررسی خواهند شد .

## rotate

این تابع برای چرخاندن یک عنصر به اندازه ۰ تا ۳۶۰ درجه در جهت یا خلاف جهت عقربه های ساعت استفاده می شود.

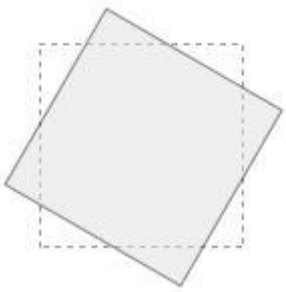
قاعده کلی :

```
transform : rotate(<angle>);
```

مثال :

```
transform : rotate(30deg);
```

دستور فوق عنصر مورد نظر را به اندازه 30 در جه در جهت عقربه های ساعت می چرخاند. مقادیر منفی در این تابع عناصر را در خلاف جهت عقربه های ساعت می چرخاند .



برای زوایا چندین واحد اندازه گیری مختلف (angle) وجود دارد که در تابع rotate قابل استفاده می باشد:

- ✓ deg : ۳۶۰ در جه (degrees) برابر با یک دایره کامل می باشد.

```
rotate(90deg);
```

- ✓ grad : ۴۰۰ گرادیان (gradians) برابر با یک دایره کامل می باشد.gradians با نام های gons یا grades نیز شناخته می شود .

```
rotate(100grad);
```

- ✓ rad : ۲π رادیان (radians) یا ۶,۲۸۳۱۸۵۳rad برابر با یک دایره کامل می باشد.

```
rotate(1.57rad);
```

- ✓ turn : یک دایره کامل برابر با turn ۱ می باشد.

```
rotate(0.25turn);
```

## scale

با استفاده از تابع scale می توان اندازه یا مقیاس یک عنصر را تغییر داده و عنصر را بزرگتر یا کوچکتر از حالت نرمال نمایش داد.

قاعده کلی :

```
transform: scaleX(<number>);
transform: scaleY(<number>);
transform: scale(scaleX, scaleY);
```

تابع scaleX() برای تغییر طول (width) و تابع scaleY() برا ی تغییر عرض (height) عنصر مورد نظر به کار می رود. پارامتر استفاده شده در این توابع یک مقدار عددی می باشد. این مقدار عددی مشخص می کند که طول یا عرض عنصر چند برابر حالت نرمال باشد.

مثال :

```
transform : scaleX(2);
```

دستور فوق طول عنصر (مقیاس افقی) را دو برابرحالت نرمال تعریف می کند .

تابع scale() حالت خلاصه و کوتاه نویسی شده توابع scaleX() و scaleY() می باشد. در این تابع دو مقدار عددی قابل استفاده می باشد که مقدار عددی اول مقیاس افقی عنصر (scaleX) و مقدار دوم مقیاس عمودی عنصر (scaleY) را مشخص می کند. در صورت استفاده از یک مقدار عددی در این تابع مقادیر مقیاس های عمودی و افقی عنصر یکسان در نظر گرفته خواهند شد. در تابع scale() مقدار پیش فرض برابر یک می باشد. مقادیر اعشاری ، مقادیر بین صفر و یک و مقادیر منفی نیز قابل استفاده می باشند .

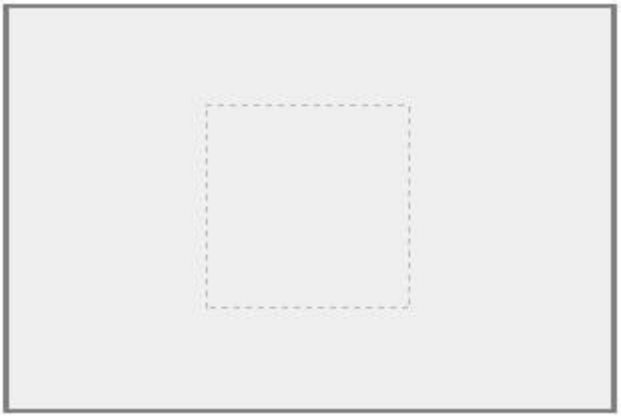
مثال :

```
scale(۳,۲);
```

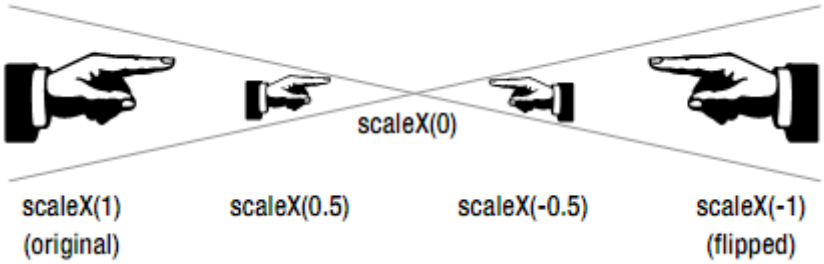
برای به دست آوردن اندازه جدید عنصر مقادیر مقیاس های افقی و عمودی را در هم ضرب می کنیم. برای مثال فوق داریم :

```
scaleX * scaleY = ۳ * ۲ = 6
```

طول عنصر ( مقیاس افقی ) سه برابر و عرض عنصر ( مقیاس عمودی ) دو برابر حالت نرمال می باشد؛ یعنی اندازه جدید عنصر شش برابر حات نرمال می باشد .



scaleX(0) باعث محو شدن عنصر می شود. scaleX(-1) و یا scale(-1,1) باعث چرخیدن عنصر در جهت افقی (flipeed horizontally) می شود . scaleY(-1) و یا scale(1,-1) باعث چرخیدن عنصر در جهت عمودی (flipped vertically) می شود .




## skew

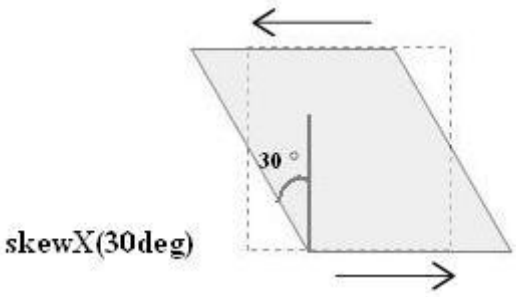
تابع skew() برای مورب کردن عناصر در صفحه به کار می رود و با استفاده از این تابع می توان زوایای اضلاع افقی و عمودی عناصر را تغییر داد .

قاعده کلی :

```
transform : skewX(<angle>);
transform : skewY(<angle>);
transform : skew(skewX,skewY);
```

## skewX

اگر دو طرف یک مستطیل را بگیریم ( گوشه پایین سمت راست و گوشه بالا سمت چپ ) و مستطیل را در جهت افقی طوری بکشیم که اضلاع آن همچنان موازی باقی بمانند، آنگاه زاویه ای که اضلاع چپ و راست با محور عمود ایجاد می کنند باهم مساوی بوده و برابر است با skewX ؛ یعنی skewX باعث مورب شدن اضلاع عمودی عناصر می شود . عکس این حالت ( گوشه بالا سمت راست و گوشه پایین سمت چپ ) یک زاویه با مقدار منفی برای skewX ایجاد خواهد کرد .

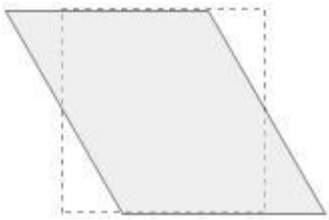



## skewY

skewY نیز مشابه skewX می باشد با این تفاوت که در این حالت باید مستطیل را در جهت عمودی بکشیم . در این حالت زاویه ای که اضلاع بالا و پایین با محور افقی ایجاد می کنند برابر است با skewY ؛ یعنی skewY باعث مورب شدن اضلاع افقی عناصر می شود .

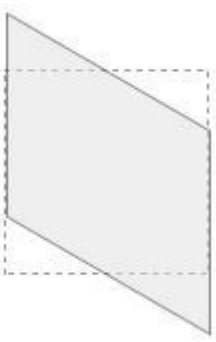
مثال :

```
transform : skewX(30deg);
```



مثال :

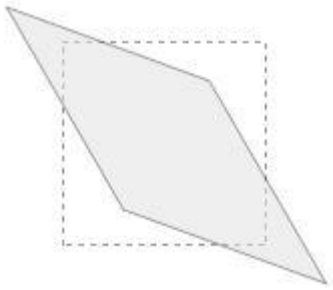
```
transform : skewY(30deg);
```



skew(X,Y) حالت کوتاه نویسی برای دو مورد قبل می باشد. در این حالت اگر یک مقدار ذکر شود، این مقدار به عنوان skewX در نظر گرفته خواهد شد و مقدار skewY برابر صفر در نظر گرفته می شود.

مثال :

```
transform : skew(30deg,۲۰deg);
```



با استفاده از skew(X,Y) و با استفاده از روش زیر می توان تابع rotate() را شبیه سازی کرد :

```
rotate(value) = skew(value, -value)
```

مثال :

```
rotate(45deg) = skew (45deg,-45deg)
```

## translate

با استفاده از تابع translate می توان عناصر HTML را نسبت به موقعیت های پیش فرض خود در جهت های عمودی و افقی جابجا کرد.

قاعده کلی :

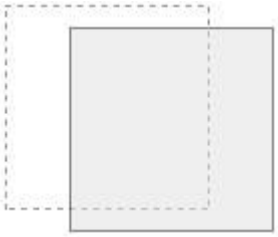
```
transform : translateX( <length> | <percentage> );
transform : translateY( <length> | <percentage> );
transform : translate(translateX,translateY);
```

تابع translateX() عناصر را در جهت افقی و تابع translateY() عناصر را در جهت عمودی جابجا می کند. تابع translate() نیز حالت خلاصه نویسی شده دو تابع قبلی می باشد. مقدار پیش فرض برای این توابع مقدار صفر می باشد .

مثال :

```
transform : translate(30%,10px);
```

از ارتفاع یا عرض خود عنصر به سمت پایین جا به جا می 10px به سمت راست و به اندازه 30%دستور فوق عنصر مورد نظر به اندازه کند .



در صورت استفاده از مقادیر منفی جهت جابجایی تغییر خواهد کرد.

مثال :

```
translateX(-50px);
```

دستور فوق عنصر مورد نظر را 50px به سمت چپ جابجا می کند .

پس از اعمال توابع transform (توابع rotate و scale و skew ) عنصر به دلیل تغییر شکل، در ظاهر موقعیت جدیدی پیدا می کنند و ممکن است با سایر عناصر موجود در صفحه تلاقی پیدا کند. با جابجایی عناصر بوسیله استفاده از تابع tranlate می توان این مشکل را برطرف کرد .

## ترکیب توابع transform

برای ترکیب توابع transform و اعمال چند تابع روی یک عنصر باید تمامی توابع مورد نظر در خاصیت transform قرار داده و این توابع را با یک space از یکدیگر جداکنیم .

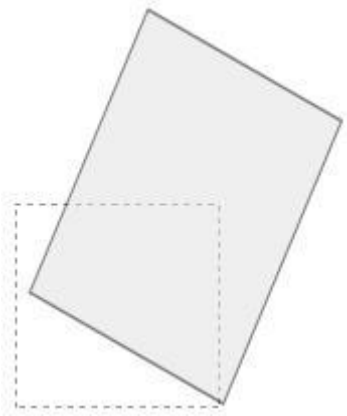
قاعده کلی :

```
transform : function(value) function(value) ... ;
```

مثال :

```
transform : rotate(30deg) scale(1.1,1.5) skewX(10deg) translate(10px,-40px);
```

در مثال فوق تمامی توابع ذکر شده در خاصیت transform روی عنصر موردنظر اعمال خواهد شد.



استفاده از خاصیت های transform به طور جاگانه برای هر تابع صحیح نمی باشد و در این حالت فقط دستور آخر روی عنصر موردنظر اعمال خواهد شد.

مثال :

```
transform : rotate(30deg);
transform : scale(1.1,1.5);
transform : skew(10deg);
```

در مثال فوق فقط دستور آخر یعنی skew(10deg) روی عنصر موردنظر اعمال خواهد شد و باقی دستورات در نظر گرفته نخواهند شد.

روش دیگر برای ترکیب توابع transform استفاده از تابع matrix() می باشد که ۶ پارامتر می گیرد. به علت پیچیدگی از بیان این روش صرف نظر می کنیم.

## transform-origin

در حالت عادی هنگامی که یکی از توابع transform را برای تغییر شکل یک عنصر به کار می برید، مرورگر مرکز عنصر را به عنوان مبدا یا نقطه اعمال تغییرات ( transformation point ) در نظر می گیرد. به عنوان مثال هنگامی که با استفاده از تابع rotate() یک عنصر را می چرخانید ، مرورگر به طور پیش فرض عنصر را حول نقطه مرکزی خود عنصر می چرخاند. CSS3 با استفاده از خاصیت transform-origin امکان تغییر محل این نقطه را فراهم می کند. این خاصیت مشابه خاصیت background-position می باشد و می توان با استفاده از کلمات کلیدی، مقادیر مطلق (absolute) مثل pixel ، مقادیر وابسته (relative) مثل ems و یا مقادیر درصدی (percentage) این خاصیت را مقدار دهی کرد.

قاعده کلی :

```
transform-origin : <x-position> <y-position>;
transform-origin : [<percentage> | <length>] | [left| center | right]] |
[[top | center | bottom]
```

در این خاصیت می توان از length ، percentage و کلمات کلیدی استفاده کرد. آسانترین روش برای تخصیص مقدار به این خاصیت استفاده از کلمات کلیدی است:

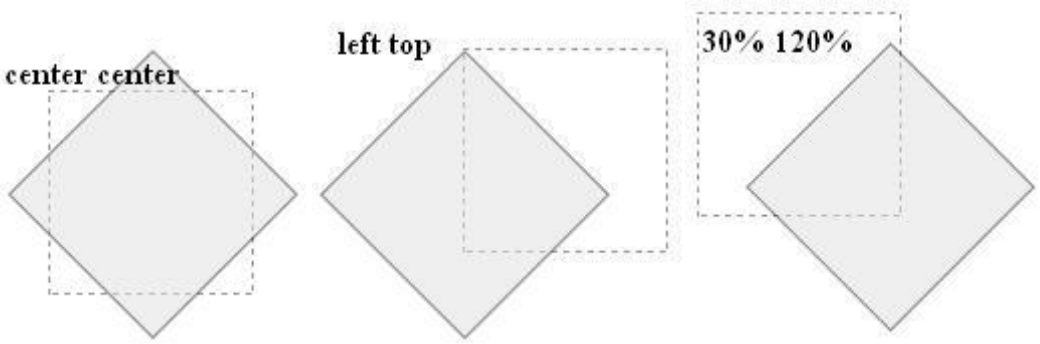
مقادیر افقی : left , center , right

مقادیر عمودی : top , center , bottom

مقدار پیش فرض center center می باشد. در این خاصیت اگر مقدار دوم ( y-position ) ذکر نشود برابر با center یا 50% در نظر گرفته خواهد شد .

مثال :

```
transform : rotate(45deg);
transform-origin : left top;
```




هنگام استفاده از مقادیر percentage و length باید ابتدا موقعیت افقی و سپس موقعیت عمودی ذکر شود.

مثال :

```
transform-origin : 30% 50%;
```

دستور فوق مشخص می کند که transformation point باید 30% در جهت افقی به سمت داخل عنصر و 50% در جهت عمودی به سمت داخل عنصر ( به سمت پایین ) قرار گیرد.

# فصل دوم

# Transition

css transition برای ایجاد animation و تغییر خصوصیات یک عنصر از یک حالت اولیه به یک حالت دیگر استفاده می شود. به عبارت دیگر css transition برای ایجاد یک animation یک مرحله ای و تغییر حالت و خصوصیات یک عنصر به صورت نرم و آهسته از یک حالت به حالت دیگر استفاده می شود که این animation به صورت حالت رفت و حالت برگشت (reverse) انجام می شود. به عنوان مثال می توان هنگام قرار گرفتن اشاره گر موس روی یک عکس (:hover) ابعاد عکس را در یک بازه زمانی یک ثانیه ای دو برابر کرد. هنگامیکه اشاره گر موس از روی عکس کنار رود animation به صورت معکوس اجرا شده و عکس به حالت و ابعاد اولیه خود باز می گردد .

transition یک animation مطلق و یک مرحله ای است به این معنی که این animation فقط زمانی اتفاق می افتد که مقادیر یکی از خاصیت های css عنصر موردنظر تغییر کرده باشد. برای اینکه یک transition انجام شود باید چهار شرط برقرار باشد ؛ یک مقدار اولیه برای خاصیت های موردنظر عنصر ،یک مقدار پایانی یا ثانویه برای خاصیت های موردنظر ،استفاده از خود خاصیت های transition و یک رویداد برای انجام animation مثل (:hover) .

## transition-property

این خاصیت مشخص می کند که کدامیک از خاصیت های یک عنصر باید animate شوند .

قاعده کلی :

```
transition-property : <css properties>  | all | none;
```

مقدار پیش فرض در خاصیت transition-property مقدار all می باشد و در صورتیکه خاصیت transition-property برای یک transition تعریف نشده باشد برابر all در نظر گرفته خواهد شد و به این معنی است که تمامی خاصیت های معتبر در صورت تغییر animate خواهند شد. در صورت استفاده از مقدار none هیچ کدام از خاصیت ها animate نخواهند شد.

