ANEXO II

CSS

Módulo: **Desenvolvemento Web en Contorno Cliente**

Ciclo: **DAW**

Introdución

A linguaxe CSS[1] foi creada co obxecto de separar a estrutura e o contido dun documento HTML da súa presentación, isto é, o aspecto visual do mesmo de cara ao usuario. Esta separación permite mellorar a accesibilidade ao contido, proporcionando maior flexibilidade e control na especificación de características de presentación.

Se cadra a principal vantaxe de CSS fronte ao formato HTML sexa que posibilita que varias páxinas compartan a mesma aparencia, reducindo a complexidade e repetición do contido estrutural.

CSS3

Ao contrario dos seus predecesores, CSS (presentado en 1996) e CSS2 (presentado en 1998), CSS3 constitúe un conxunto de módulos independentes que son publicados conforme van estando listos para o uso público, o que fai dela unha tecnoloxía altamente escalable e adaptable ás necesidades do mercado da Web actual.

Hai que ter en consideración que, polo feito de que un módulo CSS3 estea aínda no nivel de borrador non por iso non vai estar implementado nos navegadores, de feito, é moi habitual que as sucesivas actualizacións dos *browsers* vaian incluíndo estes módulos «provisorios» para ir asentando o seu uso e adaptarse paseniño a novos estándares. Dese xeito, os deseñadores poden ir probando novas características da tecnoloxía e, a un tempo, os desenvolvedores aproveitan a realimentación que destes e dos navegadores vailles chegando. O problema xorde cando cada motor de cada navegador interpreta esas propiedades de xeito distinto. Co gaio de resolver iso naceron os prefixos específicos dos navegadores, dos que falaremos en detalle máis adiante.

HTML5 e CSS3

A introdución da tecnoloxía HTML5 (entendendo coma tal a integración de HTML5, CSS3 e JavaScript) na Web supuxo unha revolución en tanto que o traslado de grande parte da carga de traballo ao lado do cliente permitiu a creación de páxinas web máis interactivas e complexas visualmente, coa consecuente ganancia en amigabilidade das mesmas.

Así pois, son as novas propiedades introducidas por CSS3 as que, principalmente, fan á Web actual ser como é en termos de aspecto: transformacións, transicións, animacións, etc.

[1] Cascading Style Sheets.

Retrocompatibilidade

Coma é obvio, e considerando como dixemos que se vai desenvolvendo e actualizando CSS3, é obvio que as versións máis antigas dos navegadores nunca van ter soporte para as propiedades máis recentes. Co gaio de resolver isto, os programadores Web combinan unha característica de HTML desenvolta por Microsoft coma son os «comentarios condicionais» (permiten tomar decisións en función do tipo de navegador empregado) con código JavaScript que autodetecte o navegador empregado, para así escoller a folla de estilos máis axeitada a cada caso. O anterior implica, loxicamente, desenvolver ficheiros de estilos separados para distintas versións dun mesmo navegador, coa carga de traballo engadida que iso implica.

```
<!--[if IE 6]>
<link href=”css/IE6.css” rel=”stylesheet” type=”text/css”>
<![endif] -->
<!--[if IE 7]>
<link href=”css/IE7.css” rel=”stylesheet” type=”text/css”>
<![endif] -->
<!--[if IE 8]>
<link href=”css/IE8.css” rel=”stylesheet” type=”text/css”>
<![endif] -->
<!--[if lte IE 8]>
<link href=”css/outro.css” rel=”stylesheet” type=”text/css”>
<![endif] -->
```

Exemplo de código CSS para navegadores Microsoft no que se escollen distintas follas de estilo segundo a versión de Internet Explorer empregada polo usuario.

Por sorte, se non temos que asegurar soporte a navegadores «moi vellos» o problema das leves diferenzas entre navegadores ao tempo de interpretar CSS pode ser resolto mediante os prefixos específicos dos que falamos anteriormente.

```
.imaxe {
  -webkit-transform: rotate(60deg);
  -moz-transform: rotate(60deg);
  -ms-transform: rotate(60deg);
  -o-transform: rotate(60deg);
  -khtml-transform: rotate(60deg);
  transform: rotate(60deg);
}
```

Exemplo de uso dos prefixos específicos de CSS para distintos navegadores.

No exemplo directamente superior, podemos ver que se introduce aparentemente por sextuplicada a mesma propiedade. Neste caso estariamos a reflectir o seguinte:

- **webkit:** referido a navegadores baseados no motor wekbit, tales coma Safari, Google Chrome ou a maioría de navegadores para dispositivos Android.

- **moz:** referido a navegadores baseados no motor Gecko tales coma Firefox ou IceWeasel.
- **ms:** referido a navegadores Microsoft baseados no motor de Internet Explorer, coma o propio IE ou Microsoft Edge.
- **o:** referido aos navegadores Opera.
- **khtml:** referido a Konqueror e ás versións antigas de Safari.

No caso de que un navegador soporte a propiedade estándar, aplicará unicamente esta. É por ese motivo que engadimos, no remate, a propiedade sen ningún tipo de prefixo.

Combinando CSS3 e HTML5

Aínda que o código CSS pode aniñarse no código HTML empregando o atributo `style` (isto é, pode ser escrito dentro do mesmo), tal xeito de programar viola a separación entre vista e estrutura da información que persigue a propia existencia de dúas tecnoloxías separadas, polo que deberemos sempre evitalo na medida do posible.

Así, cando nos queiramos referir documentos de estilos CSS dende un documento HTML, de xeito que se encarguen de establecer a vista da información, teremos que engadir unha liña como a que figura a continuación entre as etiquetas <head> e </head> de HTML (é dicir, na cabeceira do documento) para cada un dos nosos ficheiros CSS.

```
<link rel="stylesheet" href="estilos.css" type="text/css" />
```

Onde **estilos.css** é un dos documentos CSS que empregará o noso documento HTML.

Dende que se presentou o estándar HTML5 o atributo type xa non é preciso, polo que poderiamos escribir a liña como segue, conservando a funcionalidade:

```
<link rel="stylesheet" href="estilos.css" />
```

Selectores

Un **selector** ou **estilo**[2] en CSS é una etiqueta que permite identificar aquel elemento ou conxunto de elementos HTML aos que vai ser aplicado. A súa estrutura consta de dúas partes:

- **Selector:** etiqueta de texto que determina o elemento HTML (ou conxunto de elementos) aos que é aplicable o estilo.
- **Declaración:** define as propiedades e valores que constitúen o estilo marcado pola estrutura. Vai sempre seguida do selector e acoutada por chaves (**{}**).

Xerarquía dos elementos HTML

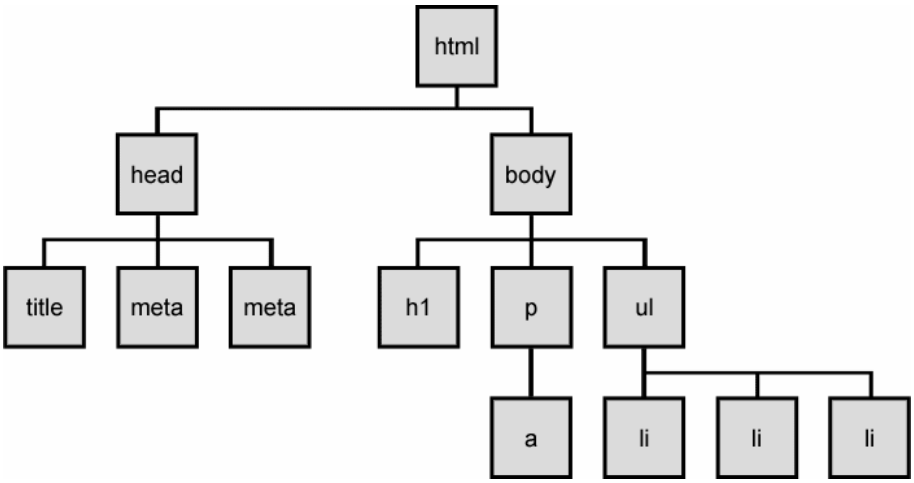
Para poder manexar de xeito apropiado os selectores de CSS é preciso coñecer a xerarquía dos elementos que van compoñer unha páxina HTML. Destacaremos aquí só os de uso habitual, e iremos introducindo o resto ao longo do tema a través de exemplos prácticos.

Para un exemplo coma o que sigue:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo para a xerarquía de elementos HTML</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="es" />
  </head>
  <body>
    <h1>Isto é unha cabeceira de texto</h1>
    <p>Un parágrafo cunha
      <a href="http://www.google.com/">ligazón</a>.
    </p>
    <ul>
      <li>Un elemento de lista.</li>
      <li>Outro.</li>
      <li>Máis un.</li>
    </ul>
  </body>
</html>
```

Así pois, nun código coma o anterior, os elementos HTML da páxina posúen a seguinte xerarquía:

[2] Coloquialmente é habitual velos denominados coma **estilos**, aínda que a designación oficial sexa **selectores**.



Temos, polo tanto, que introducir unha serie de conceptos relativos ás relacións xerárquicas entre elementos HTML que nos serán de axuda ao tempo de traballar con CSS:

- **Sucesor** ou **descendente:** elemento que está situado na subárbore dun elemento concreto (é dicir, ao que se chega só por camiños descendentes). Por exemplo: *para o elemento **body** serían sucesores **h1**, **p**, **a**, **ul** e todos os **li**.*
- **Antecesor** ou **ascendente:** elemento que está situado na superárbore dun elemento concreto (isto é, ao que se chega só por camiños ascendentes). Por exemplo: *o elemento **body** tería coma único antecesor a **html**.*
- **Fillo:** elemento que é sucesor directo dun elemento concreto (é dicir, só hai que percorrer un único tramo de camiño descendente para chegar a el). Por exemplo: *para o elemento **body** serían fillos só **h1**, **p** e **ul**.*
- **Pai:** elemento que é antecesor directo dun elemento concreto (é dicir, só hai que percorrer un único tramo de camiño ascendente para chegar a el). Por exemplo: *para calquera dos elementos **li** o seu pai sería **ul**.*
- **Irmán:** elemento que posúe o mesmo elemento pai ca o elemento en cuestión (é dicir, só hai que percorrer un único tramo de camiño ascendente seguido dun único tramo de camiño descendente para chegar a el). Por exemplo: *serían irmáns os elementos **title** e ámbolos dous **meta**.*

Subsecuentemente, pódense definir todo tipo de relacións de parentesco a semellanza daquelas das árbores xenealóxicas, aínda que non teñen por que ser de especial utilidade.

Selectores simples

CSS distingue tres tipos de **selectores simples** segundo a cantidade de elementos HTML aos que afectan e o xeito en que se determinan os elementos afectados:

- **Tipo de elemento:** empregando directamente o nome dalgún *tag* de HTML coñecido, de xeito que as súas propiedades aplicaranse a todos os elementos dese tipo con independencia da súa posición na árbore do documento. No fragmento de código que figura a continuación vemos un exemplo de estilo CSS aplicado aos elementos HTML de tipo parágrafo (p).

```
p {
  font-size: 2em;
  color: blue;
}
```

- **Identificador:** irá precedido do cancelo (**#)** e aplicarase a aqueles elementos[3] HTML que correspondan co identificador homónimo establecido no seu atributo **id** con independencia da súa posición na xerarquía do documento. O código que figura a continuación amosa a aplicación dun estilo de texto ao elemento que teña o identificador **cabeceira** no documento afectado.

```
#cabeceira {
  font-size: 14em;
  color: black;
}
```

- **Clase:** irá precedido do punto (**.**) e aplicarase só a aqueles elementos HTML que correspondan coa clase homónima establecida no seu atributo **class**, con independencia da súa posición na árbore do documento. No seguinte exemplo de código podemos ver como aplicar un estilo de bordo aos elementos que pertenzan á clase **cadro** no documento afectado.