تمام خاصیت های css قابلیت animate شدن ندارند و فقط تعداد خاصی از آنها مثل color , background , border , margin , padding , width , height , transform و ... در این خاصیت قابل استفاده می باشند .

برای ایجاد یک animation با استفاده از transition روش استاندارد و درست این است که خاصیت های مورد استفاده در transition-property را به صورت صریح تعریف کنیم و از استفاده از حالت کوتاه نویسی عناصر خودداری کنیم. به عنوان مثال برای animate کردن margin یک عنصر از خاصیت های margin-top , margin-right , margin-left و margin-bottom به جای حالت کوتاه نویسی آن (یعنی margin) استفاده کنیم.

## transition-duration

این خاصیت مدت زمان کامل شدن animation را مشخص می کند.

قاعده کلی :

```
transition-duration : <time>;
```

مقادیر قابل استفاده در این خاصیت مدت زمان برحسب ثانیه (s) یا میلی ثانیه (ms) می باشد. مقدار پیش فرض برای این خاصیت مقدار صفر است و در نتیجه اگر این خاصیت برای یک عنصر تعریف نشود animation انجام نخواهد شد. در صورت استفاده از مقادیر منفی برای این خاصیت مقدار منفی برابر صفر در نظر گرفته خواهد شد.

مثال :

```
E{
font-size : 2em;
transition-property : font-size;
transition-duration : 2.5s;
}
E:hover{
font-size : 3em;
}
```

دستور فوق باعث می شود در صورت قرار گرفتن اشاره گر موس روی عنصر مورد نظر فونت عنصر در مدت زمان ۲,۵ ثانیه از 2em به 3em ، animate شود. هنگام خروج اشاره گر موس از روی عنصر مورد نظر فونت عنصر در مدت زمان ۲ ثانیه از 3em به 2em ، (حالت معکوس ) animate خواهد شد.

می توان چندین خاصیت از یک عنصر را به صورت همزمان animate کرد. برای این منظور باید در خاصیت transition-property مقادیر خاصیت های مورد نظر را با استفاده از یک کاما (,) از یکدیگر جدا کرد. هم چنین در خاصیت transition-duration نیز می توان مقادیر زمان را با استفاده از کاما به ترتیب متناسب با مقادیر استفاده شده در خاصیت transition-property از یکدیگر جدا کرد.

مثال :

```
E{
font-size : 2em;
color : red;
width : 50px;
transition-property : font-size , color , width;
transition-duration : ۳s;
}
E:hover{
font-size : 3em;
color : blue;
width : 100px;
}
```

در مثال فوق در صورت قرار گرفتن اشاره گر موس بر روی عنصر موردنظر ،اندازه فونت ،رنگ و عرض عنصر در مدت زمان ۳ ثانیه به مقادیر ذکر شده در شبه کلاس hover ، animate خواهد شد. در هنگام خروج اشاره گر موس از روی عنصر نیز animation به صورت معکوس اجرا خواهد شد.

اگر مقادیر transition-duration برابر با مقادیر transition-property باشد در این حالت مقادیر زمانی transition-duration به ترتیب برای خاصیت های متناظر ذکر شده در transition-property اعمال خواهند شد. در غیر اینصورت دو حالت اتفاق می افتد :

- ✓ مقادیر transition-duration بیشتر از مقادیر transition-property باشد:

در این حالت مقادیر اضافی transition-duration در نظر گرفته نخواهد شد و تاثیری روی animation نخواهد داشت.

مثال :

```
E{
font-size : 2em;
color : red;
transition-property : font-size , color;
transition-duration : ۴s , 2s , ۱.۵s;
}

E:hover{
font-size : 3em;
color : blue;
}
```

در مثال فوق فونت عنصر در مدت زمان 4s از 2em به 3em و رنگ عنصر در مدت زمان 2s از red به blue، animate خواهد شد و مقدار 1.5s تاثیری روی animation نخواهد داشت.

- ✓ مقادیر transition-duration کمتر از مقادیر transition-property باشد:

در این حالت مقادیر به ترتیب برای خاصیت های ذکر شده در transition-property در نظر گرفته خواهند شد و برای باقیمانده خاصیت ها مقادیر transition-duration از ابتدا در یک چرخه در نظر گرفته خواهند شد. اگر در خاصیت transition-duration یک مقدار برای زمان ذکر شده باشد این مقدار برای تمامی خاصیت ها در نظر گرفته خواهد شد.

مثال :

```
E{
font-size : 2em;
color : red;
width : 50px;
transition-property : font-size , color , width;
transition-duration : ۳s , 2s;
}
E:hover{
font-size : ۴em;
color : blue;
width : 100px;
}
```

در مثال فوق فونت عنصر در مدت زمان 3s ، رنگ عنصر در مدت زمان 2s و عرض عنصر در مدت زمان 3s به مقادیر جدید animate خواهد شد.

## transition-timing-function

این خاصیت آهنگ انجام یک animation را مشخص می کند. با استفاده از این خاصیت می توان سرعت و آهنگ انجام animation را در طول بازه زمانی انجام animation مشخص کرد .

قاعده کلی :

```
transition-timing-function: <keyword> | steps(<number>, <keyword>) | cubic-bezier(x1,y1,x2,y2);
```

کلمات کلیدی قابل استفاده در این خاصیت عبارتند از :

- ✓ ease : animation در سه مرحله انجام خواهد شد. در ابتدای بازه زمانی تعریف شده در خاصیت transition-duration ، animation با آهنگ آهسته شروع شده و سپس به سرعت انجام شده ودر انتهای بازه زمانی نیز به سرعت پایان خواهد یافت.
- ✓ linear : کل animation در یک مرحله و با آهنگ یکسان انجام خواهد شد.
- ✓ ease-in : دو مرحله ؛ شروع با آهنگ سریع و باقی تا پایان با آهنگ آهسته
- ✓ ease-out : دو مرحله ؛ شروع با آهنگ آهسته و باقی تا پایان با آنگ سریع
- ✓ ease-in-out : سه مرحله ؛ مشابه ease ولی با شروع و پایان آهسته تر
- ✓ step-start : animation در یک مرحله و در ابتدای بازه زمانی تعیین شده انجام خواهد شد.
- ✓ step-end : animation در یک مرحله و در انتهای بازه زمانی تعیین شده انجام خواهد شد.
- ✓ steps(<number>,<keyword>) :
  - number : این عدد تعداد مراحل انجام یک animation را مشخص می کند. مدت زمان انجام animation (transition-duration) به تعداد بازه های زمانی مساوی با این عدد تقسیم می شود.
  - keyword : دوکلمه کلیدی start و stop در این پارامتر قابل استفاده می باشند ؛ کلمه کلیدی start مشخص می کند که animation باید در ابتدای هر بازه زمانی (step) انجام شود و stop نیز مشخص می کند که animation باید در انتهای هر بازه زمانی انجام شود.
- ✓ cubic-bezier() :

یکی از مقادیر قابل استفاده در خاصیت transition-timing-function تابع cubic-bezier() می باشد که دارای ساختار زیر است:

```
cubic-bezier(x1,y1,x2,y2);
```

منحنی cubic-bezier با استفاده از چهار نقطه و مختصات این چهار نقطه روی یک شبکه چهارخانه که مختصات محور های آن از صفر تا یک درجه بندی شده اند قابل ترسیم می باشد. این چهار نقطه عبارتند از p0,p1,p2,p3 .این چهار نقطه میزان خمیدگی منحنی رسم شده در این صفحه مختصات را تعریف می کنند و هر کدام از این نقاط با استفاده از مختصات دوتایی (x,y) روی صفحه مختصات چهارخانه قابل ترسیم می باشند.
مختصات نقاط p0,p3 همواره ثابت بوده و عبارت است از :

```
P0 (0,0)          و          P3(1,1)
```

مختصات دو نقطه باقیمانده p1,p2 در تابع cubic-bezier() استفاده شده و میزان خمیدگی منحنی را تعیین می کند:

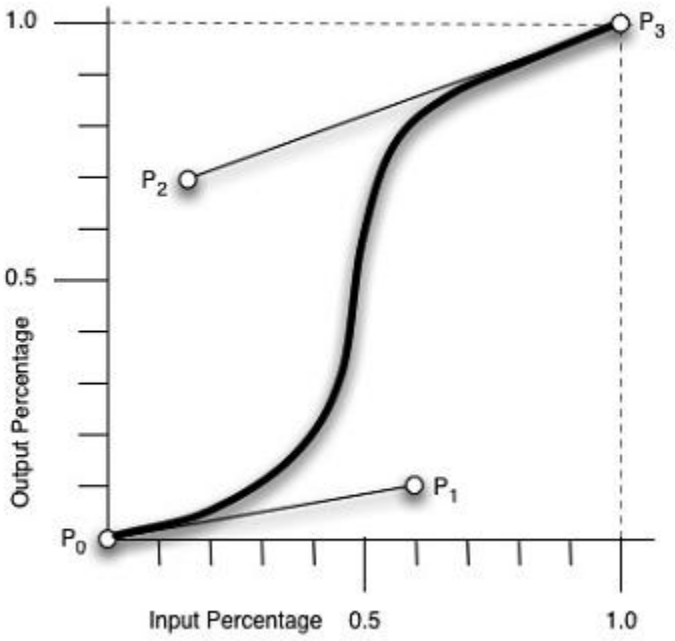
P1(x1,y1) و P2(x2,y2)

در نتیجه ساختار نهایی تابع cubic-bezier() به شکل زیر خواهد بود:

```
cubic-bezier(x1,y1,x2,y2);
```

آهنگ اجرای animation در این حالت متناسب با میزان انحنای منحنی Bezier می باشد. برای مثال مقدار تابع Bezier متناظر برای کلمه کلیدی linear یک خط مستقیم با مختصات P1(0,0) و P2(1,1) می باشد و برای باقی کلمات کلیدی عبارتند از :

```
ease :          cubic-bezier(0.25,0.1,0.25,1);
linear :        cubic-bezier(0,0,1,1);
ease-in :   cubic-bezier(0.42,0,1,1);
ease-out : cubic-bezier(0,0,0.58,1);
ease-in-out :   cubic-bezier(0.42,0,0.58,1);
```




| Point | Coordinates (x, y) |
|---|---|
| p0 | (0, 0) |
| p1 | (0.6, 0.1) |
| p2 | (0.15, 0.8) |
| p3 | (1, 1) |

مثال:

```
E{
font-size: 2em;
color : red;
width : 50px;
transition-property : font-size , color , width;
transition-duration : ۳s , 2s , 1.5s;
transition-timing-function : ease-in , ease-out , cubic-bezier(0.6,1,0.15,0.8);
}
E:hover{
font-size : ۴em;
color : blue;
width : 100px;
}
```

متناسب با تعداد خاصیت های استفاده شده در transition-property ، در خاصیت transition-timing-function نیز می توان از مقادیر timing استفاده کرد. تمامی موارد ذکر شده بالا در مورد مقادیر و ترتیب اعمال آنها در خاصیت transition-duration، در مورد خاصیت transition-timing-function نیز صدق می کند و برقرار است.

## transition-delay

استفاده از این خاصیت باعث ایجاد تاخیر در انجام animation می شود.

قاعده کلی:

```
transition-dalay : <time>;
```

مقادیر قابل استفاده در این خاصیت مدت زمان برحسب ثانیه (s) یا میلی ثانیه (ms) می باشد. مقدار پیش فرض این خاصیت برابر صفر می باشد و animation بلافاصله شروع به اجرا خواهد کرد. مقادیر منفی برای این خاصیت برابر صفر در نظر گرفته خواهند شد.

مثال :

```
E{
font-size : 2em;
transition-property : font-size;
transition-duration : 2.5s;
transition-delay : 2s;
}
E:hover{
font-size : 3em;
}
```

در مثال فوق ۲ ثانیه پس از قرار گرفتن اشاره گر موس روی عنصر موردنظر animation شروع به اجرا خواهد کرد.

## transition

خاصیت transition حالت کوتاه نویسی برای چهار خاصیت ذکر شده قبلی می باشد و این چهار خاصیت را باهم ترکیب می کند.

قاعده کلی :

```
transition : <transition-property> <transition-duration> <transition-timing-function> <transition-delay>;
```

مقادیر استفاده شده در این خاصیت باید با یک space از یکدیگر جدا شوند. به غیر از مقادیر زمانی duration و delay ترتیب استفاده از سایر مقادیر در این خاصیت مهم نیست. اولین مقدار زمانی استفاده شده در این خاصیت بیانگر duration و دومین مقدار زمانی بیانگر delay می باشد.

مثال:

```
E{
font-size : 2em;
transition : font-size 2.5s ease-in-out 1s;
}
E:hover{
font-size : ۴em;
}
```

در مثال فوق اندازه فونت عنصر در مدت زمان ۲,۵ ثانیه با آهنگ ease-in-out و با یک ثانیه تاخیر به 4em ، animate خواهد شد. در حالت کوتاه نویسی transition اولین مقدار زمانی بیانگر duration و دومین مقدار زمانی بیانگر delay در animation می باشد.

## multiple transition

می توان چندین transition را به صورت کوتاه نویسی شده باهم در یک خاصیت transition ترکیب کرد.

قاعده کلی :

```
transition : <transition> , <transition> , <transition> , ...
```

مثال :

```
E{
font-size : 2em;
color : red;
width : 50px;
transition : font-size 3s ease-in 1.5s,
             color 2s ease-out 2s,
             width 1.5s cubic-bezier(0.6,1,0.15,0.8) 0;
}
E:hover{
font-size : ۴em;
color : blue;
width : 100px;
}
```

مثال :

```
E{
font-size : 2em;
color : red;
width : 50px;
transform : rotate(0deg);
transition : all 500ms;
}
E:hover{
font-size : ۴em;
color : blue;
width : 100px;
transform : rotate(45deg);
}
```

در مثال فوق تمامی (all) خاصیت های عنصر در مدت زمان ۵۰۰ میلی ثانیه از حالت اولیه خود به حالت ثانویه تعریف شده در شبه کلاس :hover ، animate خواهند شد. در مثال فوق به جای دستور transition : all 500ms; می توان از دستور transition : 500ms; استفاده کرد؛ زیرا مقدار پیش فرض transition-property مقدار all می باشد و می توان از ذکر آن صرف نظر کرد.

## ایجاد animation چند مرحله ای با استفاده از transition

برای اینکه بتوانیم یک animation چند مرحله ای با استفاده از transition داشته باشیم، باید درهنگام تعریف خاصیت transition-delay مقدار این خاصیت را برای animation مرحله جاری برابر با مجموع مقادیر خاصیت های transition-duration از animation های مراحل قبل قرار دهیم.

مثال :

```
E{
width : 500px;
height : 300px;
font-size : 2em;
background-color : red;
color : #000000;
transition-property : font-size , background-color , color;
transition-duration : ۲s;
transition-delay : 0s , ۲s , ۴s;
}
E:hover{
font-size : 7em;
background-color : blue;
color : #ffffff;
}
```

در مثال قبل font-size عنصر در مدت زمان 2s و بدون تاخیر از 2em به 7em، animate می شود (مرحله اول) و با 2s تاخیر (مجموع transition-duration مراحل قبل) رنگ پس زمینه عنصر در مدت زمان 2s از red به blue تغییر پیدا می کند (مرحله

دوم) و در نهایت با 4s تاخیر (مجموع transition-duration مراحل قبل) رنگ متن عنصر در مدت زمان 2s از #000000 به #ffffff تغییر پیدا می کند.

## فصل سوم

# Animating with CSS Keyframes

با استفاده از css transition فقط امکان ایجاد animation های ساده و animate کردن خصوصیات یک عنصر از یک موقعیت اولیه به یک موقعیت ثانویه وجود دارد. یعنی transition فقط امکان animate کردن بین دو حالت را برای ما فراهم می کند. در این حالت تنها کنترلی که می توانیم روی animation داشته باشیم ایجاد آهنگ (pace) با استفاده از timing می باشد. اما با استفاده از ماژول animation در css3 و تعریف keyframes می توان animation های چند مرحله ای ایجاد کرد. همچنین امکان کنترل بر روی حالت بازگشت (reverse) یک animation وجود دارد و می توان تعداد دفعات انجام یک animation را مشخص کرد و یا اینکه animation را به تعداد دفعات نامحدود و در یک حلقه تکرار کرد. با استفاده از ماژول animation در css3 می توان چندین مرحله animation را روی یک عنصر ایجاد کرد. همچنین می توان یک animation را یک بار تعریف کرده و از آن به دفعات برای عناصر مختلف موجود در صفحه استفاده کرد.

## **تعریف** keyframes

ایجاد یک animation در css3 شامل دو مرحله می باشد:

۱. تعریف keyframes
۲. استفاده از keyframes تعریف شده برای یک یا چند عنصر موردنظر

بر خلاف transition که تمامی خصوصیات موردنظر برای animation باید در داخل بلاک مربوط به عنصر تعریف شود، keyfremes ها باید ابتدا به صورت جداگانه تعریف شده و سپس استفاده شوند. keyframes ها مقدار خصوصیات عناصر در هر مرحله از animation را شرح داده و تعریف می کنند. همچنین می توان آهنگ انجام animation در هر مرحله از animation را نیز به صورت اختیاری تعریف کرد؛ مشابه با آنچه در transition وجود داشت.