```
.cadro {
  border: 1px solid black;
}
```

Clases múltiples

Un mesmo elemento HTML pode ter asociadas varias clases (indicaríanse separadas por espazos pero aplicadas igualmente dende o atributo **class**), o que pode ter implicacións sobre o tratamento das mesmas dependendo de como definamos os estilos correspondentes en CSS. Por exemplo:

```
.estilo {
  background-color: yellow;
}
```

[3] A recomendación do estándar HTML5 é que os **id** sexan únicos, isto é, que non haxa varios elementos compartindo identificador nun mesmo documento HTML, aínda que non é obrigatorio.

```
p.estilo {
  color: blue;
}

div.estilo {
  border: 1px solid red;
}
```

Neste exemplo, todos os elementos da clase `estilo` terán fondo amarelo pero, particularmente, aqueles que sexan de tipo `p` terán o texto en cor azul, namentres que os que sexan de tipo `div` terán un bordo simple de 1 píxel de largura e cor vermella.

No caso de asignar múltiples clases a un elemento, aplicaránselle todas as características definidas por cada unha desas clases, impoñéndose as definidas posteriormente en caso de conflicto. Por exemplo:

```
.estiloA {
  color: gray;
  background-color: blue;
}

.estiloB {
  font-weight: bold;
  background-color: red;
}
```

En caso de que tiveramos, un `div` ao que foran aplicados os estilos `estiloA` e `estiloB` definidos no exemplo directamente superior, entón ese elemento tería as seguintes características (texto en cor gris –segundo `estiloA`- e negrita –segundo `estiloB`- e cor de fondo vermella –en tanto que `estiloB` foi definido despois e sobreescribe a característica de `estiloA`-).

A maiores, poderiamos aplicar certas características só aos elementos que pertenzan simultaneamente a determinadas clases, do xeito:

```
.estiloA.estiloB {
  border-color: orange;
}
```

No exemplo anterior estamos a implicar que a aqueles elementos que pertenzan simultaneamente ás clases `estiloA` e `estiloB` seralles aplicada unha cor de bordo laranxa.

Identificadores múltiples

De xeito case análogo a coma ocorre coas clases e, coma xa comentamos con anterioridade, é posible aplicar un mesmo identificador a diferentes elementos (aínda que isto non se recomenda). Se, pola razón que fora, facemos uso desa posibilidade, CSS danos así mesmo a posibilidade de discriminar por tipo de elemento cunha sintaxe semellante á empregada coas clases. Por exemplo:

```
#texto_normal {
 font-weight: bold;
}

p#texto_normal {
  color: green;
}
```

Do xeito anterior, todos os elementos que empreguen o identificador `texto_normal` terán formato de negrita nos textos, pero só aqueles de tipo p terán o texto en cor verde.

Selector universal

Dada a necesidade de redefinir as características de formato que veñen preestablecidas polos navegadores para os documentos HTML (coma, por exemplo, o uso da fonte `Times New Roman`), con CSS2 foi introducida a posibilidade de referirse a todos os elementos dun documento HTML mediante o uso dun único **selector universal**, tal e coma vemos no exemplo a continuación:

```
* {
  color: cyan;
  font-size: 18px;
}
```

Indicando o anterior implicaría que, por defecto, todos os elementos do documento HTML terían texto de cor cian e tamaño de fonte 18 píxeles.

Entre os usos máis estendidos do selector universal atopamos o coñecido coma *CSS reset*, no que se redefinen todas as características que nos interese para os documentos HTML que vaian empregar a nosa folla de estilos. Por exemplo:

```
* {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}
```

En ocasións vainos interesar aplicar restricións só a certa subárbore da xerarquía do documento HTML (por exemplo, a todos os descendentes dun elemento).

```
#texto_normal * {
  text-align: center;
}
```

No exemplo anterior estariamos a indicar que todos os elementos descendentes de aqueles que teñan o identificador `texto_normal` empregarán aliñamento centrado nos seus textos.

Combinadores

Existe máis un xeito de aplicar características de xeito xeralizado aos elementos dun documento HTML alén das clases e os identificadores: os **combinadores** ou **selectores combinados**. Este tipo de selectores fan uso de catro carácteres distintos segundo a relación xenealóxica existente entre os elementos afectados:

- **Coma (,):** permite referirse a todos os elementos dunha listaxe. Por exemplo, se queremos afectar a todos os elementos `p`, `div` e ademais ao elemento co identificador `proba` faremos algo coma o seguinte:

```
p,div,#proba {
  background-color: green;
}
```

- **Espazo ( ):** permite referirse a todos os **elementos descendentes**. Retomando o exemplo empregado no punto anterior, podemos encadear selectores universais separados por espazos para afectar a diferentes niveis da xerarquía, por exemplo:

```
#texto_normal * p {
  font-size: 10px;
} /* Estas propiedades afectarían a todos os elementos de tipo p netos (e
inferiores) do elemento con identificador texto_normal. */

body * * div {
  color: red;
} /* Estas propiedades afectarían a todos os elementos de tipo div bisnetos (e
inferiores) do elemento body. */
```

No exemplo anterior estariamos a establecer dúas regras: por unha banda, os elementos de tipo `p` que sexan netos de elementos co identificador `texto_normal` empregarán un tamaño de fonte de 10 píxeles; namentres que os elementos de tipo `div` que sexan bisnetos do elemento `body` terán o texto en cor vermella.

- **Maior (>):** permite referirse a todos os **elementos fillos**, pero non así os descendentes destes, é dicir, só toma o primeiro nivel de descendentes do elemento afectado.

```
#especial > div {
  border-color: brown;
}
```

Considerando o exemplo anterior teriamos que todos os elementos de tipo `div` que sexan directamente fillos de elementos que empreguen o identificador `especial` terán borde de cor marrón.

- **Suma (+):** permite referirse ao primeiro **elementos irmán** do elemento afectado:

```
p + div {
  color: blue;
}
```

Segundo o exemplo anterior aqueles elementos de tipo `div` que aparezan no documento HTML xusto despois dun elemento `p` terán o seu texto en cor azul (se houbera varios `div` consecutivos só se vería afectado o primeiro deles).

- **Vírgula (~):** permite referirse a todos os **elementos irmáns** do elemento afectado.

```
h1 ~ h2 {
  font-weight: bold;
}
```

Segundo o exemplo anterior, todos os elementos `h2` que aparezan despois dun elemento `h1` do que sexan irmáns terán o texto en negrita.

Todos os anteriores poden ser combinados entre si para implicar relacións máis complexas, coma por exemplo:

```
p > div + ul {
  color: red;
}
```

No caso anterior, todos os primeiros elementos `ul` que aparezan directamente despois dun irmán `div` e sexan fillos dun elemento `p`, terán o texto en cor vermella.

Selectores de atributos

Os **selectores de atributos** permiten aplicar distintos estilos CSS aos elementos dun documento HTML segundo os atributos que empreguen e mesmo, en función dos seus valores. Neste caso, existen sete variantes, que serían:

- **Selector de atributos simple:** aplícase a aqueles elementos HTML que teñan definido o atributo indicado, con independencia do seu valor:

```
a[href] td[colspan] {
  color: green;
}
```

No exemplo anterior estariamos a poñer o texto en cor verde para todos os elementos a que teñan definido o atributo `href`, así coma a todos os elementos `td` que estean a facer uso do seu atributo `colspan`.

- **Selector de atributos exactos:** permite aplicar estilos a elementos HTML en función do valor exacto dos seus atributos.

```
img[alt=”Esta imaxe representa unha casa”] {
  width: 200px;
}
```

O exemplo directamente superior aplica unha largura de 200 píxeles a aqueles elementos de tipo `img` que teñan no atributo `alt` o valor exacto *`Esta imaxe representa unha casa`*[4].

Ademais do anterior, este selector é aplicable en negativo, indicando por exemplo aqueles atributos que **non** cumpran a condición:

```
img:not([alt]) {
  width: 100px;
}
```

Neste último exemplo, o selector aplicará unha largura de 100 píxeles a aquelas imaxes que non teñan valor definido no seu atributo `alt`.

- **Selector de atributos por palabras:** permite aplicar estilos a elementos HTML que teñan atributos cun valor que conteña o texto indicado coma palabra[5].

```
img[src~=”imaxe”] {
  border: 2px dotted gray;
}
```

O caso anterior tomará todos aqueles elementos de tipo `img` cuxo atributo `src` conteña a cadea de texto *`imaxe`* e aplicaralles un bordo de cor gris punteado de 2 píxeles de largura.

[4] É importante lembrar, para todos estes selectores, que CSS distingue maiúsculas de minúsculas.

[5] Isto é, delimitado por espazos, tabulacións, signos de puntuación, etc. Pero non contido noutras palabras (en cuxo caso sería unha subcadea).

- **Selector de linguaxes:** permite seleccionar os elementos HTML segundo a linguaxe aplicada ao documento.

```
p[lang|=”es”] {
  font-family: Arial;
}
```

Así pois, para o exemplo anterior, aplicarase o estilo (fonte de tipo `Arial`) a aqueles elementos de tipo p que teñan coma linguaxe definida o castelán.

- **Selector de atributos por valor de inicio:** permite aplicar estilos a elementos HTML que teñan atributos definidos cuxo valor comece coa información indicada.

```
a[href^=”https://”] {
  color: green;
}
```

O exemplo directamente superior aplicará cor verde ao texto daqueles elementos de tipo a cuxo atributo `href` comece por *`https://`*.

- **Selector de atributos por valor de fin:** permite aplicar estilos a elementos HTML que teñan atributos definidos cuxo valor remate coa información indicada.

```
img[src$=”.jpg”] {
  height: 80px;
}
```

O estilo anterior aplicarase, pois, a aqueles elementos de tipo `img` cuxo atributo `src` remate co texto *`.jpg`*.

- **Selector de atributos por contido parcial:** permite aplicar estilos a elementos HTML en función das subcadeas que conteñan os valores dos seus atributos.