قاعده کلی برای تعریف مجموعه ای از keyframes ها با استفاده از کلمه کلیدی @keyframes به صورت می باشد:

```
@keyframes keyframes-name {
        /* keyfreme definitions */
}
```

در ساختار فوق keyframes-name یک نام اختیاری است که به keyframes موردنظر اختصاص می یابد. در داخل بلاک @keyframes می توان چندین بلاک دیگر تعریف کرد که هرکدام از این بلاک ها مشخص کننده یک مرحله از animation می باشند و به هر کدام از این بلاک ها یک keyframe می گویند. با تعریف این بلاک ها ساختار نهایی یک keyframes به صورت زیر خواهد بود:

```
@keyframes keyframes-name{
    from{
        /* properties definitions */
    }
    n%{
        /* properties definitions */
    }
    to{
        /* properties definitions */
    }
}
```

هر کدام از بلاک ها یا keyframe های فوق یک مرحله از animation را مشخص می کنند و خصوصیات یک عنصر در هر مرحله از animation باید در داخل بلاک های keyframe تعریف شود. در نهایت مجموعه ای از keyframe ها تشکیل keyframes نهایی را می دهند. هرکدام از این بلاک ها با یک عدد بین ۰ تا ۱۰۰ و علامت ٪ مقابل آن تعریف می شوند. در نتیجه داریم:

- ✓ بلاک شروع animation : برای بلاک شروع animation می توان از مقدار 0% و یا کلمه کلیدی from استفاده کرد.
- ✓ بلاک ها یا مراحل میانی animation : برای مراحل میانی animation می توان به تعداد موردنیاز بلاک یا keyframe تعریف کرد. برای تعریف این بلاک ها باید از یک مقدار عددی به همراه علامت ٪ استفاده کرد. به عنوان مثال 50% به معنی میانه یک animation می باشد؛ یعنی به عنوان مثال اگر مدت زمان انجام animation روی یک عنصر 6s تعریف شده باشد در این صورت 50% یعنی ۳ ثانیه پس از شروع animation.
- ✓ بلاک پایان animation : برای بلاک پایان animation می توان از مقدار 100% و یا کلمه کلیدی to استفاده کرد.

مثال :

```
@keyframes changecolor {

   from {
      background-color : red;
   }

 50% {
      background-color : green;
   }

   to {
      background-color : red;
   }

}
```

دستور فوق یک animation سه مرحله ای با نام changecolor را تعریف می کند؛ در ابتدای کار و در شروع animation رنگ پس زمینه عنصر قرمز می باشد و با رسیدن به میانه animation رنگ پس زمینه به رنگ سبز تغییر پیدا کرده و سپس دوباره شروع به بازگشت به رنگ قرمز می کند و در نهایت رنگ پس زمینه به قرمز تغییر پیدا کرده و animation به پایان می رسد.
مراحل شروع و پایان animation فوق دارای مقادیر خصوصیات کاملاً یکسان می باشند و می توان این مراحل را به صورت زیر باهم ترکیب کرد:

```
@keyframes changecolor {

   from, to {
      background-color : red;
   }

 50% {
      background-color : green;
   }

}
```

در این حالت مرورگر ابتدا به طور خودکار keyframe ها یا بلاک ها را به صورت صعودی مرتب کرده و سپس animation را اجرا می کند.

مطلب گفته شده در مورد نحوه استفاده از خاصیت ها در transition در اینجا نیز صدق می کند. در اینجا نیز برای ایجاد یک animation با استفاده از keyframes روش استاندارد و درست این است که خاصیت های مورد استفاده در داخل بلاک keyframe را به صورت صریح تعریف کنیم و از استفاده از حالت کوتاه نویسی عناصر خودداری کنیم. به عنوان مثال برای animate کردن margin یک عنصر از خاصیت های margin-left , margin-right , margin-top و margin-bottom به جای حالت کوتاه نویسی آن (یعنی margin) استفاده کنیم.
در ادامه فصل نحوه اختصاص یک animation یا keyframes تعریف شده به یک یا چند عنصر خاص توضیح داده خواهد شد. فعلاً به توضیح باقی خصوصیات و ویژگی های مربوط به keyframe می پردازیم.

## animation-timing-function

با استفاده از این خاصیت که مشابه خاصیت transition-timing-function در transition می باشد، می توان برای هر مرحله یا هر keyframe از یک animation یک timing (آهنگ انجام animation ) تعریف کرد. تمامی مطالب گفته شده در مورد خاصیت transition-timing-function در فصل مربوط به transition در مورد animatin-timing-function نیز صدق می کند و قابل استفاده می باشد.
از خاصیت فوق می توان در تمامی keyframe ها به غیر از keyframe آخر (to=100%) استفاده کرد. اگر از این خاصیت برای keyframe آخر نیز استفاده شود در این صورت این خاصیت نادیده گرفته خواهد شد.

مثال :

```
@keyframes changecolor {

   from {
      background-color : red;
      animation-timing-function : ease-out;
   }

 50% {
      background-color : green;
      animation-timing-function : ease-in;
   }

   to {
      background-color : blue;
   }

}
```

در مثال فوق از شروع تا میانه animation (تغییر رنگ پس زمینه از قرمز به سبز) با آهنگ ease-out انجام شده و از میانه تا انتهای animation (تغییر رنگ پس زمینه از سبز به آبی) نیز با آهنگ ease-in انجام می شود.

بعد از تعریف کردن keyframes های موردنظر، می توان یک animation را با استفاده از خصوصیاتی که در ادامه فصل بررسی خواهند شد به عنصر یا عناصر html موردنظر اختصاص داد.

## animation-name

از این خاصیت برای اختصاص یک animation تعریف شده با استفاده ساختار @keyframes به عناصر html استفاده می شود.

قاعده کلی :

```
animation-name : <keyframes-animation-name>;
```

مقدار پیش فرض این خاصیت none می باشد.
<keyframes-animation-name> نام یک animation تعریف شده با استفاده از قاعده @keyframes می باشد.

## animation-duration

این خاصیت مدت زمان انجام یک animation را مشخص می کند.

قاعده کلی :

```
animation-duration : <time>;
```

مقادیر قابل استفاده در این خاصیت مدت زمان برحسب ثانیه (s) یا میلی ثانیه (ms) می باشد . مقدار پیش فرض برای این خاصیت مقدار صفر است و در نتیجه اگر این خاصیت برای یک عنصر تعریف نشود animation انجام نخواهد شد. در صورت استفاده از مقادیر منفی برای این خاصیت مقدار منفی برابر صفر در نظر گرفته خواهد شد. به هریک از مراحل animation تعریف شده در قاعده @keyframes متناسب با مقادیر درصدهای تعریف شده برای هر keyframe بخشی از زمان مشخص شده در خاصیت animation-duration اختصاص داده می شود؛ در نتیجه مجموع تمامی مراحل animation در مدت زمان مشخص شده در خاصیت animation-duration اجرا خواهند شد و نه هر مرحله از animation به صورت جداگانه.

برای اعمال یک animation روی یک عنصر html باید حداقل دو خاصیت animation-name و animation-duration برای عنصر موردنظر تعریف شده باشند.

مثال :

```
@keyframes changecolor {

  from {
     background-color : red;
     animation-timing-function : ease-out;
  }

 50% {
     background-color : green;
     animatin-timing-function : ease-in;
  }

  to {
     background-color : blue;
  }
```

```
}
p:hover{
animation-name : changecolor;
animation-duration : 4s;
}
```

در مثال فوق ابتدا با استفاده از قاعده @keyframes یک animation به نام changecolor را تعریف کرده و سپس با استفاده از خاصیت animation-name ، animation تعریف شده را به عنصر p اختصاص داده ایم. مدت زمان انجام animation را نیز با استفاده از خاصیت animation-duration برابر 4 ثانیه قرار داده ایم.
در این مثال animation در صورتی اجرا خواهد شد که اشاره گر موس روی عنصر موردنظر قرار بگیرد(:hover). اگر شبه کلاس :hover در این مثال تعریف نمی شد animation بلافاصله پس از load شدن صفحه شروع به اجرا می کرد.

یکی از تفاوت های animation های ایجاد شده با این روش با روش transtion در این است که در این روش animation حالت معکوس یا reverse ندارد و برای اینکه animation به صورت معکوس نیز اجرا شود باید با استفاده از keyframe های دیگر حالت معکوس برای animation تعریف شود. تفاوت دیگری که نیز وجود دارد این است که بر خلاف روش transition در این روش می توان animation های چند مرحله ای ایجاد کرد. همچنین در این روش می توان یک animation یکسان را به چند عنصر html مختلف اختصاص داد و همچنین می توان چندین animation را به طور همزمان به یک عنصر html اختصاص داد.

مثال : اختصاص یک animation به چند عنصر html مختلف

```
...
p:hover, h1:hover{
animation-name : changecolor;
animation-duration : 4s;
}
div:hover{
animation-name : changecolor;
animation-duration : 6s;
}
```

مثال : اختصاص چند aninmation مختلف به یک عنصر html

```
...
p:hover{
animation-name : changecolor , highlight;
animation-duration : 4s , 8s;
}
```

در مثال فوق دو animation به نام های changecolor با مدت زمان ۴ ثانیه و highlight با مدت زمان ۸ ثانیه به عنصر p اختصاص داده شده اند. دقت کنید که مقادیر اختصاص داده شده به animation-duration به ترتیب متناظر با مقادیر اختصاص داده شده به animation-name می باشند.
همانطور که ملاحظه می کنید می توان با اختصاص چند مقدار مختلف به خاصیت های animation-name و animaton-duration (که هر کدام از این مقادیر با یک کاما از یکدیگر جدا شده اند) animation های چندگانه روی عناصر ایجاد کرد.

## animation-timung-function

این خاصیت آهنگ حرکت یا انتقال animation از یک keyframe به keyframe بعدی را مشخص می کند.

قاعده کلی :

```
animation-timing-function : <keyword> | steps(<number>, <keyword>) | cubic-bezier(x1,y1,x2,y2);
```

تمامی مطالب گفته شده در مورد خاصیت transition-timing-function در فصل مربوط به transition در مورد animatin-timing-function نیز صدق می کند و قابل استفاده می باشد.

برای تعریف این خاصیت از دو روش زیر می توان استفاده کرد:

۱. استفاده از این خاصیت در داخل بلاک یا بدنه یک keyframe : در این حالت که پیش از این نیز توضیح داده شد برای هر یک از keyframe های داخل یک keyframes می توان یک timing جداگانه و متفاوت با keyframe های دیگر تعریف کرد.
۲. استفاده از این خاصیت به صورت جداگانه برای هر یک از عناصر html (هریک از selector ها) : در این حالت تمامی keyframe های داخل یک keyframes دارای timing یکسان می باشند و تمامی مراحل یک animation با یک آهنگ یکسان انجام خواهد شد. دقت کنید که این timing روی هر مرحله از animation به صورت مستقل تاثیر می گذارد و نه روی کل یک animation از شروع تا پایان.

اگر خاصیت animation-timing-function با استفاده از هر دو روش فوق تعریف شده باشد در این صورت اولویت اعمال timing ها به صورت زیر خواهد بود:

۱. مقدار timing تعریف شده در قاعده @keyframes بالاترین اولویت اجرا را دارد.
۲. اگر مقدار جداگانه ای برای هر مرحله از animation در قاعده @keframes تعریف نشده باشد آنگاه مقدار timing تعریف شده در style عنصر html موردنظر اعمال خواهد شد.
اگر فقط برای چند مرحله از animation و نه تمامی مراحل، timing جداگانه در داخل بلاک keyframe تعریف شده باشد، در اینصورت مراحلی که دارای timing اختصاصی نیستند timing خود را از مقدار تعریف شده در style عنصر html گرفته (و یا در صورت عدم تعریف timing در style عنصر html از مقدار پیش فرض ease استفاده می کنند) و مراحلی که دارای timing اختصاصی هستند از timing اختصاصی خود استفاده خواهند کرد.
۳. اگر timing در هیچ حالتی تعریف نشده باشد مقدار پیش فرض ease استفاده خواهد شد.

مثال :

```
@keyframes highlight {

  from {
    background-color : rgba(255,204,0,0);
  }
  50% {
    background-color : rgba(255,204,0,0.3);
  }
  to {
    background-color : rgba(255,204,0,0);
```

```
    }

}

p:hover {
    animation-name : highlight;
    animation-duration : 4s;
    animation-timing-function : ease-out;
}
```

در مثال فوق از آنجاییکه برای مراحل animation، timing جداگانه تعریف نشده تمامی مراحل animation از مقدار ease-out تعریف شده در داخل style عنصر html استفاده خواهند کرد.

مثال فوق یک مثال ساده با تعداد مراحل کم می باشد. در صورتیکه animation ما دارای تعداد مراحل بالا مثلاً ده مرحله باشد و ما بخواهیم برای تمامی مراحل به غیر از مرحله اول و آخر timing متفاوت با سایر مراحل مثلاً timing خطی(linear) استفاده کنیم، باید timing بلاک keyframe اول و keyframe یکی مانده به آخر را در داخل بدنه keyframe تعریف کرده و timing سایر مراحل را با استفاده از style تعریف شده برای عنصر html مشخص کنیم؛ یعنی در animation های چند مرحله ای و پیچیده باید از ترکیبی از دو روش موجود برای تعریف timing استفاده کنیم.

## animation-iteration-count

به طور پیش فرض یک animation فقط یک بار اجرا شده و سپس متوقف می شود. با استفاده از خاصیت animation-iteration-count می توان تعداد دفعات انجام یک animation را کنترل کرد.

قاعده کلی :

```
animation-iteration-count : <number> | infinite;
```

- ✓ number : با اختصاص مقادیر عددی به این خاصیت می توان تعداد دفعات انجام animation را کنترل کرد. مقادیر اعشاری نیز در این خاصیت قابل استفاده می باشند که در این صورت فقط بخشی از animation اجرا خواهد شد.
- ✓ infinite : با اختصاص کلمه کلیدی infinite یه این خاصیت animation به تعدا دفعات نامحدود و به صورت بی نهایت اجرا خواهد شد.

مثال :

```
@keyframes highlight {

    from {
        background-color : rgba(255,204,0,0);
    }
    50% {
        background-color : rgba(255,204,0,0.3);
    }
    to {
        background-color : rgba(255,204,0,0);
    }
```

```
}

p.highlight {
    animation-name : highlight;
    animation-duration : 4s;
    animation-timing-function : ease-out;
    animation-iteration-count : 3;
}
```

در مثال فوق بلافاصله پس از load شدن صفحه animation تعریف شده با نام highlight با مدت زمان ۴ ثانیه و با آهنگ ease-out روی عنصر p که دارای کلاس highlight می باشد اجرا شده و پس از سه بار اجرا متوقف می شود.

## animation-direction

با استفاده از این خاصیت می توان ترتیب انجام مراحل یک animation را کنترل کرد.

قاعده کلی :

```
animation-direction : <keyword>;
```

- ✓ normal : مقدار پیش فرض این خاصیت normal می باشد. در این حالت تمامی تکرارهای animation به همان ترتیب تعریف شده در قاعده @keyframes ، با شروع از from تا to اجرا خواهند شد. منظور از تکرارهای animation همان تعداد دفعات تکرار animation می باشد که در خاصیت animation-iteartion-count تعریف شده است.
- ✓ reverse : این حالت عکس حالت normal می باشد و تمامی تکرارهای animation در خلاف جهت تعریف شده در قاعده @keyframes ، با شروع از to تا from اجرا خواهند شد.
- ✓ alternate : اگر تعداد تکرارهای animation یعنی مقدار خاصیت animation-iteration-count بزرگتر از یک باشد مقدار alternate باعث می شود که با شروع از ۱ تمامی تکرارهای فرد animation (1,3,5,..) به صورت normal و تمامی تکرارهای زوج animation (2,4,6,...) به صورت reverse اجرا شوند؛ یعنی تکرارهای animation به طور یک در میان به صورت normal و reverse انجام می شوند.
- ✓ alternate-reverse : این حالت برعکس alternate می باشد و تمامی تکرارهای فرد animation به صورت reverse و تمامی تکرارهای زوج animation به صورت normal انجام خواهد شد؛ یعنی تکرارهای animation به طور یک در میان به صورت reverse و normal انجام می شوند.

بنابر مطالب گفته شده در حالت های normal و reverse ، animation همواره در یک جهت اجرا می شود؛ در حالت normal ، animation همواره باشروع از from تا to اجرامی شود و در حالت reverse همواره با شرع از to تا from . در حالت های alternate و alternate-reverse ، animation به صورت یک در میان در جهت های normal و reverse اجرا می شود؛ در حالت alternate ابتدا normal و سپس reverse ، و در حالت alternate-reverse ابتدا reverse و سپس normal .

مثال :

```
@keyframes highlight {

    from {
```

```
    background-color : rgba(255,204,0,0);
  }
  50% {
    background-color : rgba(255,204,0,0.3);
  }
  to {
    background-color : rgba(255,204,0,0);
  }

}

p:hover {
  animation-name : highlight;
  animation-duration : 4s;
  animation-timing-function : ease-out;
  animation-iteration-count : 2;
  animation-direction : alternate;
}
```

در مثال فوق با قرار گرفتن اشاره گر موس روی عنصر موردنظر animation برای بار اول در جهت normal با شروع از from تا to و برای بار دوم در جهت reverse با شروع از to تا from اجرا خواهد شد.

## animation-play-state

از این خاصیت برای توقف (مکث : pause) یک animation در حال اجرا استفاده می شود.

قاعده کلی :

```
animation-play-state: <keyword>;
```

- ✓ running : مقدار پیش فرض running می باشد و باعث اجرای animation به صورت عادی می شود. اگر animation قبلاً متوقف شده باشد با اعمال این خاصیت animation از همان نقطه متوقف شده شروع به اجرا خواهد کرد.
- ✓ paused : مقدار paused باعث توقف اجرای animation در همان نقطه جاری می شود. پس از اجرای مجدد (اعمال مقدار running) animation دوباره از همان نقطه متوقف شده شروع به اجرا می کند.

مثال :

```
@keyframes changecolor {

  from {
    background-color : red;
    animation-timing-function : ease-out;
  }

 50% {
    background-color : green;
    animation-timing-function : ease-in;
  }
```

```
   to {
      background-color : blue;
   }

}
#coloranimation{
animation-name : changecolor;
animation-duration : 4s;
animation-timing-function : ease-out;
}
#coloranimation:hover{
animation-play-state : paused;
}
```

در مثال فوق بلافاصله پس از load شدن صفحه animation تعریف شده با نام changecolor شروع به اجرا شدن روی عنصر html با شناسه coloranimation (#coloranimation) می کند. با قرار گرفتن اشاره گر موس روی عنصر موردنظر (:hover) مقدار خاصیت animation-play-state از مقدار پیش فرض running به مقدار paused تغییر کره و animation متوقف می شود. با خارج شدن اشاره گر موس از روی این عنصر مقدار خاصیت animation-play-state دوباره به مقدار پیش فرض running برگشته و animation از همان نقطه ای که متوقف شده بود شروع به اجرا می کند.

نکته : برای متوقف کردن و اجرای یک animation با استفاده از این خاصیت معمولاً از کدهای javascript استفاده می شود.

## animation-delay

از این خاصیت برای ایجاد تاخیر در شروع اجرای یک animation استفاده می شود.

قاعده کلی :

```
animation-delay : <time>;
```

مقادیر قابل استفاده در این خاصیت مدت زمان برحسب ثانیه (s) یا میلی ثانیه (ms) می باشد. مقدار پیش فرض این خاصیت برابر صفر می باشد و animation بلافاصله شروع به اجرا خواهد کرد. این خاصیت فقط در اولین تکرار روی animation اعمال شده و روی باقی مراحل تاثیری ندارد.
مقادیر منفی برای این خاصیت برابر صفر در نظر گرفته خواهند شد و animation بلافاصله شروع به اجرا خواهد کرد اما نه از ابتدای بلاک from . به عنوان مثال اگر مدت زمان اجرای یک animation (animation-duration) برابر با ۴ ثانیه باشد و animation به ۴ مرحله با بازه های مساوی (from=0% , 25% , 50% , to=100%) تقسیم شده باشد و مقدار animation-delay برابر 1- ثانیه تعریف شده باشد، مرحله اول aimation (from=0%) اجرا نشده و animation از ابتدای مرحله دوم (25%) شروع به اجرا خواهد کرد.

مثال :

```
@keyframes changecolor {

   from {
     background-color : red;
   }
```

```
  25% {
     background-color : green;
   }

 50% {
     background-color : yellow;
  }

  to {
     background-color : blue;
  }
}
#coloranimation{
animation-name : changecolor;
animation-duration : 4s;
animation-iteration-count : 3;
animation-delay : -1s;
}
```

به هریک از مراحل animation متناسب با درصد های تخصیص داده شده بخشی از زمان تعریف شده در خاصیت animation-duration اختصاص داده می شود. در مثال فوق animation به چهار مرحله با keyframe های مساوی (25%) تقسیم شده است. در نتیجه به هر مرحله از animation یک چهارم از زمان کل animation یعنی ۱ ثانیه تخصیص داده می شود. بخشی از animation که باید در ۱ ثانیه اول اجرا می شد (بلاک from) به دلیل تخصیص مقدار منفی یک ثانیه (-1s) به خاصیت animation-delay اجرا نشده و بلافاصله باقی animation اجرا خواهد شد. این اتفاق فقط در اولین تکرار اجرای animation رخ داده و باقی تکرارها به صورت عادی اجرا خواهد شد.

## animation-fill-mode

با استفاده از این دستور می توان مقادیر خاصیت های یک عنصر html را قبل از شروع animation و بعد از پایان animation کنترل کرد. به عنوان مثال در حالت عادی پس از پایان یک animation مقادیر خاصیت های عنصر به حالت اولیه و قبل از شروع animation باز می گردد. حال اینکه با استفاده از خاصیت animation-fill-mode می توان تعیین کرد که پس از پایان animation خاصیت های عنصر موردنظر دارای چه مقادیری باشند.

قاعده کلی :

```
animation-fill-mode : <keyword>;
```

- ✓ none : مقدار پیش فرض none می باشد و پس از پایان animation خاصیت های عنصر html به مقدار اولیه خود باز می گردند.
- ✓ forwards : با استفاده از این کلمه کلیدی خاصیت های عنصر موردنظر مقادیری را که در پایان animation و در keyframe آخر (to=100%) به آنها تخصیص داده شده است را حفظ می کنند.
- ✓ backwards : اگر animation با تاخیر شروع شود (باستفاده از خاصیت animation-delay) در این حالت تا زمانیکه animation تاخیر دارد و شروع نشده است، مقادیر خاصیت های تعریف شده در اولین keyframe (from) به عنصر موردنظر اعمال خواهد شد.

- ✓ both : هر دو حالت forwards و backwards را همزمان اعمال می کند؛ یعنی اگر animation با تاخیر شروع شود در این حالت تا زمانیکه animation تاخیر دارد و شروع نشده است، مقادیر خاصیت های تعریف شده در اولین keyframe (from) به عنصر موردنظر اعمال خواهد شد و پس از پایان animation مقادیر خاصیت های این عنصر برابر با مقادیر تعریف شده در keyframe آخر (to) خواهد بود.

مثال :

```
@keyframes changecolor {

    from {
      background-color : red;
    }
   25% {
      background-color : green;
    }

  50% {
      background-color : yellow;
   }

   to {
      background-color : blue;
   }
}
#coloranimation{
animation-name : changecolor;
animation-duration : 4s;
animation-iteration-count : 3;
animation-delay : 3s;
animation-fill-mode: both;
}
```

در مثال فوق animation با ۳ ثانیه تاخیر شروع می شود. قبل از شروع animation و به مدت ۳ ثانیه رنگ پس زمینه عنصر به رنگ قرمز (مقدار تعریف شده در from) می باشد. animation در طول مدت زمان ۴ ثانیه اجرا شده و به پایان می رسد و پس از پایان animation رنگ پس زمینه عنصر مقدار پایانی خود یعنی رنگ آبی (مقدار تعریف شده در to) را حفظ می کند.

## animation

خاصیت animation حالت کوتاه نویسی برای خاصیت های ذکر شده قبلی در مورد css animation می باشد و این خاصیت ها را باهم ترکیب می کند. در این خاصیت می توان از تمامی خاصیت های ذکر شده قبلی در مورد animation به غیر از خاصیت animation-play-state استفاده کرد.

قاعده کلی :

```
animation : <animation-name> <animation-duration> <animation-delay> <animation-timing-function>     <animation-iteration-count> <animation-direction> <animation-fill-mode>;
```

مقادیر استفاده شده در این خاصیت باید با یک space از یکدیگر جدا شوند. به غیر از مقادیر زمانی duration و delay ترتیب استفاده از سایر مقادیر در این خاصیت مهم نیست. مشابه خاصیت transition در حالت کوتاه نویسی animation نیز اولین مقدار زمانی بیانگر duration و دومین مقدار زمانی بیانگر delay می باشد.

مثال :

```
@keyframes changecolor {

    from {
      background-color : red;
    }
   25% {
      background-color : green;
    }

  50% {
      background-color : yellow;
   }

   to {
      background-color : blue;
   }
}
#coloranimation{
animation : changecolor 4s 3s ease 2 alternate both;
}
```

در مثال فوق ۳ ثانیه پس از load شدن صفحه animation تعریف شده با نام changecolor روی عنصر با شناسه coloranimation با تعداد تکرار دو بار با آهنگ ease برای تمامی مراحل animation و به صورت رفت و برگشت اجرا خواهد شد.

## ایجاد animation های چندگانه

برای ایجا همزمان چندین animation روی یک عنصر از دو روش می توان استفاده کرد.

**روش اول :**

در این روش برای ایجاد همزمان چندین animation روی یک عنصر html، تمامی خاصیت های ذکر شده قبلی در مورد animation می توانند دارای چندین مقدار باشند که هر مقدار باید با یک کاما (,) از مقدار قبلی جدا شده باشد.

اگر مقادیر کمتری از تعداد مقادیر نسبت داده شده به خاصیت animation-name به سایر خاصیت ها نسبت داده شود آنگاه مرورگر مقادیر را به صورت متناوب و در یک چرخه برای باقی animation اعمال خواهد کرد (مشابه transition). مقادیر اضافی تعریف شده در یک خاصیت نادیده گرفته خواهند شد.

مثال :

```
@keyframes changecolor {

   from {
```

```
    background-color : red;
  }

 50% {
    background-color : green;
  }

  to {
    background-color : blue;
  }

}
@keyframes changewidth {

  from {
    width : 300px;
  }

 50% {
    width : 500px;
  }

  to {
    width : 300px;
  }

}

div.animation{
animation-name : changecolor , changewidth;
animation-duration : 4s , 6s;
animation-timing-function : ease , ease-out;
animation-delay : 0s , 2s;
animation-fill-mode : forwards , backwards;
}
```

در مثال فوق به محض load شدن صفحه دو animation به نام های changecolor و changewidth روی عنصر div با کلاس animation اجرا می شوند.

**روش دوم :**

در این روش از حالت کوتاه نویسی خاصیت animation برای ایجاد animation چندگانه روی یک عنصر استفاده می شود. استفاده از این روش ساده تر و کوتاه تر از روش قبل می باشد.

قاعده کلی :

```
animation : <animation> , <animation> , ...
```

مثال : مثال قبل با استفاده از این روش به صورت زیر نوشته می شود :

```
/* keyframes definitions */
...
div.animation{
animation : changecolor 4s ease forwards,
```

```
            changewidth 6s 2s ease-out backwards;
}
```

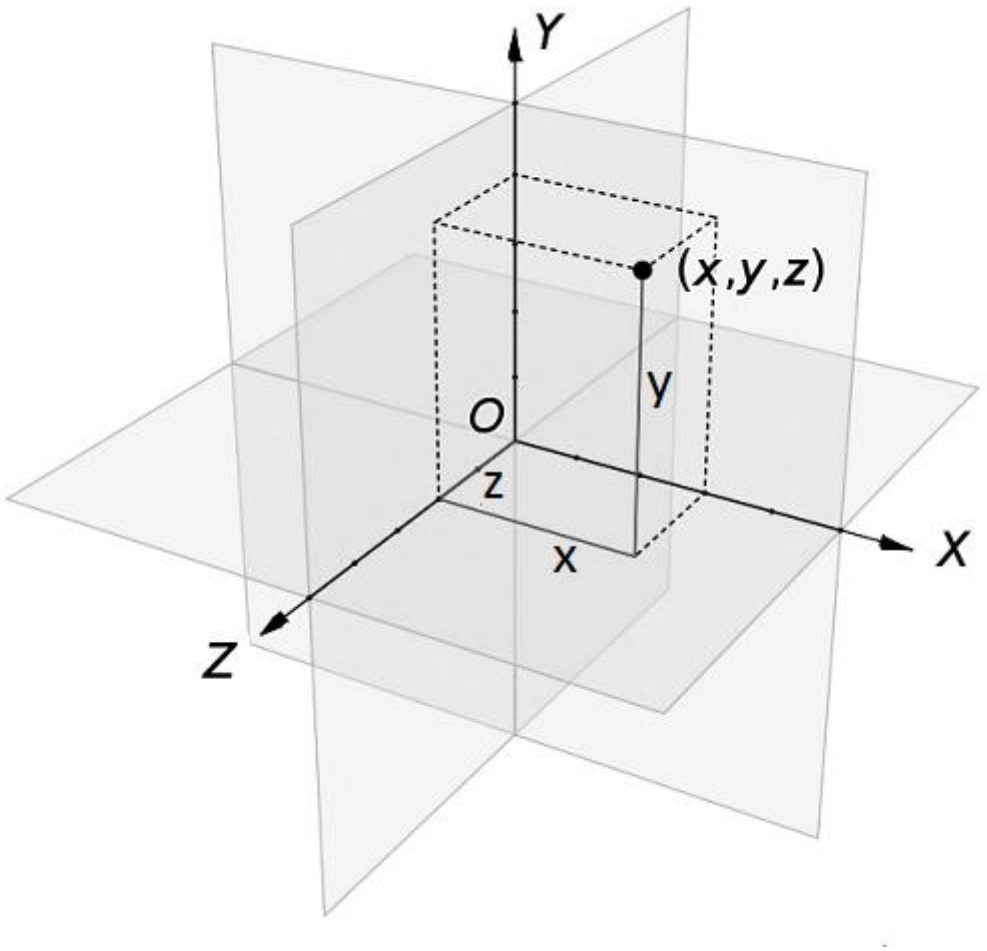
در این مثال برای animation اول (changecolor) فقط یک مقدار زمانی ذکر شده و این مقدار به عنوان animation-duration در نظر گرفته شده و مقدار animation-delay برابر صفر درنظر گرفته شده و animation بدون تاخیر اجرا خواهد شد.

در animation های چندگانه اگر delay به animation ها اختصاص داده نشود، animation ها به طور همزمان روی عنصر موردنظر اعمال خواهند شد. اما اگر delay اختصاص داده شده به یک animation برابر با مجموع duration های اختصاص داده شده به animation های قبلی باشد، animation ها پشت سرهم اجرا خواهند شد.

# فصل چهارم

# 3D Transforms

تمامی مباحث و مطالب مطرح شده تا این جا و در فصول قبلی در مورد css در فضای دو بعدی مطرح شده و مورد بحث قرار گرفته است. هر عنصر html در فضای دو بعدی دارای یک پهنا (width) و یک ارتفاع (height) می باشد، اما هیچ عمقی (depth) ندارد. تمامی مباحث در فضای دو بعدی حول محورهای x (x-axis) و y (y-axis) انجام می شود. اما با معرفی محور z (z-axis) در css3 روشی برای تغییر شکل عناصر (transforming) در فضای سه بعدی ارائه شد.




در این جا نیز مشابه 2d transform خاصیت اصلی استفاده شده برای تغییر شکل عناصر خاصیت transform می باشد.در این جا نیز با استفاده از برخی از توابع و اختصاص مقادیر به این توابع می توان نحوه و میزان تغییر شکل عناصر را مشخص کرد .

## rotation

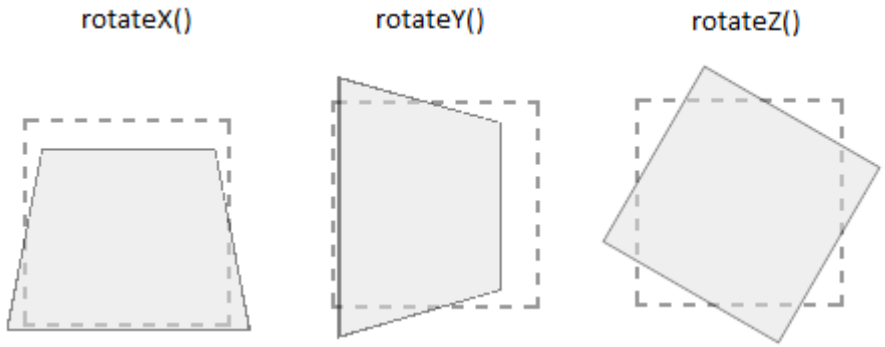
توابع ()rotate برای چرخاندن یک عنصر به اندازه ۰ تا ۳۶۰ درجه حول یکی از محورهای x ، y و یا z استفاده می شود. برای چرخش حول هر یک از محورها تابعی متناسب با همان محور استفاده می شود.