```
a[href*=”www”] {
  font-family: Verdana;
}
```

O anterior exemplo mostra un caso de aplicación dunha fonte (`Verdana`) aos elementos de tipo a que conteñan, para o seu atributo `href`, a subcadea de texto *www*.

Pseudoclases

As **pseudoclases** ou **pseudoselectores** de CSS permiten determinar o estilo dun elemento HTML baseándose en metainformación externa á árbore do documento, é dicir, información alén da etiqueta, os atributos (e os seus valores) e o contido.

A táboa seguinte amosa as pseudoclases dispoñibles en CSS3 coa súa sintaxe de uso e unha breve descrición.

Sintaxe	Descrición
`etiqueta:active`	Aplícase ao elemento cando se está a soster o clic do rato no mesmo.
`etiqueta::after`	Insire algo despois de cada elemento dun certo tipo.
`etiqueta::before`	Insire algo antes de cada elemento dun certo tipo.
`input:checked`	Aplícase aos `input` escollidos.
`input:default`	Aplícase aos `input` por defecto.
`input:disabled`	Aplícase aos `input` deshabilitados.
`etiqueta:empty`	Aplícase a cada etiqueta dun tipo que non teña fillos.
`input:enabled`	Aplícase aos `input` habilitados.
`p:first-child`	Aplícase a cada `p` que é o primeiro fillo do seu pai.
`p::first-letter`	Aplícase á primeira letra de cada `p`.
`p::first-line`	Aplícase á primeira liña de cada `p`.
`p:first-of-type`	Aplícase a cada `p` que é o primeiro descendente `p` doutro.
`input:focus`	Aplícase ao `input` que ten o foco.
`etiqueta:hover`	Aplícase ao elemento situado baixo o cursor do rato.
`input:in-range`	Aplícase aos `input` cuxo valor estea fora do rango indicado.
`input:indeterminate`	Aplícase aos `input` en estado indeterminado.
`input:invalid`	Aplícase aos `input` que teñan un valor inválido.
`etiqueta:lang(`*`idioma`*`)`	Aplícase a cada elemento dese tipo quet teña o idioma indicado.
`etiqueta:last-child`	Aplícase a cada elemento que sexa o último fillo do seu pai.
`etiqueta:last-of-type`	Aplícase a cada elemento que sexa o último fillo dese tipo do seu pai.
`a:link`	Aplícase a hipervínculos non visitados.
`:not(etiqueta)`	Aplícase a todos os elementos que **non** sexan do tipo indicado.
`etiqueta:nth-child(n)`	Aplícase a cada elemento que sexa o fillo n-ésimo do seu pai.
`etiqueta:nth-last-child(n)`	Aplícase a cada elemento que sexa fillo n-esimo do seu pai contando dende o último fillo.
`etiqueta:nth-last-of-type(n)`	Aplícase a cada elemento que sexa o n-ésimo fillo dese tipo do seu pai contando dende o último fillo.
`etiqueta:nth-of-type(n)`	Aplícase a cada elemento que sexa o n-ésimo fillo dese tipo.
`etiqueta:only-of-type`	Aplícase a cada elemento que sexa o único fillo dese tipo do seu pai.
`etiqueta:only-child`	Aplícase a cada elemento que sexa fillo único.
`input:optional`	Aplícase aos `input` **non** teñan o atributo *required* definido.
`input:out-of-range`	Aplícase aos `input` cuxo valor estea fora do rango indicado.
`input::placeholder`	Aplícase aos `input` que teñan o atributo *placeholder* definido.
`input:read-only`	Aplícase aos `input` que teñan o atributo *readonly* definido.
`input:read-write`	Aplícase aos `input` que **non** teñan o atributo *readonly* definido.
`input:required`	Aplícase aos `input` teñan o atributo *required* definido.
`:root`	Aplícase ao elemento raíz do documento.
`::selection`	Aplícase á porción do elemento seleccionada polo usuario.
`:target`	Aplícase ao elemento activo actual.
`input:valid`	Aplícase a todos os `input` cun valor válido.
`a:visited`	Aplícase aos hipervínculos visitados.

Sintaxe	Descrición

Textos

CSS fornece funcionalidade para a manipulación de texto con formato, permitindo maquetar case calquera posibilidade de edición de que atopariamos tipicamente en procesadores de texto enriquecido e aplicacións de deseño gráfico.

Na manipulación de texto vía CSS3 interveñen tres módulos do estándar:

- **CSS Text:** reúne a maioría de características relacionadas coa transformación e decoración de textos, tanto a nivel de carácter e palabra coma de parágrafo.
- **CSS Writing Modes**[6]**:** determina a direccionalidade do texto.
- **CSS Line Grid:** permite aliñar o texto conforme a unha grella invisible, e está orientado a linguaxes de escritura vertical.

Propiedades básicas

Salientaremos neste apartado aquelas características de uso habitual en CSS3 que afectan ao formato de texto. Así atopamos:

- **color:** determina a cor das letras que compoñen o texto.
- **background-color:** determina a cor de fondo do texto.

```
#texto {
  color: #491409;
  background-color: #97D7E6;
}
```

O resultado será semellante ao seguinte:

Exemplo de texto con formato CSS3.

- **font-size:** determina o tamaño de letra que empregará o noso estilo. Admite tanto unidades absolutas (cm, mm, in, px –relativos ao dispositivo-, pt, pc) coma relativas (em, ex, ch, rem, vw, vh, vmin, vmax, %).
- **font-family:** determina a familia de fonte que empregaremos. Podemos empregar unha «fonte segura» (recoñecida pola W3C) directamente mediante nome, ou definir unha mediante a regra **@font-face** que trataremos máis adiante.
- **font-style:** define o estilo da fonte, que pode tomar os seguintes valores:
  - *normal*.
  - *italic*.

[6] Chamado Text Layout até CSS2.

- *oblique*.

- **font-variant:** define se o texto debe ser ou non amosado en versaletas, admitindo os valores:
  - *normal*.
  - *small-caps*.
- **font-weight:** define a gordura dos carácteres. Pode tomar os valores:
  - *normal*.
  - *bold*.
  - *bolder*.
  - *lighter*.
  - «medida»: onde 400 é o equivalente a normal e 700 o equivalente á negrita típica (*bold*).
- **font:** admite todas as características relativas á fonte nunha mesma propiedade (combina ás vistas anteriormente).
- **text-shadow[7]:** permite definir a sombra do texto en base a catro valores:
  - *h-shadow[8]:* posición horizontal da sombra.
  - *v-shadow[9]:* posición vertical da sombra.
  - *blur-radius:* radio do esborranchamento (difusión) da sombra.
  - *color:* cor da sombra.
- **font-strecht:** permite escoller un subtipo concreto dunha fonte dentro da mesma familia de tipos de letra. Ese subtipo vai determinar a compresión da letra no eixo horizontal. Soporta os seguintes valores:
  - ***ultra-condensed***.
  - ***extra-condensed***.
  - ***condensed***.
  - ***semi-condensed***.
  - ***normal***.
  - ***semi-expanded***.
  - ***expanded***.
  - ***extra-expanded***.
  - ***ultra-expanded***.

Podemos ver aquí un exemplo de CSS empregando algunhas das propiedades anteriores:

[7] Internet Explorer (até a versión 10) non recoñece esta propiedade, o que nos obriga a empregar a propiedade `filter` con valor `DropShadow` para resolver ese problema.

[8] Esta propiedade admite valores negativos que levarían a sombra cara á esquerda dos carácteres.

[9] Esta propiedade admite valores negativos que levarían a sobmra cara á parte superior dos carácteres.

```
#texto {
  color: #491409;
  background-color: #97D7E6;
  font-size: 36pt;
  font-family: helvetica;
  text-shadow: 2px 3px 6px black;
  font-stretch: condensed;
}
```

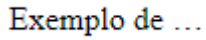
Obtendo algo semellante ao seguinte (o texto non está a escala co exemplo anterior):

- **text-overflow:** determina o xeito en que o texto desborda e se é ou non amosado ese desbordamento. Admite cinco valores distintos:
  - *hidden:* agóchase o desbordamento.
  - *clip:* trunca o desbordamento.
  - *ellipsis:* emprega o carácter da elipse (triple punto) no canto do desbordamento.
  - *«cadea de texto»:* amosa a cadea indicada entre comiñas.
  - *fade*[10]*:* esvaece o texto indicando unha lonxitude ou porcentaxe.

Neste caso aplicamos o noso estilo a un div que contén directamente ao texto de exemplo (obsérvese como complementamos coas propiedades `overflow` e `white-space` para consolidar o resultado):

```
#caixa {
   overflow: hidden;
   width: 100px;
  text-overflow: ellipsis;
  white-space: nowrap;
}
```

Sendo o resultado algo así:



- **text-align:** permite definir o aliñamento horizontal do texto dentro dun elemento de tipo bloque (*block*) asumindo que estamos a empregar unha orientación na escrita dende a esquerda cara á dereita. Admite múltiples valores:
  - *left:* aliñará o texto contra a marxe esquerda do contedor.
  - *right:* aliñará o texto contra a marxe dereita do contedor.
  - *center:* aliñará o punto medio de cada liña de texto sobre o punto medio entre as marxes esquerda e dereita.

[10] Esta propiedade está en desenvolvemento e non é soportada pola maioría de navegadores aínda.

- *justify:* aplica aliñamento xustificado (todas as liñas do parágrafo esténdense dende a marxe esquerda á dereita, agás a última –que equivale a un aliñamento á esquerda-).
- *start*[11]*:* aliñará o texto contra a marxe de comezo do sentido da escrita (no noso caso, a esquerda).
- *end*[12]*:* aliñará o texto contra a marxe de remate do sentido da escrita (no noso caso, a dereita).

- **word-wrap:** define como o navegador vai realizar a quebra daquelas palabras que, pola súa largura, non caiban dentro da liña actual na que se atopan no seu elemento contedor. Manexa os seguintes valores:
  - *normal:* as liñas só quebran entre palabras, pero non dentro delas.
  - *break-word*[13]*:* permite quebrar as palabras arbitrariamente.

Supoñamos que temos agora tres divs con texto dentro dos mesmos aos que aplicamos os seguintes estilos:

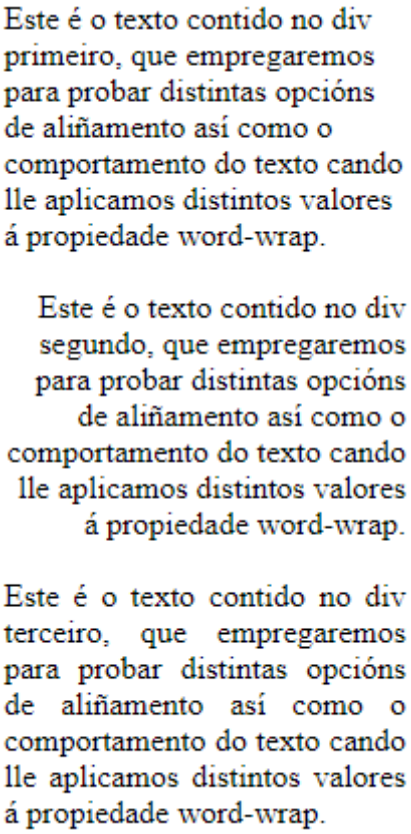
```
#primeiro {
  width: 200px;
  text-align: left;
  word-wrap: normal;
}

#segundo {
  width: 200px;
  text-align: right;
  word-wrap: break-word;
}

#terceiro {
  width: 200px;
  text-align: justify;
  word-wrap: break-word;
}
```

[11] Non soportado por todos os navegadores na actualidade.

[12] Non soportado por todos os navegadores na actualidade.

[13] Esta propiedade só se aplicará en casos extremos na que a separación entre palabras non sexa posible.

O resultado podería ser algo coma o que sigue:



- **white-space:** define o xeito en que se procesarán os espazos en branco nun texto, considerando as características particulares que HTML fai dese tipo de carácteres. Os valores que manexa son os seguintes:
    - *normal:* considera as ocorrencias consecutivas dun carácter en branco coma se foran unha única e realiza axuste de liñas cando sexa preciso (este é o valor por defecto da propiedade).
    - *pre:* conserva as secuencias de carácteres en branco e só fai axuste de liñas cando haxa quebras de liña.
    - *nowrap:* consolida as secuencias de carácteres en branco nun só, pero non realiza axuste de liñas (só se atopa unha etiqueta `<br />`).
    - *pre-wrap:* preserva as secuencias de carácteres en branco e realiza axuste de liñas onde sexa preciso e nas quebras de liña.
    - *pre-line:* consolida as secuencias de carácteres en branco e realiza axuste de liñas onde sexa preciso e nas quebras de liña.

Neste caso aplicamos o noso estilo a tres etiquetas de tipo parágrafo co seguinte contido:

```
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
```

Velaquí o código CSS que aplicamos:

```
#paragrafo1 {
  white-space: nowrap;
}

#paragrafo2 {
  white-space: normal;
}

#paragrafo3 {
  white-space: pre;
}
```

Obtendo algo semellante a isto:

Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un te

Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.

Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.
Aquí hai un texto. Aquí hai un texto. Aquí hai un texto.

- **text-indent:** establece a sangría ou «indentación» da primeira liña do texto dentro dun contedor. Admite valores en medida (px, pt, cm, em, etc.) ou porcentaxe (da largura do elemento contedor).

Aplicando a un par de exemplos atrás:

```
#primeiro {
  width: 200px;
  text-align: left;
  word-wrap: normal;
  text-indent: 15px;
}
```

Co seguinte resultado:

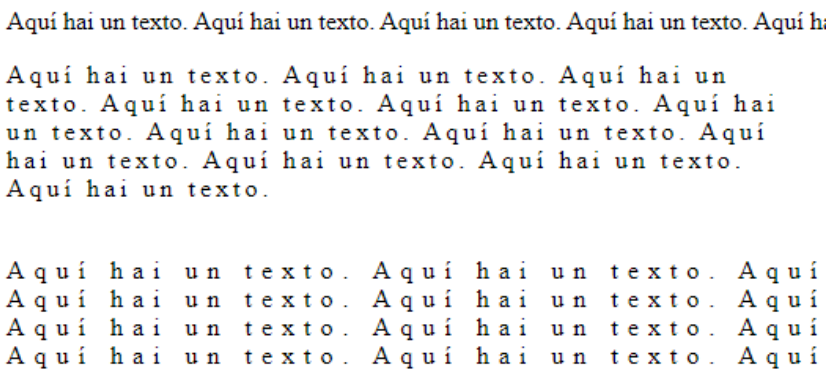
Este é o texto contido no div primeiro, que empregaremos para probar distintas opcións de aliñamento así como o comportamento do texto cando lle aplicamos distintos valores á propiedade word-wrap.

- **letter-spacing:** define o espazamento entre letras no texto, definido cunha medida.

Retomando o exemplo relativo á xestion de espazos en branco, teriamos:

```
#paragrafo1 {
  white-space: nowrap;
  letter-spacing: normal;
}

#paragrafo2 {
  white-space: normal;
  letter-spacing: 3px;
}

#paragrafo3 {
  white-space: pre;
  letter-spacing: 6px;
}
```

Cuxo resultado será semellante a isto:

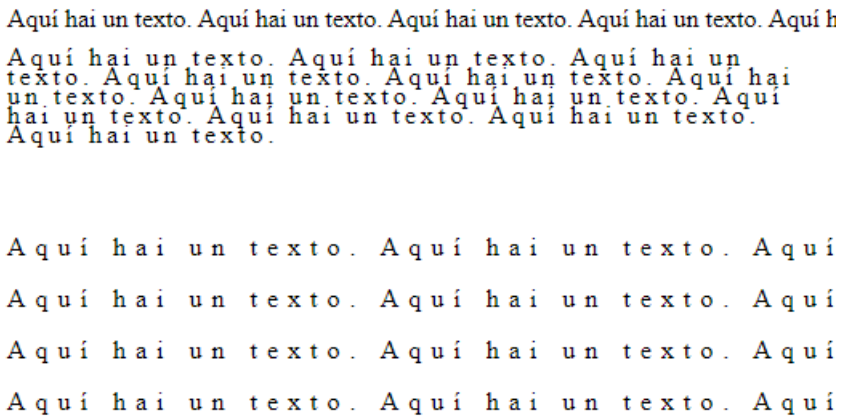


- **line-height:** permite determinar o espazamento entre liñas (ou altura de liña), soportando os valores típicos (normal, número –múltiplo do tamaño de fonte empregado-, medida ou porcentaxe).

Continuando pois co exemplo anterior:

```
#paragrafo1 {
  white-space: nowrap;
  letter-spacing: normal;
  line-height: 5px;
}

#paragrafo2 {
  white-space: normal;
  letter-spacing: 3px;
  line-height: 80%;
}

#paragrafo3 {
  white-space: pre;
  letter-spacing: 6px;
  line-height: 2;
}
```

Sendo o resultado semellante a isto:



Empregar fontes propias

Coma comentamos anteriormente ao explicar a propiedade `font-family`, o comportamento por defecto de CSS e HTML impídenos empregar fontes alén daquelas recollidas no conxunto de «fontes seguras».

@font-face

A pesares do anterior, CSS3 fornece a posibilidade de traballar con calquera tipografía sempre que dispoñamos dun ficheiro de fonte válido a través do mecanismo **`@font-face`**. Así, se queremos empregar unha fonte non segura referíndonos á mesma como **TipografiaNova** (dende a propiedade `font-family`) introduciremos ao comezo do noso ficheiro CSS algo semellante ao seguinte:

```
@font-face {
  font-family: 'TipografiaNova';
  /* Para IE9 e compatibles */
  src: url('tipografianova.eot');
  /* Para IE6, IE7 e IE8 */
  src: url('tipografianova.eot?#iefix') format('embedded-opentype'),
       /* Para navegadores actuais */
       url('tipografianova.woff2') format('woff2'),
       url('tipografianova.woff') format('woff'),
       /* Para navegadores Webkit */
       url('webfont.ttf')  format('truetype'),
       /* Para Safari 4.1 ou inferior */
       url('tipografianova.svg#svgFontName') format('svg');
}
```

Como podemos ver, sopórtanse múltiples formatos de ficheiros de fontes distintos, co gaio de asegurar a compatibilidade entre navegadores. Entre eses formatos atopamos:

- **woff/woff2:** acrónimo de **Web Open Font Format**, este formato foi desenvolvido por Mozilla para mellorar os tempos de carga das fontes nas páxinas web.
- **svg/svgz:** acrónimo de **Scalable Vector Graphics**, é un formato de imaxe vectorial que as versións máis vellas de Safari empregaban para representar fontes.

- **eot:** acrónimo de **Embedded Open Type**, foi creado por Microsoft (quen introduciu orixinalmente o mecanismo de `@font-face`).
- **otf/ttf:** acrónimos de **OpenType Font** e **TrueType Font** respectivamente, son os formatos de ficheiros de fonte máis empregados e difundidos na informática en xeral.

@import

No caso de que non teñamos aloxada a fonte no noso servidor, existe máis unha posibilidade para traballar con fontes non seguras, coma é empregar o mecanismo **`@import`**. Igual que ocorría con @font-face, deberemos definilo ao comezo do documento CSS, podendo a partires dese punto referir a nosa fonte dende calquera propiedade `font-family`.

Neste exemplo vemos como importar unha fonte do catálogo de Google Fonts ao noso código CSS:

```
@import url(//fonts.googleapis.com/css?family=Open+Sans);
```

Cores, fondos e degradados

Dado que o formato de documentos HTML vía CSS3 adoita implicar a emprega de cores (xa sexan sólidas, con transparencia e/ou con degradados), empregaremos este apartado para introducir os distintos xeitos en que esta tecnoloxía permite referir as cores así coma as propiedades que saquen proveito das mesmas.

Describindo cores

En CSS3 atopamos tres posibilidades (con divesas variantes) de indicar a cor que queremos empregar nunha propiedade:

- **Por nome:** esta característica, herdada de HTML, é soportada por todos os navegadores de uso habitual, e posibilita o uso de 140 **«cores seguras»** predefinidas a través do seu nome.

```
.actual {
  background-color: aliceblue;
}
```

O exemplo anterior formatará o elemento con id `actual` para que teña unha cor de fondo azul moi clara (case branca).
Podemos consultar a listaxe de «cores seguras» no seguinte enlace oficial do W3Consortium:

https://www.w3schools.com/cssref/css_colors.asp

- **Por código RGB[14]:** é unha escala aditiva (parte da cor negra) que expresa a cantidade de vermello, verde e azul que representan á nosa cor. Existen catro alternativas para a representación de cores empregando ese método:
  - Decimal sen transparencia: usa tres números decimais de 8 bits cada un (é dicir, poden tomar calquera valor enteiro entre 0 e 255) para representar a cantidade de vermello, verde e azul da imaxe.

```
.actual {
  background-color: rgb(248,240,255);
}
```

[14] A escala RGB (Reg Green Blue) emprega 8 bits para cada unha das canles de cor.

O exemplo anterior representa a mesma cor definida no primeiro exemplo do apartado (`aliceblue`), pero mediante o seu código RGB expresado en hexadecimal.

- Hexadecimal sen transparencia (rgb): usa un número hexadecimal de 6 cifras precedido do símbolo da grella (#) para representar a cantidade de vermello, verde e azul que posúe a imaxe.

```
.actual {
  background-color: #F0F8FF;
}
```

Novamente, con este exemplo, estariamos a empregar a cor `aliceblue`, neste caso expresada mediante o seu código RGB en decimal.

- Decimal con transparencia (rgba): incorpora un número adicional para representar a canle de transparencia *alpha*, na que o 0 implica transparencia total e o 1 opacidade total.

```
.actual {
  background-color: rgba(248,240,255,0.8);
}
```

Máis unha vez aplicamos a cor `aliceblue`, pero indicando que o elemento ten unha opacidade do 80%.

Así mesmo, ese valor pode ser representado en porcentaxe, de xeito equivalente:

```
.actual {
  background-color: rgba(248,240,255,80%);
}
```

- Hexadecimal con transparencia: incorpora dúas cifras hexadecimais adicionais para representar os 8 bits correspondentes a *alpha*.

```
.actual {
  background-color: #F0F8FF32;
}
```

Coma no exemplo anterior, neste caso estariamos a representar a cor de fondo `aliceblue` cunha transparencia de 50.

- **Por código HSL[15]:** é unha escala que expresa o ton, saturación e luminosidade que representan á nosa cor. Existen dúas alternativas para a representación de cores empregando ese método:
    - HSL sen transparencia (hsl): define só as propiedades de ton, saturación e luminosidade:

```
.actual {
  background-color: hsl(208, 100%, 97%);
}
```

Este exemplo volve representar a cor `aliceblue`, neste caso en escala HSL.

- HSL con transparencia (hsla): engade unha canle *alpha* de xeito análogo a coma ocorría con **rgba**, expresable cun número do 0 (transparencia total) ao 1 (opacidade total):

```
.actual {
  background-color: hsla(208, 100%, 97%, 0.8);
}
```

Ou, igualmente, expresado en porcentaxe:

```
.actual {
  background-color: hsla(208, 100%, 97%, 80%);
}
```

Casos particulares

Cabe comentar que as propiedades relativas a cor non son herdables do elemento pai, polo que CSS3 incorpora a posibilidade de forzar esa herdanza empregando o valor `currentColor`. Así, se nos escribimos:

```
.actual {
  background-color: currentColor;
}
```

Estaremos a indicar que os elementos pertencentes á clase `actual` herdarán a cor de fondo definida para o elemento que os contén (o seu elemento pai).

Outro caso particular é o de definir unha cor puramente transparente algo que, se ven podemos definir mediante as escalas **rgba** e **hsla** tamén podemos realizar empregando a palabra reservada `transparent`. Vemos, no exemplo seguinte, como definir unha cor de fondo completamente transparente para os elementos pertencentes á clase `actual`:

[15] A escala HSL (Hue Saturation Lightness) representa o ton coma o número de graos sexasemais nun círculo de cor, e a saturación e a luminosidade coma porcentaxes.

```
.actual {
  background-color: transparent;
}
```

Propiedades básicas

A continuación citaremos as propiedades CSS que empregan cor e que son de uso máis habitual no formatado de páxinas web, centrándonos naquelas con maior soporte nos navegadores máis empregados:

- **color:** xa introducida no punto anterior, como sabemos determina a cor do texto (e todas as propiedades do mesmo que impliquen cor) e, senón se indica o contrario a través das propiedades específicas para os bordes, tamén determina a cor dos mesmos.
- **background-color**[16]**:** determina a cor de fondo de elementos de tipo *block*.
- **border-color:** combínase co resto de propiedades relativas aos bordos (estilo, largura e posición) para establecer a cor dos mesmos. Admite cinco variantes segundo afectemos a todos os bordos do elemento ou só a un concreto, denominándose estas últimas:
  - *border-top-color:* cor do borde superior.
  - *border-right-color:* cor do borde dereito.
  - *border-left-color:* cor do borde esquerdo.
  - *border-bottom-color:* cor do borde inferior.