قاعده کلی :

```
transform : rotateX(<angle>);
          : rotateY(<angle>);
          : rotateZ(<angle>);
```

- ✓ rotateX : از این تابع برای چرخش عنصر حول محور X (محور افقی) استفاده می شود.
- ✓ rotateY : این تابع برای چرخش عنصر حول محور Y (محور عمودی) استفاده می شود. چرخش حول محور Y مشابه باز و بسته کردن یک در می باشد.
- ✓ rotateZ : این تابع برای چرخش عنصر حول محور Z (محور عمود بر صفحه) استفاده می شود. این تابع مشابه تابع rotate در 2d transform عمل می کند و باعث چرخش عنصر در جهت یا خلاف جهت عقربه های ساعت استفاده می شود.

استفاده از مقادیر منفی در این توابع مجاز می باشد و عناصر را در خلاف جهت نرمال می چرخاند.




به منظور درک بهتر مباحث مربوط به rotation و 3d transform ابتدا مبحث مربوط به perspective را مطرح کرده و سپس به بیان باقی مطالب مربوط به rotation می پردازیم.

## perspective

در یک فضای سه بعدی واقعی عناصر نزدیکتر به چشم بیننده بزگتر از عناصر دورتر دیده می شوند و بالعکس. به منظور ایجاد یک فضای سه بعدی واقعی برای عناصر html ، کلید اصلی استفاده از خاصیت perspective می باشد. در واقع این خاصیت پیش نیاز ایجاد حالت 3D یا سه بعدی در عناصر html می باشد.

برای تخصیص مقدار به خاصیت perspective از دو روش زیر می توان استفاده کرد :

```
perspective : <length>;

transform : perspective(<length>);
```

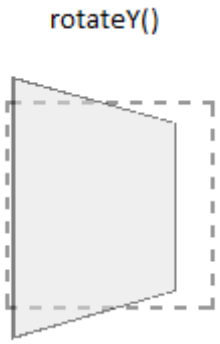
مقدار اختصاص داده شده به این خاصیت فاصله صفحه Z=0 (view point) را از چشم بیننده مشخص می کند. مقدار پیش فرض برای این خاصیت صفر می باشد و فقط مقادیر مثبت در این خاصیت قابل استفاده می باشند.
این خاصیت باعث ایجاد عمق مجازی (illusion depth) در یک عنصر می شود؛ یعنی قسمت هایی از یک عنصر که در فاصله دورتری نسبت به چشم بیننده قرار دارند کوچکتر از قسمت های نزدیک به چشم بیننده دیده می شوند.

با اختصاص خاصیت perspective به یک عنصر به تنهایی، هیچ اتفاقی روی این عنصر نمی افتد. یعنی اختصاص این خاصیت به یک عنصر روی خود عنصر هیچ تاثیری ندارد بلکه روی فرزندان آن عنصر تاثیر خواهد گذاشت، مشروط به اینکه این فرزندان یکی از توابع rotate() را پذیرفته باشند.

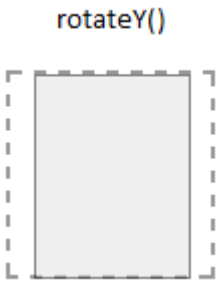
مثال :

```
…
<div class=”parent”>
    <div class=”child”></div>
</div>
…
…
. parent{
perspective : 150px;
}
. parent .child{
transform : rotateY(40deg);
}
…
```

در مثال فوق خاصیت perspective برای عنصر div با کلاس parent تعریف شده است، در نتیجه تمامی فرزندان این عنصر که از تابع rotate() استفاده می کنند به صورت perspective و با عمق مجازی نمایش داده خواهند شد.




اگر در مثال فوق از خاصیت perspective برای عنصر پدر (.parent) استفاده نکنیم، عنصر فرزند بدون عمق مجازی و به صورت زیر نمایش داده خواهد شد:




در این حالت نیز عنصر فرزند (.child) به اندازه زاویه ۴۰ درجه در راستای محور Y چرخانده شده است، اما از آنجاییکه برای عنصر پدر (.parent) از خاصیت perspective استفاده نشده است، عنصر فرزند دارای عمق مجازی نبوده به صورت سه بعدی نمایش داده نمی شود.

## ادامه مبحث rotation

پس از توضیح perspective و روشن شدن مطلب به ادامه مبحث مربوط به rotation می پردازیم.

مثال :

```
transform : rotateX(45deg);
```

مثال فوق عنصر موردنظر را به اندازه ۴۵ درجه حول محور X می چرخاند.

برای اعمال همزمان توابع ()rotate روی یک عنصر باید از روش کوتاه نویسی خاصیت transform استفاده کرد. برای این کار می توان از دو روش زیر استفاده کرد:

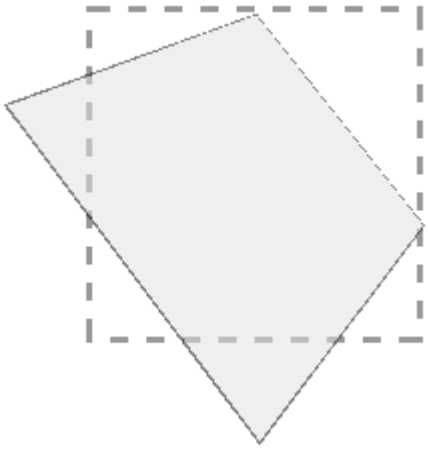
۱. استفاده همزمان از چند تابع rotate در خاصیت transform:

```
transform : rotateX(<angle>) roateY(<angle>) rotateZ(<angle>);
```

در این روش توابع استفاده شده باید با space از یکدیگر جدا شوند.

مثال :

```
transform : rotateX(30deg) rotateY(30deg) rotateZ(45deg);
```



۲. استفاده از تابع ()rotate3d :

```
transform : rotate3d(x,y,z,<angle>);
```

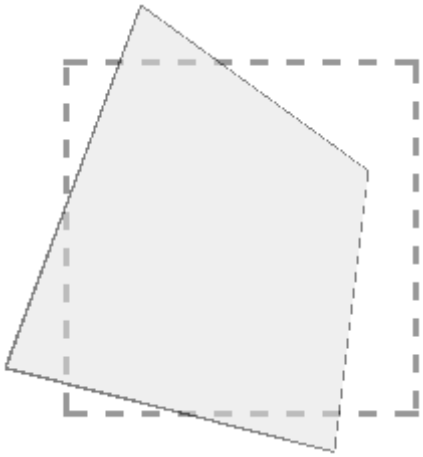
خطی رسم کنیم که از نقطه o(0,0,0) در این تابع مختصات یک نقطه در فضای سه بعدی می باشد. اگر از مبدا مختصات z و y ، x

) می چرخد. مقادیر oa حول این محور (خط (<angle>) عبور کند آنگاه عنصر موردنظر ما به اندازه زاویه تعریف شده a(x,y,z) نیز قابل استفاده می باشند.z و y ، xمنفی برای

مثال :

```
transform : rotate3d(1,2,1,45deg);
```

در این مثال اگر یک خط مستقیم از مبدا مختصات o(0,0,0) به نقطه a(1,2,1) رسم کنیم عنصر موردنظر ما به اندازه 45 درجه حول این خط می چرخد.



برخی از حالت های خاص تابع rotate3d() دارای خروجی مشابه با توابع استاندارد rotate() می باشد:

```
transform : rotate3d(1,0,0,<angle>);  ≡      transform : rotateX(<angle>);
transform : rotate3d(0,1,0,<angle>);  ≡      transform : rotateY(<angle>);
transform : rotate3d(0,0,1,<angle>);  ≡      transform : rotateZ(<angle>);
```

علامت ≡ (identical to) به معنی برابر بودن و یکسان بودن می باشد.

هنگام استفاده از دو مقدار غیر صفر و یکسان در تابع roate3d() مقدار عددی استفاده شده مهم نبوده و در صورت استفاده از هر عددی نتیجه و خروجی کار یکسان خواهد بود:

```
rotate3d(1,1,0,<angle>);   ≡   rotate3d(10,10,0,<angle>);   ≡
rotate3d(n,n,0,<angle>);

rotate3d(1,0,1,<angle>);   ≡   rotate3d(10,0,10,<angle>);   ≡
rotate3d(n,0,n,<angle>);

rotate3d(0,1,1,<angle>);   ≡   rotate3d(0,10,10,<angle>);   ≡
rotate3d(0,n,n,<angle>);
```

همچنین هنگام استفاده از یک مقدار غیر صفر در تابع roate3d() (گردش حول یکی از محورهای Y،X و Z ) مقدار عددی استفاده شده مهم نبوده و در صورت استفاده از هر عددی نتیجه و خروجی کار یکسان خواهد بود:

```
rotate3d(1,۰,0,<angle>);   ≡   rotate3d(n,۰,0,<angle>);   ≡
rotateX(<angle>);

rotate3d(0,1,0,<angle>);   ≡   rotate3d(0,n,0,<angle>);   ≡
```

```
rotateY(<angle>);

rotate3d(0,٠,1,<angle>);    ≡    rotate3d(0,٠,n,<angle>);    ≡
rotateZ(<angle>);
```

## transform-style

اگر یک عنصر فرزند که به صورت perspective و با عمق مجازی نمایش داده شده است خود نیز دارای فرزند دیگری باشد و برای این فرزند دوم نیز تابع rotate() را در نظر گرفته باشیم، در این صورت perspective فرزند دوم را می توان با استفاده از خاصیت transform-style با دقت بیشتری کنترل کرد.

قاعده کلی :

```
transform-style : flat | preserved-3d;
```

مقدار پیش فرض کلمه کلیدی flat می باشد.

در مثال زیر کاربرد و مفهوم هریک از این کلمات کلیدی توضیح داده شده است.

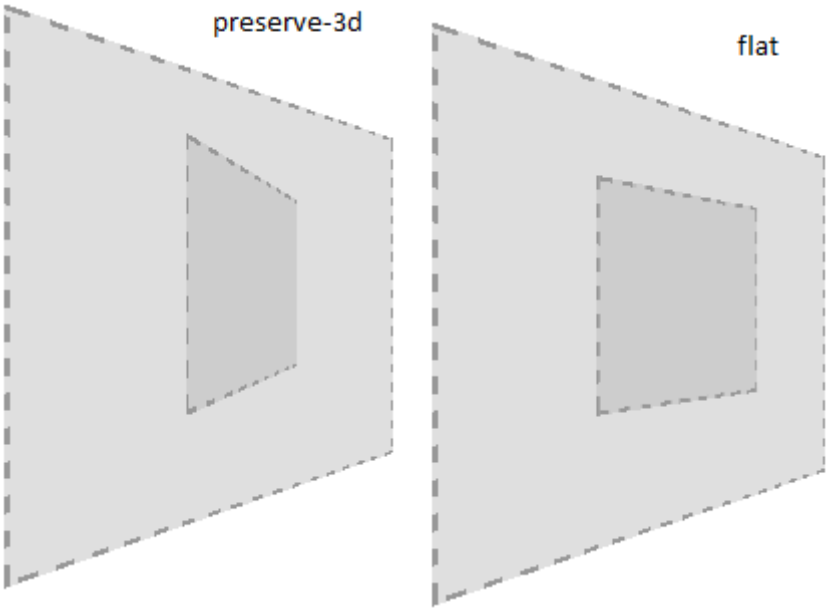
مثال :

```
…
<div class=”parent”>
    <div class=”first-child”>
        <div class=”second-child”>
        </div>
    </div>
</div>
…
…
.parent{
perspective : 150px;
}
.first-child{
transform : rotateY(30deg);
transform-style : preserved-3d;
}
.second-child{
transform : rotateY(40deg);
}
…
```

در مثال فوق خاصیت perspective برای عنصر با کلاس parent تعریف شده است و برای فرزند اول این عنصر با کلاس first-child نیز تابع rotateY() را تعریف کرده ایم که باعث می شود این عنصر (first-child) با عمق مجازی و به صورت perspective (دورنما) نمایش داده شود.

در اینجا عنصر first-child نیز خود دارای عنصر فرزندی با کلاس second-child می باشد که برای این عنصر نیز تابع roateY() تعریف شده است. اگر خاصیت transform-style را برای عنصر first-child تعریف نکنیم (و یا مقدار flat را برای آن تعریف کنیم) در این صورت عنصر second-child در عمق مجازی خود از عنصر پدر خود یعنی first-child تبعیت کرده و هر دو عنصر دارای perspective یکسان خواهند بود. اما اگر مقدار خاصیت transform-style را برای عنصر first-child به صورت preserved-

3d تعریف کنیم، انگاه عنصر فرزند آن یعنی second-child دارای عمق مجازی مستقل بوده و perspective به صورت مستقل روی عنصر second-child اعمال خواهد شد.




## perspective-origin

این خاصیت همراه با خاصیت perspective استفاده شده و زاویه و مبدا نگاه به یک عنصر 3D را مشخص می کند.

قاعده کلی :

```
perspective-origin : <x-position> <y-position>;
perspective-origin: [<percentage> | <length>] | [left| center | right]] |
[[top | center | bottom]
```

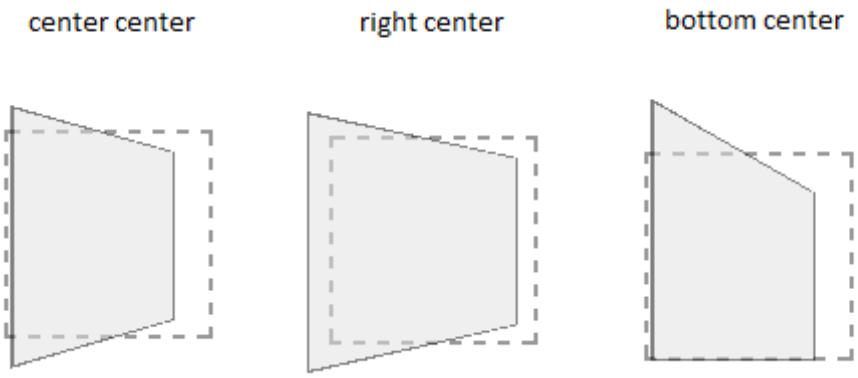
x-position مقدار افقی و y-position مقدار عمودی را مشخص می کند. مقادیر استفاده شده در این خاصیت مشابه خاصیت background-position می باشد و می توان با استفاده از کلمات کلیدی left,right,center برای x-position و top,bottom,center برای y-position ، مقادیر مطلق (absolute) مثل pixel ، مقادیر وابسته (relative) مثل ems و یا مقادیر درصدی (percentage) این خاصیت را مقداردهی کرد .

مقدار پیش فرض center center یا 50% 50% می باشد؛ یعنی عمق ایجاد شده برای عنصر بر اساس دید بیننده از مرکز عنصر می باشد.

مثال :

```
…
<div class=”parent”>
    <div class=”child”></div>
</div>
…
…
. parent{
```

```
perspective : 150px;
perspective-origin : right center;
}
. parent .child{
transform : rotateY(40deg);
}
…
```




## backface-visibility

اگر یک عنصر با استفاده از توابع rotate() به قدری چرخانده شود که پشت عنصر نمایان شود، با استفاده از خاصیت backface-visibility می توان نحوه نمایش پشت عنصر یا backface را کنترل کرد.
این خاصیت مشابه خاصیت visibility در css2 می باشد.

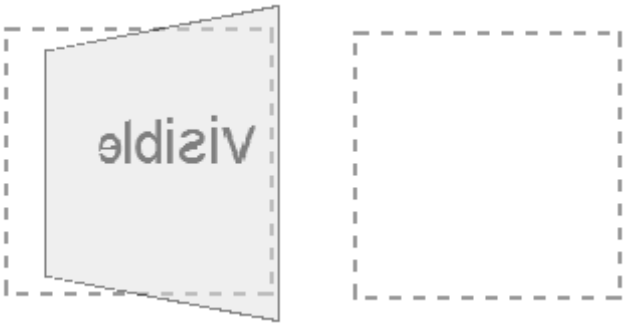
قاعده کلی :

```
backface-visibility : visible | hidden;
```

- ✓ visible : مقدار visible مقدار پیش فرض بوده و باعث می شود پشت عنصر قابل مشاهده باشد.
- ✓ hidden : این مقدار باعث می شود پشت عنصر غیر قابل مشاهده باشد؛ یعنی دقیقاً مشابه حالتی که مقدار خاصیت visibility را برابر hidden قرار می دهیم.

مثال :

```
.child{
transform : rotateY(150deg);
backface-visibility : visible;
}
```




## translation

همان طور که پیش از این در فصل مربوط به 2d transform گفته شد ،با استفاده از تابع translate می توان عناصر HTML را در فضای دو بعدی نسبت به موقعیت های پیش فرض خود در جهت های عمودی (محور Y) و افقی (محور X) جابجا کرد. در فضای سه بعدی و در 3d transform برای جابجایی عنصر در راستای محور Z باید از تابع جدید ()translateZ استفاده کرد. روش استفاده از این تابع نیز مشابه حالت دو بعدی می باشد.

قاعده کلی :

```
transform : translateX(<length>);
          : translateY(<length>);
          : translateZ(<length>);
```

در اینجا نیز استفاده از مقادیر منفی مجاز است.

در translate نیز مشابه rotate برای اعمال همزمان چند تابع روی یک عنصر باید از روش کوتاه نویسی خاصیت transform استفاده کرد. برای این کار می توان از دو روش زیر استفاده کرد:

۱. استفاده همزمان از چند تابع translate در خاصیت transform:

```
transform : translateX(<length>) translateY(<length>) translateZ(<length>);
```

در این روش توابع استفاده شده باید با space از یکدیگر جدا شوند.

مثال :

```
transform : translateX(20px) translateY(20px) translateZ(40px);
```

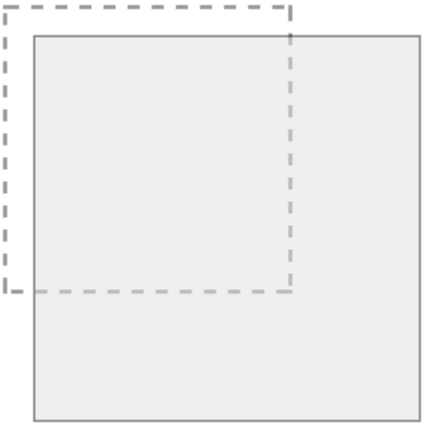
۲. استفاده از تابع translate3d() :

```
transform : translate3d(translateX,translateY,translateZ);
```

در این تابع از سه مقدار length برای مشخص کردن میزان جابجایی عنصر در راستای هریک از محورها استفاده می شود.

مثال :

```
transform : translate3d(20px,20px,40px);
```



## scaling

علاوه بر دو تابع scaleX() و scaleY() که در 2d transform معرفی شد، در 3d transform می توان از تابع جدید scaleZ() نیز برای تغییر مقیاس یک عنصر استفاده کرد.

قاعده کلی :

```
transform : scaleX(<number>);
          : scaleY(<number>);
          : scaleZ(<number>);
```

استفاده از تابع scaleZ() به تنهایی هیچ تاثیری روی عنصر نداشته و مقیاس عنصر تغییری نمی کند، زیرا یک عنصر به تنهایی هیچ عمقی (depth) ندارد. برای اینکه تابع sclaeZ() بتواند مقیاس یک عنصر را تغییر دهد باید در کنار این تابع از تابع translateZ() نیز استفاده شود. در صورت استفاده از تابع translateZ() در کنار تابع scaleZ() مقدار عددی تعریف شده در تابع scaleZ() در مقدار عددی استفاده شده در تابع translateZ() ضرب شده و عنصر موردنظر به اندازه مقدار حاصلضرب در راستای محور Z به جلو خواهد آمد؛ یعنی اگر مقدار عددی استفاده شده در تابع sclaeZ() برابر عدد n باشد آنگاه مقدار استفاده شده در تابع translateZ() ، n برابر خواهد شد.

مثال :

```
transform : scaleZ(3) translateZ(10px);
scaleZ() * translateZ() = 3 *10 = 30  =>      translateZ(30px)
```

در مثال فوق عنصر موردنظر به اندازه ۳۰ پیکسل در راستای محور Z و به سمت جلو جابجا خواهد شد.

در اینجا نیز مشابه 2d transform مقدار پیش فرض تابع برابر یک می باشد. scaleZ(0) باعث محو شدن عنصر می شود. استفاده از مقدار منفی باعث کوچکتر شدن عنصر نسبت به اندازه اصلی آن می شود.

scale3d(1,1,-3) translateZ(10px);

در scale نیز برای اعمال همزمان چند تابع روی یک عنصر باید از دو روش گفته شده برای کوتاه نویسی خاصیت transform استفاده کرد:

۱. استفاده همزمان از چند تابع scale در خاصیت transform:

```
transform : scaleX(<number>) scaleY(<number>) scaleZ(<number>);
```

در این روش توابع استفاده شده باید با space از یکدیگر جدا شوند.

مثال :

```
transform : scaleX(1.5) scaleY(2) scaleZ(3) translateZ(10px);
```

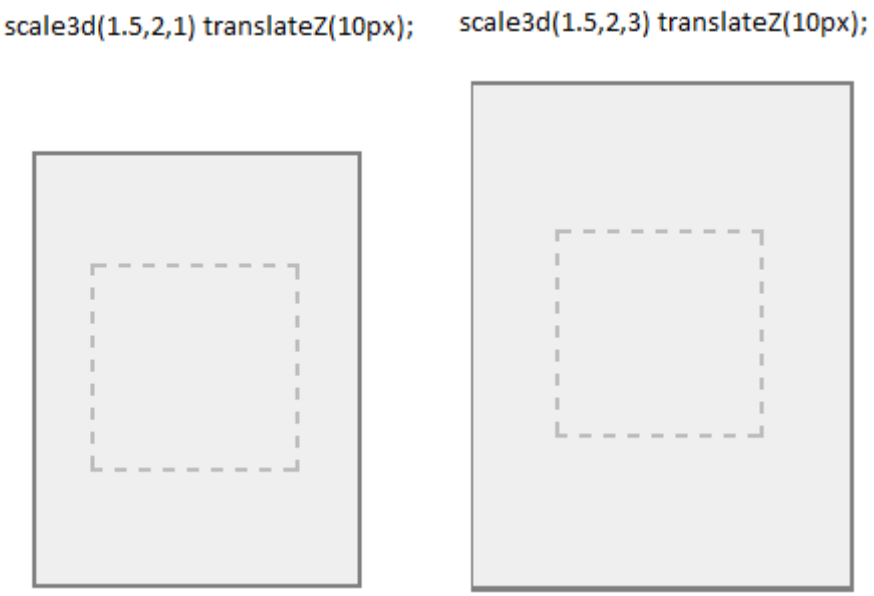
۲. استفاده از تابع scale3d() :

```
transform : scale3d(scaleX,scaleY,scaleZ);
```

در این تابع از سه مقدار number برای مشخص کردن مقیاس و اندازه جدید عنصر در هریک از ابعاد مختلف استفاده می شود.

مثال :

```
transform : scale3d(1.5,2,3) translateZ(10px);
```




## ترکیب توابع transform

مشابه 2d transform در 3d transform نیز می توان توابع مختلف را در یک خاصیت transform باهم ترکیب کرده و به کار برد. در اینجا نیز باید تمامی توابع مورد نظر را در خاصیت transform قرار داده و این توابع را با یک space از یکدیگر جداکنیم.

```
transform : function(value) function(valye) … ;
transform : rotate3d() scale3d() translate3d();
```

همانطور که در 2d transform گفته شد استفاده از خاصیت های transform به طور جاگانه برای هر تابع صحیح نمی باشد و در این حالت فقط دستور آخر روی عنصر موردنظر اعمال خواهد شد .

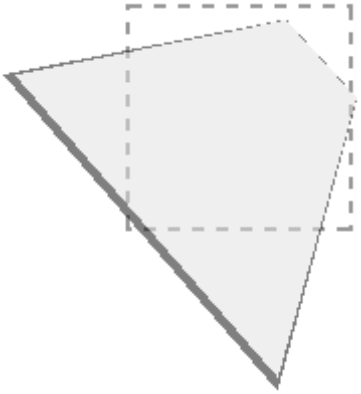
مثال :

```
transform : rotate3d(1,0,0,45deg);
transform : scale3d(1.1,1.5,3);
transform : translate3d(30px,30px,40px);
```

در مثال فوق فقط دستور آخر یعنی translate3d(30px,30px,40px) روی عنصر موردنظر اعمال خواهد شد و باقی دستورات در نظر گرفته نخواهند شد.

مثال :

```
transform: rotateX(50deg) rotateY(40deg) rotateZ(30deg)
           scale3d(0.75,2,2)
           translate3d(5px,-5px,10px);
```



روش دیگر برای ترکیب توابع transform استفاده از تابع matrix3d() می باشد که ۱۶ پارامتر می گیرد. به علت پیچیدگی این روش از بیان آن صرف نظر می کنیم .

## transform-origin

این خاصیت مشابه خاصیت transform-origin در 2d transform می باشد. تمامی مطالب گفته شده در حالت 2d در این حالت نیز صدق می کند. تنها تفاوت این حالت با حالت 2d در این است که در این حالت از سه مقدار برای مشخص کردن مبدا یا نقطه اعمال تغییرات ( transformation point ) استفاده می شود. این سه مقدار محل تلاقی محورهای X,Y,Z را مشخص می کنند. مقدار پیش فرض برای این خاصیت برابر با مرکز عنصر می باشد.

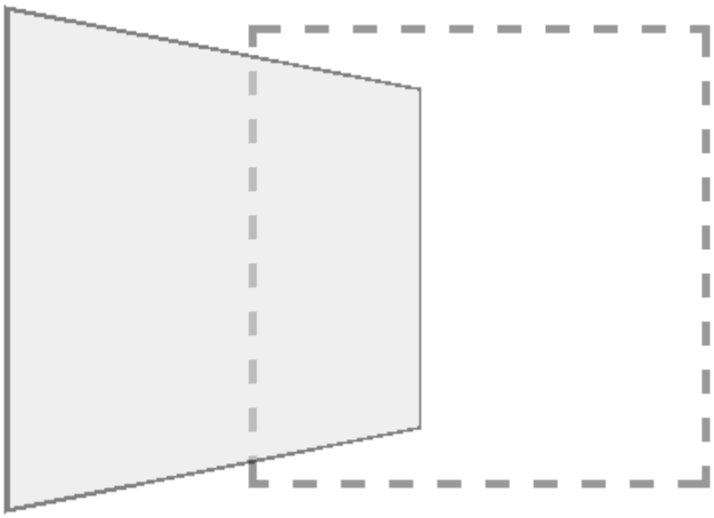
قاعده کلی :

```
transform-origin : <x-position> <y-position> <z-position>;
```

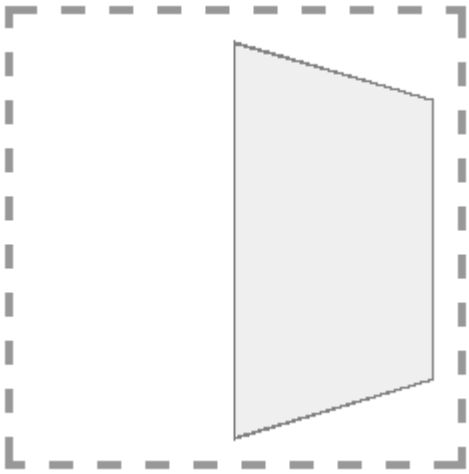
مقادیر قابل استفاده برای x-position و y-position همان مقادیر قابل استفاده در 2d transform می باشند. برای z-position فقط مجاز به استفاده از مقادیر length هستیم و نمی توان از مقادیر درصدی و کلمات کلیدی استفاده کرد.
z-position فاصله ای در امتداد محور z (محور عمود بر صفحه) است که transformation در آنجا اتفاق می افتد. اگر مقادیر منفی برای z-position اختصاص یابد، transform-origin در پشت عنصر قرار خواهد گرفت و عنصر فرزند پشت عنصر پدر خود قرار می گیرد. اما اگر مقادیر مثبت به z-position اختصاص یابد transform-origin در جلوی عنصر قرار خواهد گرفت و عنصر فرزند روی عنصر پدر خود قرار می گیرد.

مثال :

```
.child{
transform : rotateY(40deg);
transform-origin : left center 70px;
}
```



```
.child{
transform : rotateY(40deg);
transform-origin : left center -70px;
}
```



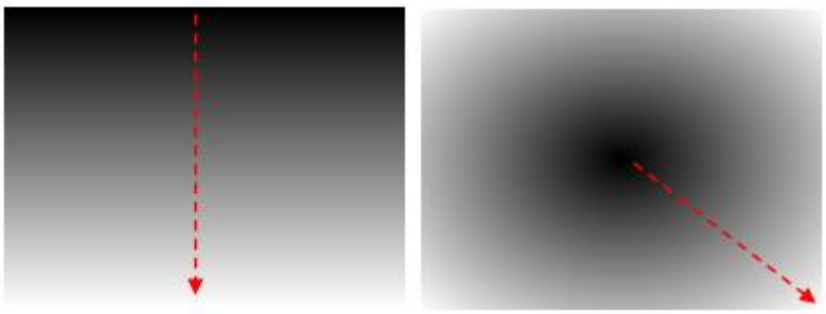
# فصل پنجم

# Gradient

gradient عبارت است از تغییر رنگ ملایم و تدریجی (gradual transition) بین دو یا چند رنگ. در گرافیک از gradient برای ایجاد عمق کاذب، دادن حجم به عناصر و ایجاد افکت های سه بعدی استفاده می شود.

در css3 از خاصیت background-image و یا حالت کوتاه نویسی خاصیت background برای ایجاد gradient در عناصر html استفاده می شود. دو نوع gradient در css3 وجود دارد که عبارتند از :

- ✓ linear-gradient : در حالت linear تغییر رنگ پس زمینه عناصر در امتداد یک خط مستقیم انجام می شود. به این خط فرضی در اصطلاح gradient line گفته می شود.
- ✓ radial-gradient : در حالت radial تغییر رنگ پس زمینه عناصر به صورت یک دایره و یا یک بیضی و در امتداد خط فرضی gradient ray که این خط همان شعاع دایره یا بیضی می باشد، انجام می شود. نقطه شروع خط gradient ray مرکز دایره می باشد.



در مرورگرها یک gradient به صورت پیش فرض عرض (width) و پهنای (height) و padding یک عنصر html را پر می کند که به این محدوده در اصطلاح gradient box گفته می شود. از آنجاییکه برای ایجاد gradient از خاصیت background-image استفاده می شود، در نتیجه برای کنترل size و position یک gradient (gradient box) می توان به ترتیب از خاصیت های background-size و background-position استفاده کرد.

## linear gradient

در css3 برای ایجاد linear-gradeint از تابع ()linear-gradient استفاده می شود. با استفاده از این تابع می توان جهت خط gradient line و رنگ های استفاده شده در یک gradient خطی را مشخص کرد.

قاعده کلی برای استفاده از تابع ()linear-gradient به صورت زیر می باشد:

```
background-image : linear-gradient(direction,color-stops);
```

همانطور که ملاحظه می کنید تابع ()linear-gradient دو پارامتر دریافت می کند، که این پارامترها با یک کاما(,) از یکدیگر جا می شوند.

- ✓ direction : این پارامتر که یک پارامتر اختیاری است، جهت و یا زاویه انجام gradient یا همان خط gradient line را مشخص می کند. در این تابع می توان از کلمات کلیدی و یا یک زاویه (<angle>) به منظور مشخص کردن جهت خط gradient line استفاده کرد.
- ✓ color-stops : این پارامتر یک لیست دو یا چند مقداری از رنگ ها می باشد که با یک کاما از یکدیگر جدا شده اند.

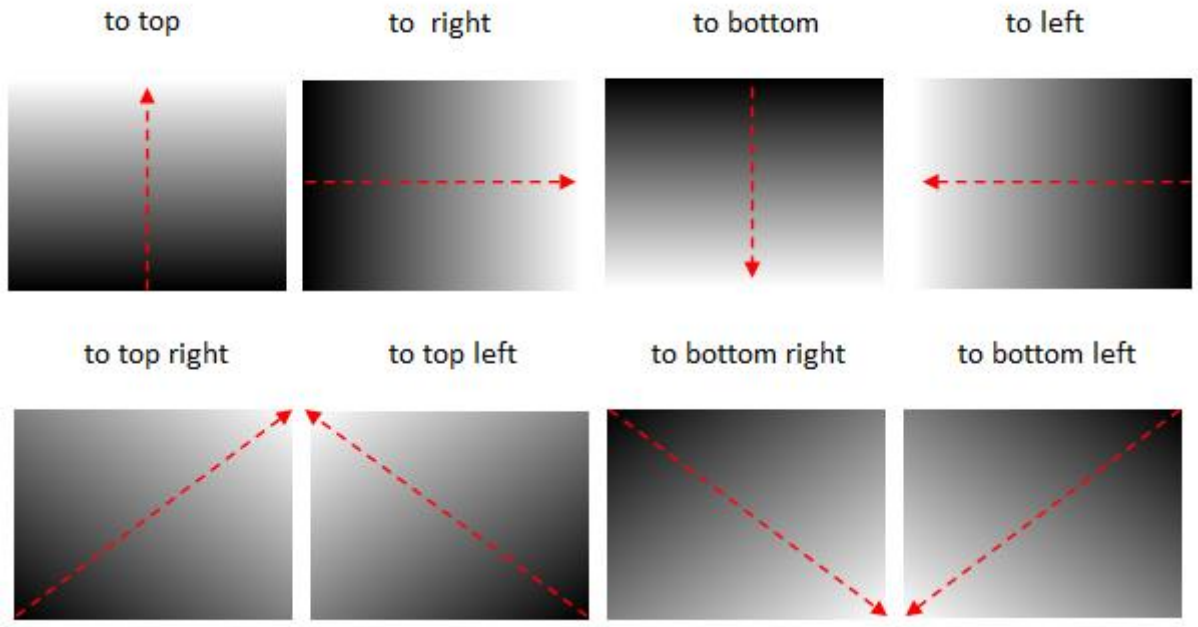
## direction

همانطور که گفته شد برای مشخص کردن جهت gradient line می توان از دو روش استفاده کرد:

۱. کلمات کلیدی
۲. استفاده از یک زاویه (<angle>)

## کلمات کلیدی

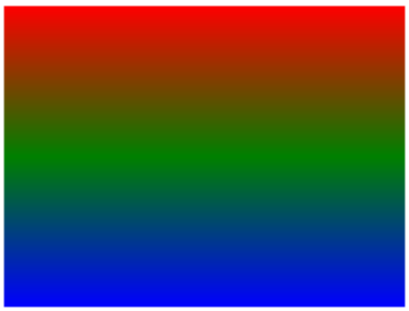
می توان ازترکیب کلمات کلیدی top,bottom,right,left برای مشخص کردن نقطه پایان gradient line یا همان جهت خط gradient line استفاده کرد. این کلمات کلیدی جهت انجام gradient را در داخل gradient box مشخص می کنند. می توان این کلمات کلیدی را باهم ترکیب کرده و گوشه gradient box (corner) را با استفاده از آنها مشخص کرد؛ مثل top right و یا botttom left . برای مشخص کردن جهت خط gradient line باید کلمه کلیدی to را به صورت پیشوند با کلمات کلیدی فوق ترکیب کرد؛ مثل to top right . فقط هشت ترکیب زیر از این کلمات کلیدی معتبر بوده و قابل استفاده می باشد :



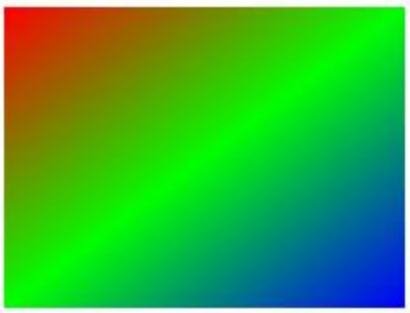

مقدار پیش فرض to bottom می باشد.