```
.caixa {
  color: #6dba81;
  background-color: rgb(203, 112, 163, 0.7);
  border-color: hsl(270, 35%, 43%);
}
```

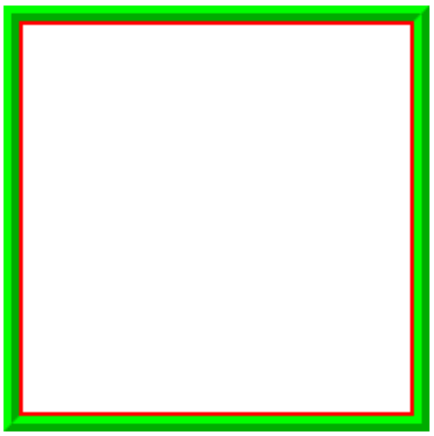
No exemplo directamente superior aplicamos cores empregando distintas escalas ao texto, fondo e bordos daqueles elementos do documento HTML que pertenzan á clase `caixa`.

- **outline-color:** define a cor da contorna dun elemento. A diferenza fundamental con respecto aos bordes é que as contornas non admiten tantas características de formato e, ademais, ocupan espazo fora do elemento, no canto de dentro (coma ocorre con `border`).

[16] Para definir cores de fondo nos elementos pode ser empregada de xeito equivalente a propiedade `background`.

```
#cadrado {
  width: 200px;
  height: 200px;
  border-width: 2px;
  border-style: solid;
  border-color: red;
  outline-width: 8px;
  outline-style: ridge;
  outline-color: lime;
}
```

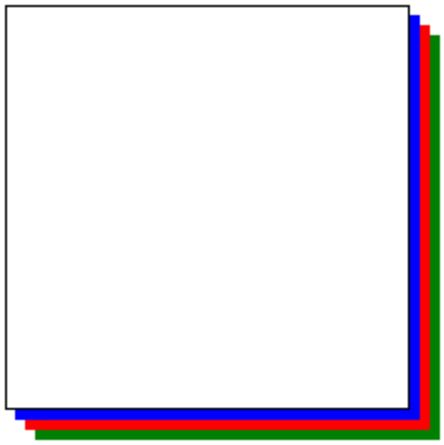
O resultado de aplicar o anterior a un `div` con identificador `cadrado` será algo semellante ao que sigue:



- **box-shadow:** de xeito semellante a como funciona a propiedade text-shadow para textos, podemos empregar estoutra propiedade para engadir sombra aos elementos de tipo bloque. Admite os seguintes valores:
    - *h-offset:* desprazamento da sombra no eixo horizontal. Un valor positivo significará cara á dereita, e un valor negativo cara á esquerda.
    - *v-offset:* desprazamento da sombra no eixo vertical. Un valor positivo significará cara abaixo, e un valor negativo cara arriba.
    - *blur:* radio de esborranchamento (difusión) da sombra.
    - *spread:* radio de esparexemento da sombra.
    - *color:* cor da sombra.

Se consideramos a aplicación do seguinte estilo a un `div` con identificador `cadrado` podemos ver como aplicar múltiples sombras aniñadas a un mesmo elemento:

```
#cadrado2 {
  border: 1px solid black;
  width: 200px;
  height: 200px;
  box-shadow: 5px 5px blue, 10px 10px red, 15px 15px green;
}
```



Fondos

Para comprender como se aplican aos elementos de tipo `block` propiedades tales como aquelas que afectan ás cores/imaxes de fondo, bordos e marxes internas (`padding`) é preciso comprender os tres tipos de caixas (`boxes`) que manexa o estándar CSS3:

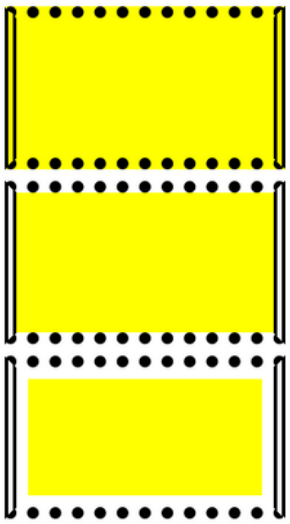
- ***Content-box:*** é o espazo interno do elemento no que se organizan os seus contidos (sen contar desbordamentos) unha vez descontado o espazo ocupado polos bordes e ao `padding`.
- ***Padding-box:*** é o espazo ocupado polas marxes internas, que vai formar un marco en torno á *content-box*.
- ***Border-box:*** é o espazo ocupado polos bordes do elemento, que formará un marco en torno á *padding-box*.

Para determinar cal vai ser o noso fondo e a área de influencia do mesmo, é dicir, a cal das tres caixas afectaremos, empregamos as seguintes propiedades (a maiores das xa tratadas con anterioridade):

- **background-clip:** especifica se o fondo dun elemento (xa sexa cor ou imaxe) esténdese ou non por debaixo do borde e até onde do mesmo faino.
  Podemos ver un exemplo do comportamento no código seguinte, supoñamos aplicado a tres divs:

```
div {
  border: 10px black;
  border-style: dotted double;
  margin: 10px;
  padding: 10px;
  background: yellow;
  width: 200px;
  height: 100px;
 }

.caixa1 {
  background-clip: border-box;
}

.caixa2 {
  background-clip: padding-box;
}

.caixa3 {
  background-clip: content-box;
}
```

Resultando algo semellante ao que sigue, onde podemos apreciar onde establece o final do `background` a propiedade `background-clip` en cada caso.



- **background-image:** permite establecer unha ou máis imaxes de fondo para un elemento, en calquera dos formatos soportados[17] polos navegadores.

[17] En xeral: APNG, AVIF, GIF, JPG, PNG, SVG, WEBP, BMP e ICO. Cada un coas súas variantes de extensión habituais.

- **background-size:** determina o tamaño do fondo xa sexa en valores absolutos ou con respecto ao contedor.

  Neste exemplo podemos ver a aplicación práctica das dúas propiedades anteriores a 6 `divs` con cadanseu identificador.

```
div {
  border: 1px solid black;
  width: 200px;
  height: 100px;
  margin: 10px;
  background-image: url(google.png);
}

#caixa1 {
  /* Non definimos ningún tipo de adaptación e a imaxe amosa o seu tamaño natural.
*/
}

#caixa2 {
  background-size: 200px 100px; /* Aplica as dimensións indicadas (largo x ancho)
ao fondo. */
}

#caixa3 {
  background-size: 100%; /* Escala a imaxe con respecto ás dimensións do contedor.
*/
}

#caixa4 {
  background-size: contain; /* Adapta ao tamaño dispoñible e aplica repetición en
mosaico. */
}

#caixa5 {
  background-size: cover; /* Adapta ao maior tamaño posible dentro do contedor,
cortando o desbordamento nunha dimensión en caso de existir. */
}

#caixa6 {
  background-size: 50% 70%; /* Adapta con respecto ás dimensións do contedor
especificando un largo e un alto e aplica repetición en mosaico. */
}
```

Considerando que a imaxe empregada ten 460x248 píxeles de resolución natural, obteremos algo semellante ao que sigue:

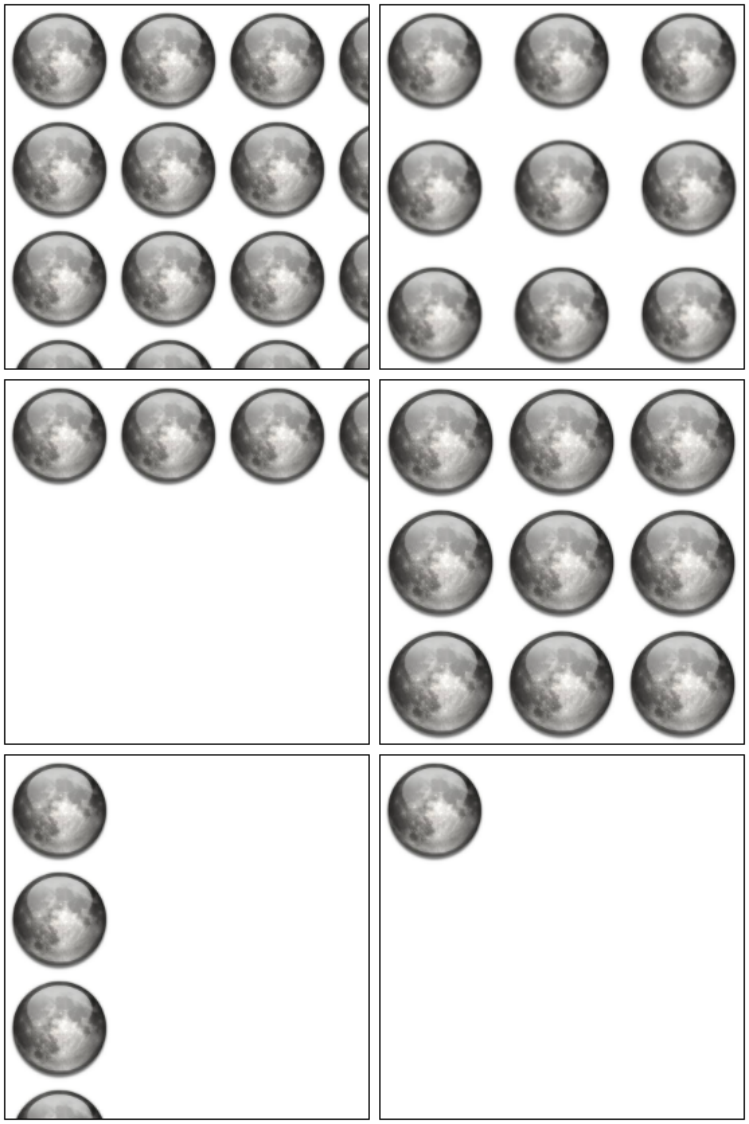
- **background-repeat:** determina a existencia ou non de repetición en mosaico dos fondos e, no caso de habela, o xeito en que esa repetición prodúcese. Pode tomar os seguintes valores:
    - *repeat:* o fondo repítese en mosaico ao longo de ambos eixos.
    - *repeat-x:* o fondo repítese só ao longo do eixo horizontal.
    - *repeat-y:* o fondo repítese só ao longo do eixo vertical.
    - *space:* a imaxe repítese o máximo posible sen recortes. A primeira e última imaxes fíxanse a cada lado do elemento, distribuíndo de xeito equitativo o espazo en branco entre as imaxes.
    - *round:* as imaxes repetiranse en mosaico deformándose para ocupar o espazo dispoñible namentres non haxa espazo para unha nova copia (isto ocorrerá cando haxa máis da metade do espazo preciso).
    - *no-repeat:* non hai repetición.

Así mesmo, esta propiedade admite a recepción de dous valores, onde o primeiro pasará a referirse á repetición no eixo horizontal e o segundo á repetición na vertical.

Partimos pois dun exemplo no que dispoñemos de 6 `divs` aos que aplicamos o mesmo fondo con distintos patróns de repetición (aplicamos posicionamento absoluto só por motivos de comodidade ao visualizar os seis obxectos):

```
div {
  border: 1px solid black;
  width: 300px;
  height: 300px;
  background-image: url(lua.jpg);
  background-size: 30%;
  position: absolute;
}