مثال :

```
background-image : linear-gradient(to bottom,red,green,blue);
```



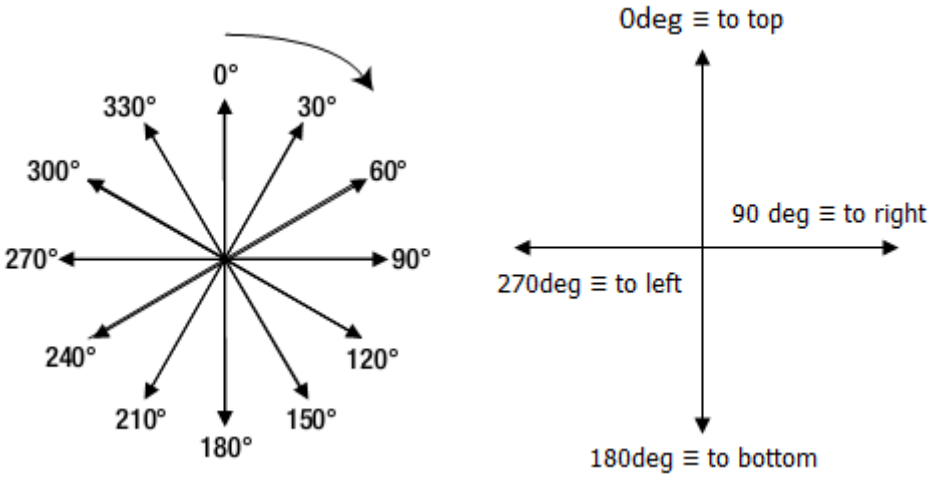
```
background-image : linear-gradient(to bottom right , #ff0000 , #00ff00 ,          #00
```



## زاویه (angle)

برای تعیین جهت خط gradient line به جای استفاده از کلمات کلیدی می توان از یک زاویه استفاده کرد. استفاده از زوایا و واحد اندازه گیری deg ساده می باشد اما در این حالت بیشتر مقادیر استفاده شده برای زوایا باعث ایجاد نقاط شروع و پایان در خارج از gradient box می شود و بخشی از gradient به خارج از محدوده gradient box گسترش پیدا می کند.

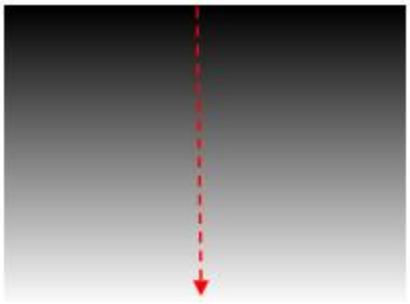
در این روش مقدار پیش فرض برای gradient line ، ۱۸۰ درجه است که برابر با یک خط مستقیم از بالا به پایین می باشد (همان حالت پیش فرض to bottom). زاویه صفر برای gradient line یک خط مستقیم از پایین به بالا می باشد (معادل حالت to top). زوایای بزرگتر از صفر درجه برای gradient line نیز در جهت عقربه های ساعت حرکت می کنند. در این روش زوایا نسبت به مرکز gradient box محاسبه می شوند و gradient line به اندازه زاویه تعریف شده در هر دو جهت امتداد پیدا می کند. مقادیر بزرگتر از 360 درجه و نیز مقادیر کمتر از صفر درجه (مقادیر منفی برای تعریف زوایا در خلاف جهت عقربه های ساعت) قابل استفاده می باشند. در این حالت مرورگر به صورت اتوماتیک مقادیر خارج از محدوده صفر تا ۳۶۰ درجه را به مقادیر بین صفر تا ۳۶۰ درجه تبدیل کرده و سپس استفاده می کند. به عنوان مثال زاویه -45deg برابر است با زاویه 315deg .



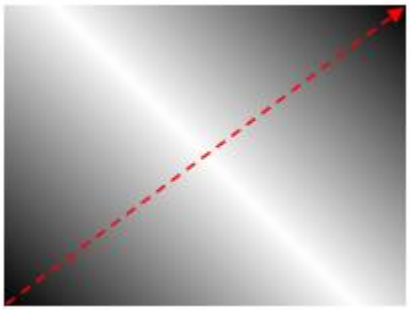

به این روش برای مشخص کردن جهت خط gradient line در اصطلاح compass coordinate system گفته می شود.

مثال :

```
background-image : linear-gradient(180deg,black,white);
```

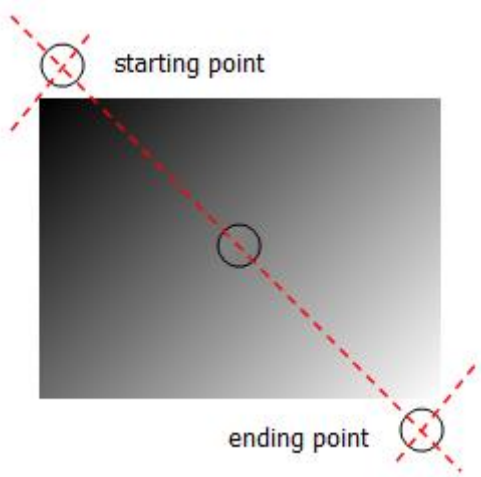


```
background-image : linear-gradient(45deg,black,white,black);
```



همانطور که پیش از این نیز گفته شد در این روش بیشتر مقادیر استفاده شده برای زوایا باعث ایجاد نقاط شروع و پایان در خارج از gradient box می شود و بخشی از gradient line در خارج از محدوده gradient box قرار گرفته و gradient به خارج از

محدوده gradient box گسترش پیدا می کند. برای محاسبه نقاط شروع و پایان gradient line در این حالات خط gradient line را با شروع از مرکز عنصر و به اندازه زاویه تعریف شده در هر دو جهت امتداد داده و از روش زیر استفاده می کنیم :




در این روش محل تقاطع خط عمود بر gradient line که این خط عمود از نزدیکترین گوشه gradient box عبور می کند، نقاط شروع و پایان خط gradient line را مشخص می کند.

## color-stops

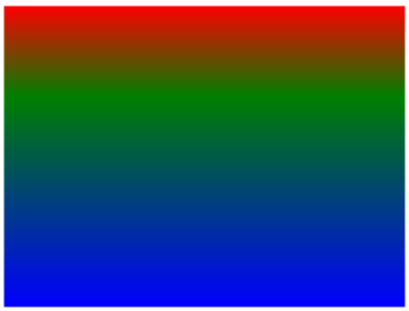
همانطور که قبلاً نیز گفته شد پارامتر color-stops در تابع ()gradient یک لیست دو یا چند مقداری از رنگ ها می باشد که باید با یک کاما از یکدیگر جدا شده باشند. به هریک از این رنگ ها یک color-stop می گویند. نقاط شروع و پایان این رنگ ها در یک gradient (موقعیت یا position رنگ ها در طول خط gradient line) را می توان به صورت اختیاری با استفاده از مقادیر length و percentage مشخص کرد. مقادیر عددی یا length با شروع از نقطه شروع gradient (ابتدای gradient box) اندازی گیری می شوند و مقادیر درصدی وابسته به نقاط شروع و پایان gradient (ابتدا و انتهای gradient box) می باشند. در حالت linear-gardient ، 0% بیانگر نقطه شروع gradient و 100% بیانگر نقطه پایان gradient می باشد و در حالت radial-gradient ، 0% بیانگر مرکز دایره یا بیضی و 100% بیانگر نقطه پایان gradient می باشد. مقادیر این position ها باید با یک space از مقادیر رنگ ها جداشود.

اگر در یک gradient موقعیت رنگ ها یا همان color-stop ها تعریف نشده باشند مرورگر به صورت پیش فرض مقادیر زیر را در نظر می گیرد:

- ✓ موقعیت (position) اولین رنگ یا color-stop برابر با ابتدای خط فرضی gradient line یعنی 0% می باشد.
- ✓ موقعیت آخرین color-stop برابر با انتهای خط gradient line یعنی 100% می باشد.
- ✓ برای color-stop های میانی طول خط gradient line به طور مساوی بین این color-stop ها تقسیم شده و به هر color-stop یک فضای مساوی در طول خط gradient line تخصیص داده می شود.

مثال :

```
background-image : linear-gradient(to bottom,red,green 30%,blue);
```



در مثال فوق gradient با رنگ red از موقعیت 0% یعنی ابتدای خط gradient line (موقعیت شروع پیش فرض) شروع شده و به تدریج رنگ پس زمینه عنصر موردنظر به به سبز تغییر کرده و در موقعیت 30% به رنگ green می رسد و سپس با شروع از موقعیت 30% رنگ پس زمینه عنصر به تدریج از سبز به آبی تغییر پیدا کرده و در موقعیت 100% (موقعیت پایانی پیش فرض) به رنگ blue می رشد.

مثال :

```
background-image : linear-gradient(to bottom,red,green 30%,yellow
65%,blue);
```

در مثال فوق gradient با رنگ red از موقعیت 0% یعنی ابتدای خط gradient line (موقعیت شروع پیش فرض) شروع شده و به تدریج رنگ پس زمینه عنصر موردنظر به به سبز تغییر کرده و در موقعیت 30% به رنگ green می رسد و سپس با شروع از موقعیت 30% رنگ پس زمینه عنصر به تدریج از سبز به زرد تغییر پیدا کرده و در موقعیت 65% به رنگ yellow می رسد و پس از آن با شروع از موقعیت 65% به تدریج از رنگ زرد به رنگ آبی تغییر کرده و در موقعیت 100% (موقعیت پایانی پیش فرض) به رنگ blue می رشد.

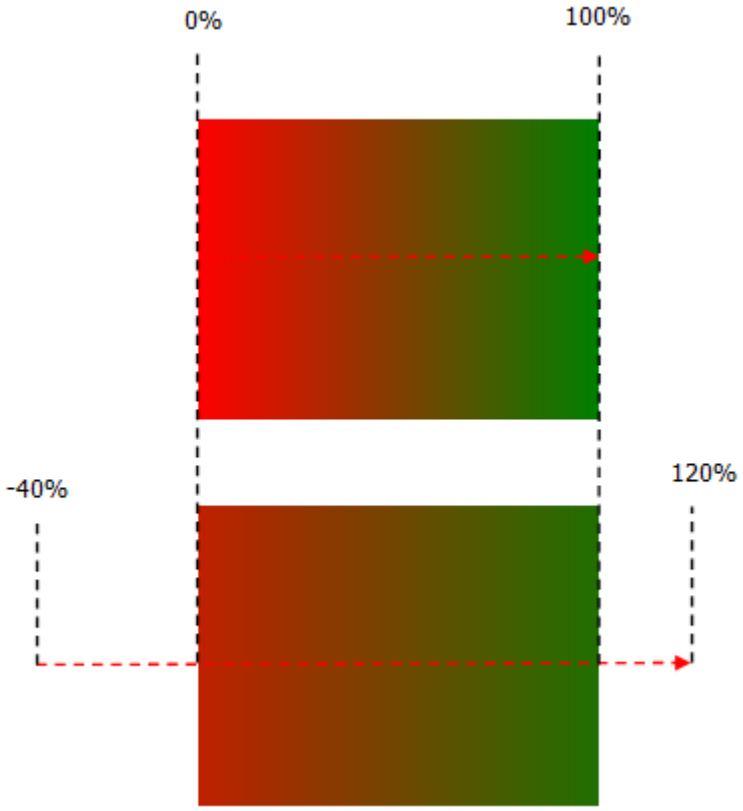
در حالت کلی محدودیتی در تعداد color-stop های استفاده شده وجود ندارد اما این color-stop ها باید براساس موقعیت خود به ترتیب صعودی مرتب شوند. اگر یک color-stop دارای موقعیتی کمتر از یکی از color-stop های قبلی باشد در این صورت موقعیت یا position این color-stop برابر بالاترین مقدار قبلی در نظر گرفته می شود.

مثال :

```
background-image : linear-gradient(to bottom,red,green 35%,yellow
25%,blue);
```

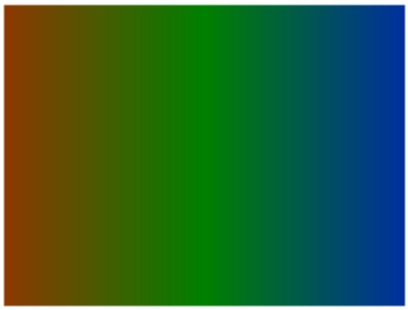
در این مثال در طول 35% ابتدای خط gradient line تغیر رنگ ملایمی از رنگ red به رنگ green را خواهیم داشت و سپس بلافاصله رنگ yellow ظاهر خواهد شد (زیرا موقعیت این رنگ برابر 25% است که کمتر از موقعیت رنگ قبل یعنی 35% می باشد) و در 65% باقی مانده طول خط gradient line تغییر رنگ ملایمی از رنگ yellow به رنگ blue را خواهیم داشت.

رنگ های استفاده شده در gradient یا همان color-stop ها به طور معمول بین نقاط شروع و پایان محدوده gradient box قرار می گیرند. اما gradient line می تواند به صورت نامحدود در هر دو جهت امتداد پیدا کند. بنابراین می توان موقعیت اولین color-stop را قبل از نقطه شروع gradient box و موقعیت آخرین color-stop را نیز بعد از نقطه پایان gradeint box تعریف کرد.




مثال :

```
background-image : linear-gradient(90deg,red -40%,green 50%,blue 130%);
```

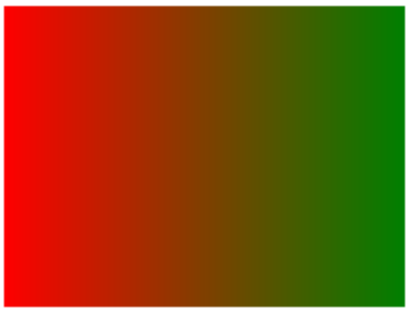


در این مثال نقاط شروع و پایان خط gradient line در خارج از محدوده gradient box قرار می گیرند.

اگر اولین color-stop دارای موقعیتی بزرگتر از 0% باشد در اینصورت شروع gradient دارای رنگ یکدست (solid) خواهد بود. به طور مشابه اگز آخرین color-stop دارای موقعیت کمتر از 100% باشد در اینصورت پایان gradient نیز دارای رنگ یکدست خواهد بود.

مثال :

```
background-image : linear-gradient(to right,red,green);
```



```
background-image : linear-gradient(to right,red 40%,green);
```

```
background-image : linear-gradient(to right,red,green 60%);
```

## radial-gradient

برای ایجاد یک radial-gradient در css3 از تابع radial-gradient() به صورت زیر استفاده می شود:

```
background-image : radial-gradient(shape size position,color-stops);
```

- ✓ shape : این پارامتر شکل radial-gradient را تعریف کرده و مشخص می کند که gradient به صورت دایره باشد و یا بصورت بیضی. برای این پارامتر دو کلمه کلیدی circle برای حالت دایره ای و کلمه کلیدی ellipse برای حالت بیضوی قابل استفاده می باشند. اگر در این پارامتر از هیچ مقداری استفاده نشود و یا فقط از یک مقدار عددی برای پارامتر size استفاده شود در این صورت مقدار پیش فرض shape برابر circle خواهد بود.
- ✓ size : این پارامتر ابعاد یک gradient و یا طول خط gradient ray را تعریف می کند. به عبارت دیگر این پارامتر اندازه دایره یا بیضی را مشخص می کند. با استفاده از مقادیر length , percentage می توان ابعاد دایره یا بیضی را به صورت دقیق مشخص کرد.