#caixa1 {
  background-repeat: repeat;
  left: 10px;
  top: 10px;
}

#caixa2 {
  background-repeat: repeat-x;
  left: 10px;
  top: 320px;
}

#caixa3 {
  background-repeat: repeat-y;
  left: 10px;
  top: 630px;
}

#caixa4 {
  background-repeat: space;
  left: 320px;
  top: 10px;
}

#caixa5 {
  background-repeat: round;
  left: 320px;
  top: 320px;
}

#caixa6 {
  background-repeat: no-repeat;
  left: 320px;
  top: 630px;
}
```

O resultado de aplicar o código CSS anterior a un documento HTML conveniente será semellante ao que sigue:



- **background-attachment:** establece se a posición da imaxe de fondo será fixa ou se desprazará ao navegar pola páxina. Admite os seguintes valores:
    - *scroll:* a imaxe moverase xunta co elemento que a contén.
    - *fixed:* a imaxe será fixa e non se moverá ao navegar a páxina, non sendo relativa ao seu contedor.

  O código CSS que figura a continuación amosa como aplicar esta propiedade ao fondo de `divs`:

```
div {
  border: 1px solid black;
  width: 300px;
  height: 300px;
  background-image: url(lua.jpg);
  margin: 10px;
}

#caixa1 {
  background-attachment: fixed;
}
```

```
#caixa2 {
  background-attachment: scroll;
}
```

- **background-origin:** establece o punto de orixe do que parte o noso fondo (hai que ter en conta que, en informática, o punto de orixe dos elementos enténdese sempre que é a súa esquina superior-esquerda). Admite os valores que figuran de seguido:
    - *border-box:* o fondo posicionarase con respecto ao borde.
    - *padding-box:* o fondo posicionarase con respecto ás marxes internas.
    - *content-box:* o fondo posicionarase con respecto á área de contidos.

  A continuación podemos ver un exemplo de código CSS no que aplicamos as tres propiedades a tres divs:

```
div {
  border: 10px dashed black;
  width: 100px;
  height: 100px;
  background-image: url(css-logo.png);
  background-repeat: no-repeat;
  background-size: 100%;
  margin: 10px;
  padding: 10px;
}

#caixa1 {
  background-origin: border-box;
}

#caixa2 {
  background-origin: padding-box;
}

#caixa3 {
  background-origin: content-box;
}
```

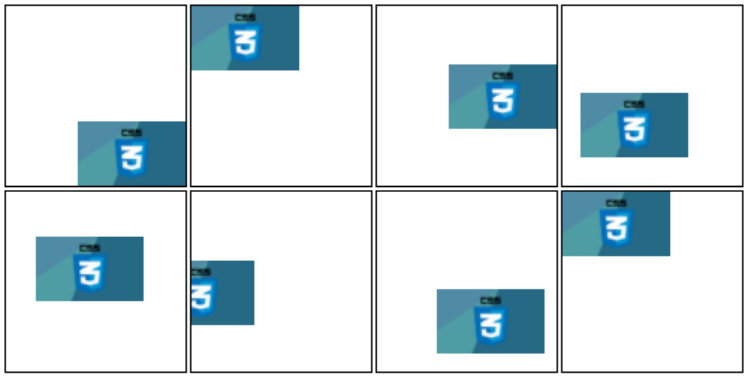
E velaquí temos o resultado, que será algo aproximadamente como isto:

- **background-position:** esta propiedade determina, coma o seu nome indica, a posición de cada imaxe de fondo. A posición será relativa á capa establecida pola propiedade anterior (`background-origin`). Soporta os seguintes valores (e as súas combinacións):
    - *x% y%:* determina as posicións horizontal e vertical colocando o píxel x%-ésimo do elemento sobre o píxel x%-ésimo do contedor, e o píxel y%-ésimo do elemento sobre o píxel y%-ésimo do contedor
    - *xpos ypos:* determina as posicións horizontal e vertical empregando magnitudes absolutas (a unidade de traballo habitual son os píxeles).
    - *left|center|right top|center|bottom:* establece as posicións horizontal e vertical empregando palabras significativas.

```
td {
  border: 1px solid black;
  width: 100px;
  height: 100px;
  background-image: url(css-logo.png);
  background-repeat: no-repeat;
  background-size: 60%;
  margin: 10px;
  padding: 10px;
}

#cela1 {
  background-position: right bottom;
}

#cela2 {
  background-position: left top;
}

#cela3 {
  background-position: right center;
```

```
}

#cela4 {
  background-position: 25% 75%;
}

#cela5 {
  background-position: 20px 30px;
}

#cela6 {
  background-position: -30px 60%;
}

#cela7 {
  background-position: left 40px bottom 15%;
}

#cela8 {
  background-position: inherit;
}
```

No exemplo anterior aplicamos distintos formatos de posicionamento de fondo para as oito celas dunha táboa HTML con dúas filas e catro columnas, resultando nalgo semellante ao que figura a continuación:



Degradados

Pola súa particularidade, compre estudar o mecanismo que CSS3 pon á nosa disposición para a representación de degradados no fondo dos elementos HTML. Así, temos varias funcións (ambas aplicables ás propiedades `background` e `background-image`), segundo o tipo de degradado que queremos realizar:

- **linear-gradient:** permite introducir degradados lineares[18]. Ten unha sintaxe flexible que permite indicar, entroutros parámetros:
    - Dirección: en caso de indicala, ten que que ser sempre o primeiro parámetro da función, admitindo os valores:
        - *to left:* o degradado esténdese de dereita a esquerda.

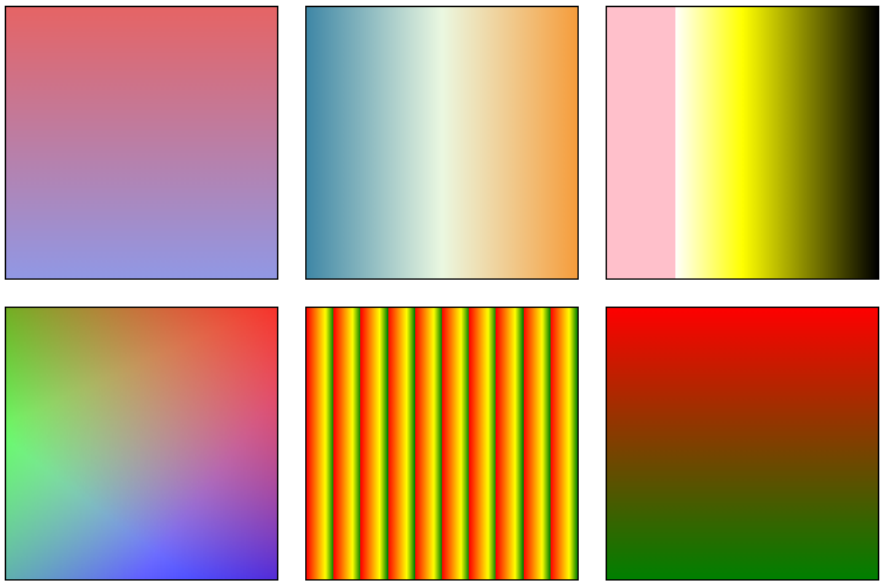
[18] Isto é, degradados nos que a variación de cor prodúcese ao longo dun eixo recto, con independencia da dirección do mesmo.

        - *to right:* o degradado esténdese de esquerda a dereita.
        - *to top:* o degradado esténdese de abaixo cara arriba.
        - *to bottom:* o degradado esténdese de arriba cara abaixo.
        - *Ángulo:* un valor (en calquera das unidades angulares, de período o de frecuencia admitidas por CSS[19]) que determina a pendente do eixo de dirección.
    - Listaxe de cores: en calquera dos formatos soportados por CSS e separados por comas. As cores poden ir seguidas dun valor (en porcentaxe ou lonxitude) que indicará en que punto do eixo detense a cor indicada.
- **repeating-linear-gradient:** é equivalente á anterior, coa diferenza de que repite o patrón de degradado ao longo do eixo.

```
table {
  border-spacing: 20px;
  border-collapse: separate;
}

td {
  border: 1px solid black;
  width: 200px;
  height: 200px;
  background-color: white; /* Unha cor por defecto para navegadores que non
soporten degradados. */
}

#cela1 {
  background: linear-gradient(#e66465, #9198e5);
}

#cela2 {
  background: linear-gradient(0.25turn, #3f87a6, #ebf8e1, #f69d3c);
}

#cela3 {
  background: linear-gradient(to left, black, yellow 50%, white 75%, pink 75%);
}

#cela4 {
  background: linear-gradient(217deg, rgba(255,0,0,.8), rgba(255,0,0,0) 70.71%),
                        linear-gradient(127deg, rgba(0,255,0,.8), rgba(0,255,0,0)
70.71%),
                        linear-gradient(336deg, rgba(0,0,255,.8), rgba(0,0,255,0)
70.71%);
}

#cela5 {
  background-image: repeating-linear-gradient(90deg,red,yellow 7%,green 10%);
}

#cela6 {
  background-image: linear-gradient(180deg,red, green);
}
```

[19] Isto é: deg (graos sexaxesimais), grad (graos centesimais), rad (radiáns), turn (número de voltas), ms (milisegundos), s (segundos), Hz (Hertzios) ou kHz (kilohertzios).

O exemplo anterior aplica distintos estilos CSS correspondentes a degradados de fondo ás seis celas dunha táboa de dúas filas e tres columnas. O resultado será algo así:



- **radial-gradient:** permite determinar degradados radiais no fondo dos elementos. Pode empregar os seguintes parámetros:
    - position: admite os valores típicos de `background-position` (se non indicamos nada o valor por defecto é `center`).
    - shape: indica a forma do degradado, que só admite dous valores:
        - *ellipse* (o valor por defecto).
        - *circle*.
    - size: determina o tamaño, admitindo os valores:
        - *farthest-corner*: o tamaño vai do centro da forma até a esquina máis afastada (o valor por defecto).
        - *closest-corner:* o tamaño vai do centro da forma até a esquina máis próxima.
        - *closest-side:* o tamaño vai do centro da forma ao lado máis próximo.
        - *farthest-side:* o tamaño vai do centro da forma ao lado máis afastado.
    - Listaxe de cores: equivalente a coma operamos cos degradados lineares.

```
table {
  border-spacing: 20px;
  border-collapse: separate;
}

td {
  border: 1px solid black;
  width: 300px;
  height: 150px;
  background-color: white; /* Unha cor por defecto para navegadores que non
soporten degradados. */
}
```

```
#cela1 {
  background: radial-gradient(red 5%, green 15%, blue 60%);
}

#cela2 {
  background: radial-gradient(farthest-side at 60% 55%, blue, green, yellow,
black);
}

#cela3 {
  background: radial-gradient(ellipse farthest-corner at 45px 45px , #00FFFF 0%,
rgba(0, 0, 255, 0) 50%, #0000FF 95%);
}

#cela4 {
  background:  radial-gradient(16px at 60px 50% , #000000 0%, #000000 14px, rgba(0,
0, 0, 0.3) 18px, rgba(0, 0, 0, 0) 19px);
}
```

Neste exemplo aplicamos distintos degradados radiais ás catro celas dunha táboa con dúas filas e dúas columnas.

Opacidade

Merece, se cadra, un apartado de seu a propiedade **opacity**, en tanto que permite (con independencia do resto de valores noutras propiedades) definir a opacidade xeral dun elemento, a cal será herdada polos seus sucesores. A opacidade admite un único valor que será un número comprendido entre 0 (transparencia total) e 1 (opacidade total).

Supoñamos, polo tanto, que temos tres div coa mesma estrutura e fondo, todos eles irmáns nun mesmo documento HTML, aos que aplicamos o seguinte código CSS:

```
div {
  background-image: url('lua.jpg');
  background-size: cover;
  height: 300px;
  width: 300px;
  margin: 20px;
  display: inline-block;
}

#caixa1 {
  opacity: 0.9;
}

#caixa2 {
  opacity: 0.6;
}

#caixa3 {
  opacity: 0.2;
}
```

O resultado será algo aproximadamente así:

Bordes e sombras

Neste apartado do tema completaremos o tratado no punto anterior en relación a como poden afectar os estilos de CSS aos bordes e sombras dos distintos tipos de elementos que soportan esas propiedades.

Bordes

CSS soporta unha variedade de propiedades que poden ser aplicadas a bordes e, do mesmo xeito que ocorría cos fondos no caso da propiedade `background`, posúe así mesmo unha propiedade abreviada chamada `border` que nos permite aplicar todas as outras en conxunto de modo simplificado.

Entre as propiedades que afectan aos bordes, atopamos:

- **border-width:** determina a largura dos bordes e admite calquera das unidades típicas de CSS. Ten 3 modos de traballo distintos:
  - Un só valor: que aplica a todos os bordes do elemento.
  - Dous valores: aplicando o primeiro aos bordes superior e inferior, e o segundo aos bordes esquerdo e dereito.
  - Catro valores: aplicando o primeiro ao borde superior, o segundo ao borde dereito, o terceiro ao borde inferior, e o último ao borde esquerdo.
- **border-style:** determina o «estilo» (é dicir, a decoración) do borde. Admite os seguintes valores:
  - *solid*: borde de liña simple.
  - *dotted*: borde de puntos.
  - *dashed*: borde de raias.
  - *double*: borde dobre.
  - *groove:* bordo acanalado (o efecto dependerá da cor do mesmo).
  - *ridge:* bordo crestado (o efecto dependerá da cor do mesmo).
  - *inset*: bordo afundido (o efecto dependerá da cor do mesmo).
  - *outset:* bordo sobresaínte (o efecto dependerá da cor do mesmo).
  - *hidden:* borde oculto.
  - *none:* sen borde (valor por defecto).

  Igual ca anterior, permite traballar definindo un, dous ou catro valores.
- **border-color:** xa tratada no punto anterior.
- **border-radius:** permite arredondar as esquinas determinando o radio de curvatura dos vértices. Admite as unidades típicas soportadas por CSS, se ben o habitual é

traballar en píxeles. Coma as anteriores, admite traballar con distinto número de valores:

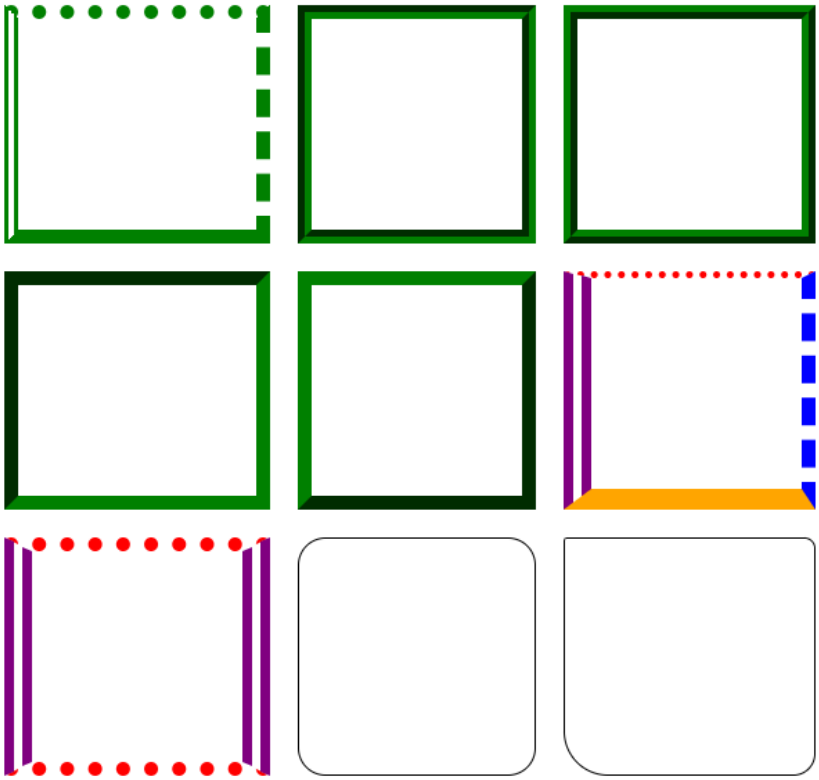
- Un valor: aplica o mesmo radio a todos os vértices.
- Dous valores: o primeiro valor aplícase aos vértices NO e SL, e o segundo valor aos vértices NL e SO.
- Catro valores: o primeiro valor aplícase ao vértice NO, o segundo ao NL, o terceiro ao SL e o cuarto ao SO.

A maiores do anterior, o propio valor de radio dun borde pode ser dual, o que permite representar bordes arredondados irregulares.

```
table {
  border-spacing: 20px;
  border-collapse: separate;
}

td {
  width: 150px;
  height: 150px;
  border-width: 10px;
  border-color: green;
}

#primeiro {
  border-top-style: dotted; /* De puntos. */
  border-right-style: dashed; /* De guións. */
  border-bottom-style: solid; /* Continuo. */
  border-left-style: double; /* Dobre. */
}

#segundo {
  border-style: groove; /* Remarcado. */
}

#terceiro {
  border-style: ridge; /* Con saínte. */
}

#cuarto {
  border-style: inset; /* Cara adentro. */
}

#quinto {
  border-style: outset; /* Cara afora. */
}

#sexto {
  border-style: dotted dashed solid double;
  border-width: 5px 10px 15px 20px;
  border-color: red blue orange purple;
}

#septimo {
  border-style: dotted double;
  border-width: 10px 20px;
  border-color: red purple;
}

#octavo {
  border: 1px solid black;
  border-radius: 20px;
```

```
}

#noveno {
  border: 1px solid black;
  border-radius: 2px 8px 16px 32px;
}
```

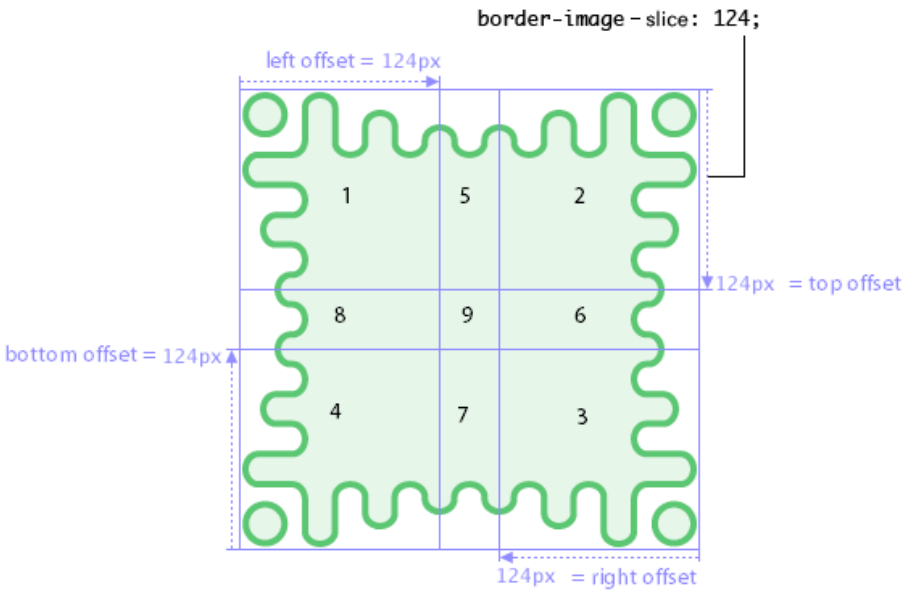
No exemplo anterior vemos distintas propiedades de bordo aplicadas ás celas dunha táboa HTML de tres filas por tres columnas. O resultado será semellante ao que sigue:



- **border-image:** coma o seu nome indica, esta propiedade permite empregar imaxes para representar os bordes de elementos. Se ben podemos empregala así, no modo unificado coñecido coma *shorthand*, tamén podemos separar esta propiedade noutras cinco:
    - **border-image-source:** a orixe do ficheiro de imaxe que empregaremos, representada mediante unha URL.
    - **border-image-slice:** divide a imaxe indicada pola propiedade anterior en nove rexións (tipo tres-en-raia). A rexión media non é empregada, aínda que podemos estender a imaxe coma imaxe de fondo até a mesma empregando a palabra reservada *fill*.

      Pode empregar até catro valores (en porcentaxe ou valor absoluto):
        - Un valor: para o conxunto.
        - Dous valores: horizontal e vertical.
        - Tres valores: superior, horizontal e inferior.
        - Catro valores: arriba, dereita, abaixo e esquerda.

Na seguinte imaxe podemos ver un exemplo desta distribución en zonas:



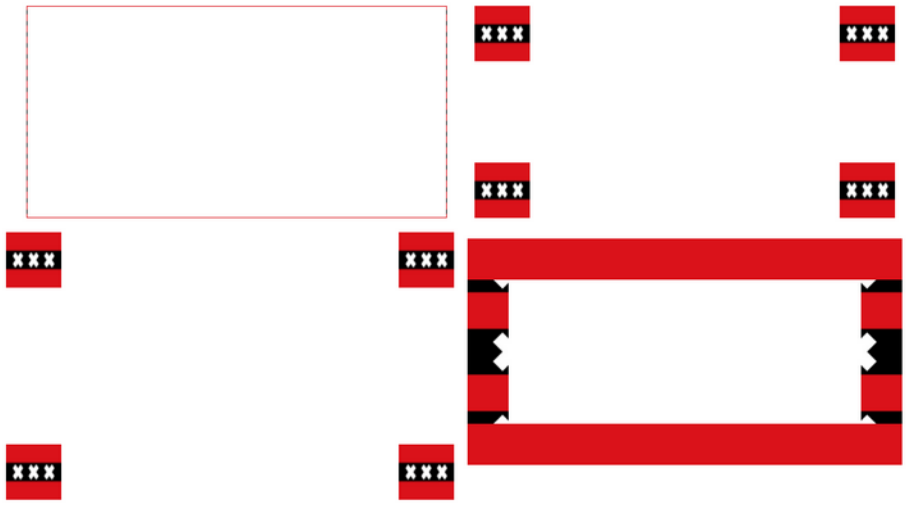
- **border-image-width:** determina, coma nos casos anteriores, a largura do borde.
- **border-image-outset:** define o xeito en que a imaxe do borde estenderase fora da caixa do elemento (sen producir desprazamento do contido). Pode empregar até catro valores:
  - Un valor: desbordamento tanto na horizontal coma na vertical.
  - Dous valores: desbordamento vertical e desbordamento horizontal.
  - Tres valores: desbordamento por arriba, desbordamento horizontal e desbordamento por abaixo.
  - Catro valores: desbordamento por arriba, desbordamento pola dereita, desbordamento por abaixo, desbordamento pola esquerda.
- **border-image-repeat:** indica o xeito en que se produce (ou non) a repetición da imaxe escollida coma borde. Pode tomar os seguintes valores:
  - *stretch:* estende a imaxe para ocupar toda a largura e altura do borde. É o comportamento por defecto.
  - *repeat:* repite a imaxe ao longo da largura e altura (pode producir cortes se o tamaño da imaxe non é múltiplo daquelas).
  - *round:* repite a imaxe ao longo da largura e altura redimensionando se é preciso.

  Esta propiedade, ademais, admite até un máximo de dous valores:
  - Un valor: aplicable á repetición en todas as dimensións:
  - Dous valores: repetición na horizontal e repetición na vertical.

Velaquí temos o código CSS que aplicariamos a unha táboa HTML de dúas filas e dúas columnas:

```
table {
  border-spacing: 20px;
  border-collapse: separate;
}

td {
  border: 1px solid black;
  width: 300px;
  height: 150px;
}

#cela1 {
  border-image: url("amsterdam.png") 50 50 50 50 repeat repeat;
}

#cela2 {
  border-image-source: url("amsterdam.png");
  border-image-width: 40px;
}

#cela3 {
  border-image-source: url("amsterdam.png");
  border-image-width: 40px;
  border-image-outset: 10px 5px 30px 15px;
}