در حالت circle فقط یک مقدار length قابل استفاده است که مشخص کننده شعاع دایره است. در حالت circle به دلیل وجود ابهام استفاده از مقادیر درصدی (percentage) مجاز نیست زیرا در این صورت نمی توان مشخص کرد که مقادیر درصدی وابسته به طول gradient box می باشد یا به وابسته به ارتفاع آن.

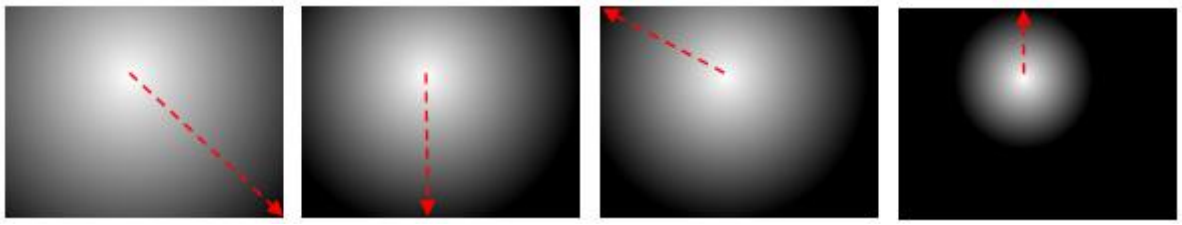
در حالت ellipse دو مقدار عددی می توان استفاده کرد که این مقادیر می توانند به هر دو صورت length و percentage باشند. مقدار اول شعاع افقی بیضی و مقدار دوم شعاع عمودی بیضی را مشخص می کند. مقادیر درصدی وابسته به طول (width) و ارتفاع gradient box (height) می باشد.

در هر دو حالت circle و ellipse می توان از کلمات کلیدی زیر نیز برای مشخص کردن size استفاده کرد:

- farthest-corner : این مقدار که مقدار پیش فرض نیز می باشد، طول خط gradient ray را به اندازه فاصله مرکز دایره از دورترین گوشه gradient box تعریف می کند. (دورترین گوشه)
- farthest-side : این مقدار شعاع gradient را به اندازه فاصله مرکز دایره از دورترین ضلع gradient box تعریف می کند. (دورترین ضلع)
- closet-corner : در این حالت شعاع gradient به اندازه فاصله مرکز دایره تا نزدیکترین گوشه gradient box می باشد. (نزدیکترین گوشه)

در هر سه حالت فوق ممکن است بخشی از gradient به خارج از محوده gradient box گسترش پیدا کند.

- closest-side : این مقدار شعاع gradient را به اندازه فاصله مرکز دایره از نزدیکترین ضلع gradient box تعریف می کند. (نزدیکترین ضلع)

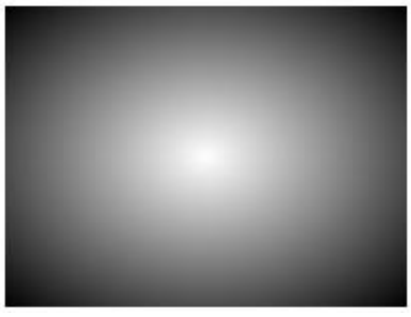


- ✓ position : این پارامتر موقعیت مرکز دایره یا بیضی را در داخل gradient box تعیین می کند. مقادیر قابل استفاده در این پارامتر دقیقاً مشابه مقادیر قابل استفاده در خاصیت background-position می باشد، با این تفاوت که قبل از ذکر مقادیر باید از کلمه کلیدی at استفاده کرد. مقدار پیش فرض این پارامتر برابر با at center می باشد و باعث قرار گرفتن دایره یا بیضی در مرکز gradient box می شود.
- ✓ color-stops : این پارامتر یک لیست دو یا چند مقداری از رنگ ها می باشد که با یک کاما از یکدیگر جدا شده اند. تمامی مطالب گفته شده در مورد color-stop در linear-gradient در radial-gradient نیز صدق می کند.

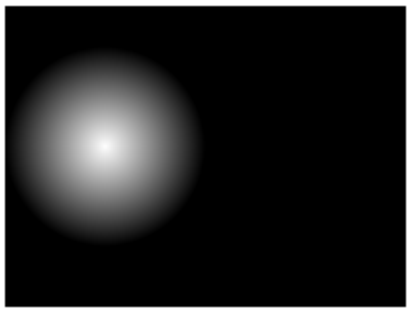
تنها پارامتر اجباری تابع radial-gradient() پارامتر color-stops می باشد و باقی پارامتر ها اختیاری است. لیست مقادیر پارامترهای shape,size و position باید با یک space از یکدیگر جدا شوند. ترتیب مقادیر shape و size هیچ اهمیتی ندارد اما مقدار position باید همواره بعد از این دو مقدار ذکر شود (مگر اینکه مقادیر shape و size استفاده نشده باشند). لیست رنگ ها (color-stops) همواره در انتها می آید و باید با یک کاما از سایر پارامترها جدا شود.

مثال :

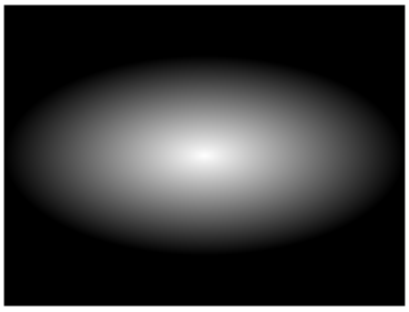
```
background-image : radial-gradient(white,black);
```



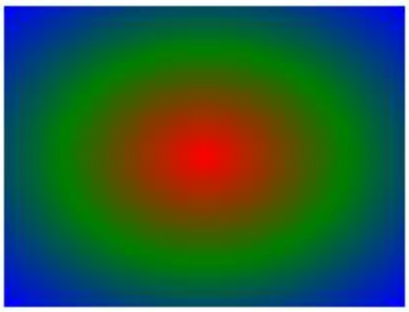
```
background-image : radial-gradient(circle closest-side at 50px
70px,white,black);
```



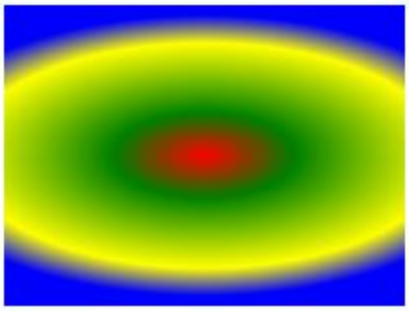
```
background-image : radial-gradient(ellipse 100px 50px,white,black);
```



```
background-image : radial-gradient(red,green,blue);
```



```
background-image : radial-gradient(ellipse 150px 70px at 50% 50% ,red,green 30%,yellow 80%,blue);
```

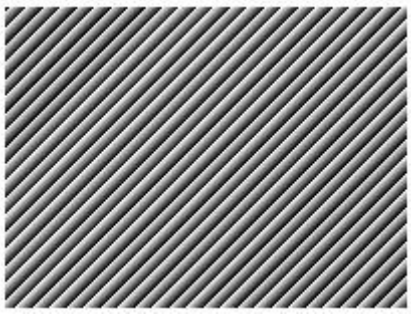


## repeating-gradient

برای پس زمینه عناصر می توان الگوهای تکراری gradient را اعمال کرد. برای ایجاد gradient های تکراری در یک عنصر از توابع repeating-linear-gradient() و repeating-radial-gardient() استفاده می شود. پارامترهای دریافتی این توابع دقیقاً همان پارامترهای استفاده شده در توابع غیرتکراری می باشند. تنها نکته ای که در این جا وجود دارد این است که برای داشتن rapeating-gardient باید موقعیت color-stop نهایی قبل از انتهای خط gradient line یا gardient ray باشد. position پیش فرض برای آخرین color-stop مقدار 100% می باشد که این مقدار پیش فرض مانع از ایجاد repeating-gradient می شود.

مثال : repeating-linear-gradient

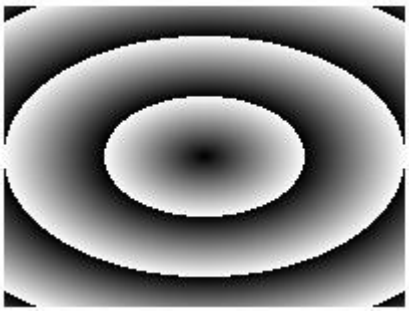
```
background-image : repeating-linear-gradient(-45deg, #000, #FFF 6px);
```



```
background-image: repeating-linear-gradient(to right, #dfdfdf, #cdcdcd 5%,
#dfdfdf 10%);
```

مثال : repeating-radial-gradient

```
background-image : repeating-radial-gradient(ellipse 50px 30px, #000, #FFF 50px);
```



```
background-image : repeating-radial-gradient(circle at 50% 100%, #dfdfdf, #cdcdcd 5%);
```



## multiple gradients

از آنجاییکه css gradient با استفاده از خاصیت background-image تعریف می شود و می توان به یک عنصر چندین background-image اختصاص داد، در نتیجه می توان به یک عنصر نیز چندین gradient مختلف تخصیص داد. در این حالت باید هر تعریف gradient با یک کاما از تعریف دیگر جدا شود. تعریفی که در ابتدا بیاید بالاتر از باقی gradient ها (front) نمایش داده خواهد شد. برای اینکه gradient های تعریف شده بعدی نیز دیده شوند و زیر اولین gradient تعریف شده پنهان نشوند می توان آخرین color-stop در اولین تعریف gradient را برابر با transparent قرار داد.

با استفاده از این قابلیت می توان به عناصر موجود در صفحه حجم داده و عناصر سه بعدی ایجاد کرد.

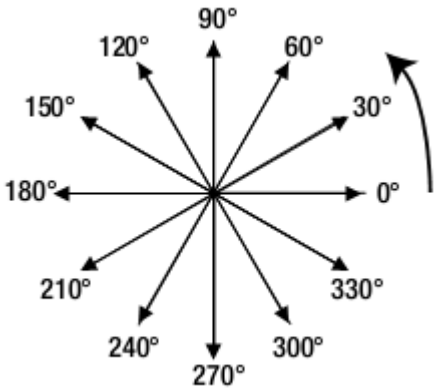
مثال : ایجاد یک توپ سه بعدی

```
#ball {
    width: 100px;
    height: 100px;
    border-radius: 50%;
    background-image: radial-gradient(circle 50px at 25px 30px,
#dfdfdf,transparent 70px),
                                radial-gradient(circle at 30% 30%, #f00,
#500);
}
```

## ایجاد gradient در مرورگرهای قدیمی

### linear-garedient

قبل از اینکه سازمان W3C توابع ()gradient را استانداردسازی کند، مرورگرها خود به صورت آزمایشی شروع به پشتیبانی از این خاصیت کردند. firefox اولین مرورگری بود از زاویه (angle) برای تعیین جهت خط gradient line استفاده کرد. firefox در ابتدا از polar coordinate system برای تعیین جهت استفاده کرد. در این سیستم صفر درجه برابر با مقدار right (یک خط افقی از چپ به راست) می باشد و باقی زوایا در خلاف جهت عقربه های ساعت افزایش پیدا می کنند. مرورگر opera و مرورگرهای webkit (مثل chrome , safari) نیز بعداً از این سیستم تبعیت کردند. همانطور که ملاحظه می کنید این روش با روش استاندارد W3C متفاوت می باشد.




مرورگرهای IE10 و firefox16 از استاندارد رسمی W3C برای gradient (compass coordinate) پشتیبانی می کنند.

مرورگرهای قدیمی که برای نمایش خاصیت ها از پیشوندهای مخصوص خود استفاده می کنند، برای مشخص کردن زوایا از روش قدیمی polar coordinate استفاده می کنند و مرورگرها جدید از روش جدید compass coordinate استفاده می کنند.

## تبدیل زوایای روش compass coordianate به زوایای متناظر در روش polar coordinate

زاویه صفر درجه در روش polar متناظر با زاویه ۹۰ درجه در روش compass می باشد. به منظور محاسبه زاویه polar برای مرورگرهای قدیمی از مراحل زیر پیروی می کنیم:

۱. اگر زاویه در روش compass کوچکتر یا مسوی ۹۰ درجه بود مقدار آن را از ۹۰ کم می کنیم.
۲. اگر زاویه در روش compass بزرگتر از ۹۰ درجه بود مقدار آن را از ۴۵۰ کم می کنیم.
۳. علامت منفی را از نتیجه محاسبات مراحل قبل حذف می کنیم تا زاویه polar موردنظر بدست آید.

مثال :

```
compass : 30° < 90°    =>      30 - 90 = -60          =>      polar : 60°
compass : 135° > 90°   =>      135 - 450 = -315       =>      polar : 315°
```

مثال :

```
background-image : -moz-linear-gradient(315deg,black,white);
background-image : -webkit-linear-gradient(315deg,black,white);
background-image : -o-linear-gradient(315deg,black,white);
background-image :linear-gradient(135deg,black,white);
```

در مثال فوق زوایای سه حالت اول (۱۳۵) کاملاً متفاوت با حالت استاندارد (۳۱۵) می باشند اما نتیجه gradient در تمامی مرورگرها یکسان خواهد بود.

پیشوندهای استفاده شده برای نمایش خاصیت های جدید css3 در مرورگرهای قدیمی عبارتند از :

- ✓ -moz- : برای مرورگر firefox
- ✓ -webkit- : برای مرورگرهای webkit مثل google chrome و safari
- ✓ -o- : برای مرورگر opera

### radial-garedient

نسخه های قدیمی مرورگرها از ساختار استاندارد W3C برای ایجاد radial-gradient پشتیبانی نمی کنند. در این مرورگرها باید از روش مخصوص همان مرورگرها (browser-specific) برای ایجاد radial-gradient استفاده کرد. تابع ()radial-gradient در مرورگرهای قدیمی دارای پارامترهای زیر می باشد :

- ✓ position : مشابه حالت استاندارد.

- ✓ shape and size : همان کلمات کلیدی استفاده شده در حالت استاندارد در اینجا نیز به کار می روند. دو کلمه کلیدی contain معادل closest-side و cover معادل farthest-corner نیز قابل استفاده می باشند. مقدار پیش فرض این پارامتر برابر ellipse farthest-side می باشد.
- ✓ color-stops : مشابه حالت استاندارد.

پارامترهای فوق بسیار مشابه حالت استاندارد می باشند ولی با تفاوت های زیر :

- ✓ پارامتر position در صورت تعریف باید اولین پارامتر باشد و با کاما از سایر پارامترها جدا شود.
- ✓ در اینجا برای تعریف مقدار پارامتر position (مقدار length) نیازی به استفاده از کلمه کلیدی at نیست.
- ✓ در مرورگر opera و مرورگرهای webkit برای تعریف size به صورت عددی نیاز به تعیین مقدار برای پارامتر shape نیست و فقط باید دو مقدار عددی به ترتیب به عنوان شعاع افقی و عمودی تعریف شوند. برای داشتن دایره باید هر دو شعاع افقی و عمودی یکسان باشند.
- ✓ مرورگر firefox نسخه ۱۵ و قبل از آن از مقادیر عددی برای تعیین size پشتیبانی نمی کنند.

تفاوت های اصلی در دو حالت به صورت خلاصه عبارتند از :

در روش استاندارد پارامترهای shape,size,position در یک لیست که با space از هم جدا شده اند قرار دارد و پارامتر position اخرین پارامتر می باشد :

```
circle closest-side at 100px 50px
```

در مرورگرهای قدیمی (حالت browser-specific) پارامتر position در ابتدا می آید و با کاما از پارامتر shape and size جدا می شود :

```
100px 50px , circle closest-side
```

مثال :

```
background-image : -moz-radial-gradient(100px 50px,circle closest-side,black,white);
background-image : -webkit-radial-gradient(100px 50px,circle closest-side,black,white);
background-image : -o-radial-gradient(100px 50px,circle closest-side,black,white);
background-image : radial-gradient(circle closest-side at 100px 50px,black,white);
```

در مرورگرهای قدیمی تر که از هیچ یک از حالت های توابع ()gradient پشتیبانی نمی کنند، می توان از یک رنگ ثابت برای پس زمینه استفاده کرده و یا برای ایجاد پس زمینه به صورت gradient از تصاویر استفاده کرد. به عنوان مثال می توان یک linear-gardient را در یک نرم افزار گرافیکی ایجاد کرده و سپس یک برش باریک از این تصویر را در پس زمینه عنصر با استفاده از خاصیت background-repeat تکرار کرد. برای اینکه صفحه ما در تمامی مرورگرها به یک صورت نمایش داده شود باید style خود را به صورت زیر تعریف کرده و مرتب کنیم :

۱. استفاده از رنگ پس زمینه (background-color)
۲. استفاده از تصاویر (gardient image)
۳. استفاده از توابع مخصوص مروگرهای قدیمی (browser-specific gradient functions)
۴. استفاده از توابع استاندارد (standard css gardient syntax)

مثال :

```
background-color : #181818;
background : url(gradient-image.png) repeat-x;
background-image : -moz-linear-gradient(deg,black,white);
background-image : -webkit-linear-gradient(۲۷۰deg,black,white);
background-image : -o-linear-gradient(۲۷۰deg,black,white);
background-image :linear-gradient(۱۸۰deg,black,white);
```