#cela4 {
  border-image-source: url("amsterdam.png");
  border-image-slice: 20%;
  border-image-width: 30px;
  border-image-outset: 5px;
  border-image-repeat: repeat;
}
```

Sendo o resultado algo semellante ao que figura a continuación:



Layouts

En HTML (sen intervención de CSS), o fluxo natural dos elementos ocorre de esquerda a dereita e de arriba cara abaixo. Dado que ese xeito de traballar está obsoleto, e practicamente non hai deseñadores que o empreguen, desenvolvéronse en CSS distintos xeitos de «organizar» espacialmente os elementos HTML dunha páxina. A eses métodos de organización coñéceselles coma *layouts*.

Columnas

O módulo *multicolumn layout* de CSS3 permite distribuír os elementos mediante columnas de texto, como aquelas típicas dos xornais e da imprensa en xeral. No devandito módulo introdúcense dous xeitos de traballar mediante columnas determinados polas correspondentes propiedades[20]:

- **column-count:** define un número específico de columnas.
- **column-width:** define unha largura para as columnas de xeito que o navegador creará as mesmas dinamicamente en función da largura do elemento pai.

Consideremos, por exemplo, que aplicamos o código seguinte a dous `div` consecutivos:

```
div {
  margin: 20px;
  border: 1px solid gray;

  text-overflow: ellipsis;
}

#caixa1 {
  width: 300px;
  column-count: 3;
}

#caixa2 {
  width: 50%;
  column-width: 100px;
}
```

[20] Se combinamos ambas propiedades, **column-count** actuará coma valor máximo do número total de columnas dentro do elemento afectado.

O resultado, dependendo da largura da páxina, resultará como sigue (como se pode ver, as columnas do primeiro `div` teñen un comportamento estático, fronte aos do segundo, que se adaptan dinamicamente):

Duis aliquet ligula quam, sit amet malesuada risus fermentum a. Ut eu molestie lacus. Aliquam erat volutpat. Nullam at ipsum id ante egestas varius. Etiam quis pellentesque metus, quis egestas nisi. In vestibulum sit amet orci nec tempor. Cras luctus leo nec quam viverra eleifend. Maecenas in mi auctor leo commodo pharetra. Nullam commodo magna sit amet ex euismod convallis.

Duis aliquet ligula quam, sit amet malesuada risus fermentum a. Ut eu molestie lacus. Aliquam erat volutpat. Nullam at ipsum id ante egestas varius. Etiam quis pellentesque metus, quis egestas nisi. In vestibulum sit amet orci nec tempor. Cras luctus leo nec quam viverra eleifend. Maecenas in mi auctor leo commodo pharetra. Nullam commodo magna sit amet ex euismod convallis.

Se ampliamos a largura da páxina, en cambio:

Duis aliquet ligula quam, sit amet malesuada risus fermentum a. Ut eu molestie lacus. Aliquam erat volutpat. Nullam at ipsum id ante egestas varius. Etiam quis pellentesque metus, quis egestas nisi. In vestibulum sit amet orci nec tempor. Cras luctus leo nec quam viverra eleifend. Maecenas in mi auctor leo commodo pharetra. Nullam commodo magna sit amet ex euismod convallis.

Duis aliquet ligula quam, sit amet malesuada risus fermentum a. Ut eu molestie lacus. Aliquam erat volutpat. Nullam at ipsum id ante egestas varius. Etiam quis pellentesque metus, quis egestas nisi. In vestibulum sit amet orci nec tempor. Cras luctus leo nec quam viverra eleifend. Maecenas in mi auctor leo commodo pharetra. Nullam commodo magna sit amet ex euismod convallis.

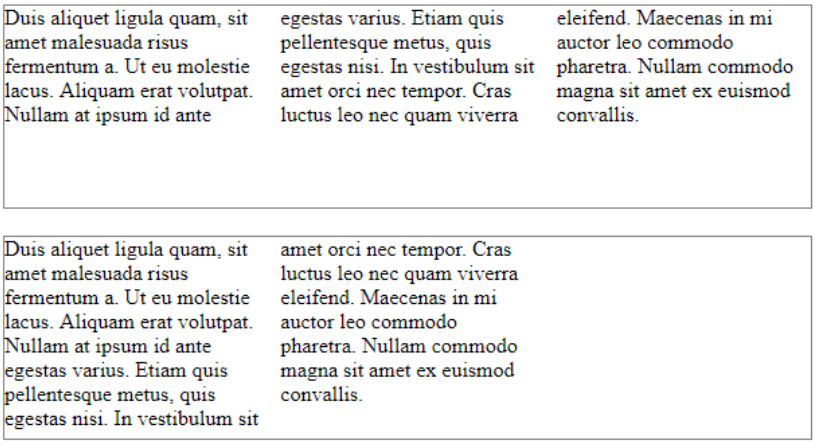
Outras propiedades de CSS relativas a columnas son:

- **columns:** permite combinar o relativo ás propiedades `column-count` e `column-width` nunha propiedade única a modo de *shorthand*, e que recibe dous valores:
  - O primeiro é a largura das columnas.
  - O segundo é o número de columnas.
- **column-fill:** permítenos establecer o xeito en que se distribúe o texto ao longo das columnas, podendo tomar os seguintes valores:
  - *balance:* enche cada columna coa mesma cantidade de contidos, sen permitirlles superar a altura establecida (é o valor por defecto da propiedade).
  - *auto:* enche cada columna (de esquerda a dereita) ocupando a altura máxima admitida até finalizar os contidos.

  No seguinte exemplo vemos unha aplicación práctica desta propiedade sobre dous `div` que conteñen o mesmo texto:

```
div {
  margin: 20px;
  border: 1px solid gray;
  height: 150px;
  width: 600px;
  column-width: 150px;
}

#caixa1 {
  column-fill: balance;
}

#caixa2 {
  column-fill: auto;
}
```

Un posible resultado vémolo na seguinte imaxe:

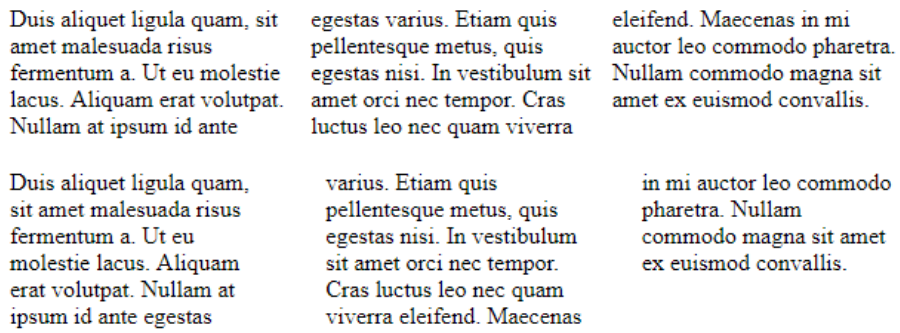


- **column-gap:** especifica o tamaño do espazo entre columnas.

  O seguinte fragmento de código exemplifica a aplicación do espazamento entre columnas a dous `div` que teñen o mesmo contido.

```
div {
  margin: 20px;
  width: 600px;
  column-count: 3;
}

#caixa1 {
  column-gap: 10px;
}

#caixa2 {
  column-gap: 40px;
}
```

O resultado será semellante a isto:



- **column-rule:** debuxa unha liña de separación entre columnas no medio do espazamento de separación. Admite tres valores:
  - O primeiro valor determina a cor da liña.
  - O segundo valor corresponde ao estilo (equivalente a aqueles vistos nos bordes).
  - O terceiro indica a largura da liña.

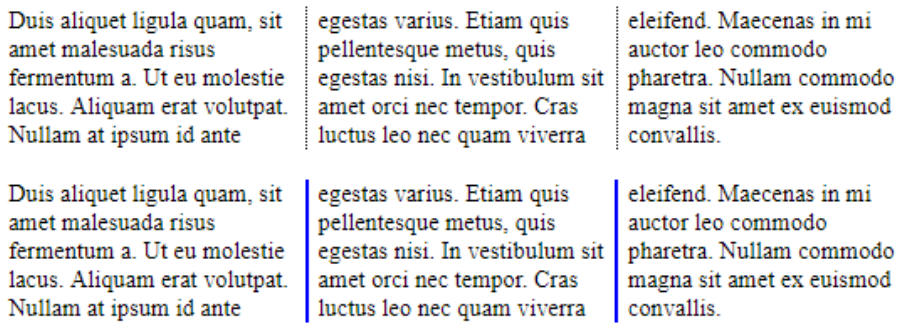
  Ademais, esta propiedade é *shorthand* das seguintes (correspondentes cos tres valores mencionados enriba):
  - **column-rule-color**.
  - **column-rule-style**.
  - **column-rule-width**.

  Aplicando o código CSS que figura a continuación a dous `div` que conteñen o mesmo texto e a mesma estrutura de columnas:

```
div {
  margin: 20px;
  width: 600px;
  column-count: 3;
}

#caixa1 {
  column-rule: black dotted 2px;
}

#caixa2 {
```

```
  column-rule-color: blue;
  column-rule-style: solid;
  column-rule-width: 3px;
}
```

Na imaxe podemos apreciar o resultado:



- **column-span:** esta propiedade posibilita indicar se un elemento se estende ao longo de todas as columnas ou se está restrinxido a unha sola. Admite dous valores:
    - *none:* o elemento vai ocupar só tanta largura máxima coma teña a primeira columna, situándose na mesma.
    - *all:* o elemento vai ocupar tanta largura máxima coma teña o conxunto de columnas, estendéndose a través de todas elas.

  O seguinte fragmento de código representa a aplicación desta propiedade a dous `div` que conteñen ambos un elemento h2 e exactamente o mesmo contido de texto:

```
div {
  margin: 20px;
  width: 600px;
  column-count: 3;
}

#titulo1 {
  column-span: all;
}

#titulo2 {
  column-span: none;
}
```

Obteremos, ao aplicar o código CSS anterior ao correspondente documento HTML algo así:

Aínda que, por motivos de simplicidade, non nos estenderemos aquí no uso das mesmas, existen tres propiedades que permiten determinar en que xeito se distribúen os elementos ao longo das columnas: **break-inside**, **break-before** e **break-after**.

Outros *layouts*

O resto de métodos de disposición de elementos en CSS poden aplicarse a través do uso da propiedade `display`. Así, salientaremos tres *layouts* de entre os diversos xeitos de traballo que posibilita a tecnoloxía:

- ***Flexible box layout***.
- ***Website layout***.
- ***Grid layout/positioning***.

Dado que os aspectos relativos a *layouts* corresponden xa ao deseño avanzado en CSS, simplemente describiremos neste apartado cada un deles así como as principais propiedades asociadas.

Flexible box layout

Este *layout*, tamén coñecido coma **FlexBox** permite redimensionar os elementos contidos naquel ao que se aplica sen necesidade de recalcular os valores da largura e da altura. Para empregar esta disposición, o elemento pai ten que empregar a propiedade `display` co valor *flex*.

En relación con este xeito de organizar os elementos atopamos as seguintes propiedades que afectan ao elemento pai:

- **flex-direction:** determina o xeito en que sentido se colocan os elementos no eixo principal (por defecto o horizontal).
- **flex-wrap:** permite indicar como se comportarán os elementos se non caben todos na mesma liña.
- **flex-flow:** *shorthand* para as dúas anteriores.
- **justify-content:** define o aliñamento dos elementos no eixo principal (por defecto o horizontal).
- **align-items:** define o aliñamento dos elementos no eixo secundario.
- **align-content:** define o aliñamento do conxunto de liñas e das propias liñas entre si con respecto ao eixo principal.

As propiedades que afectan aos elementos fillos son as seguintes:

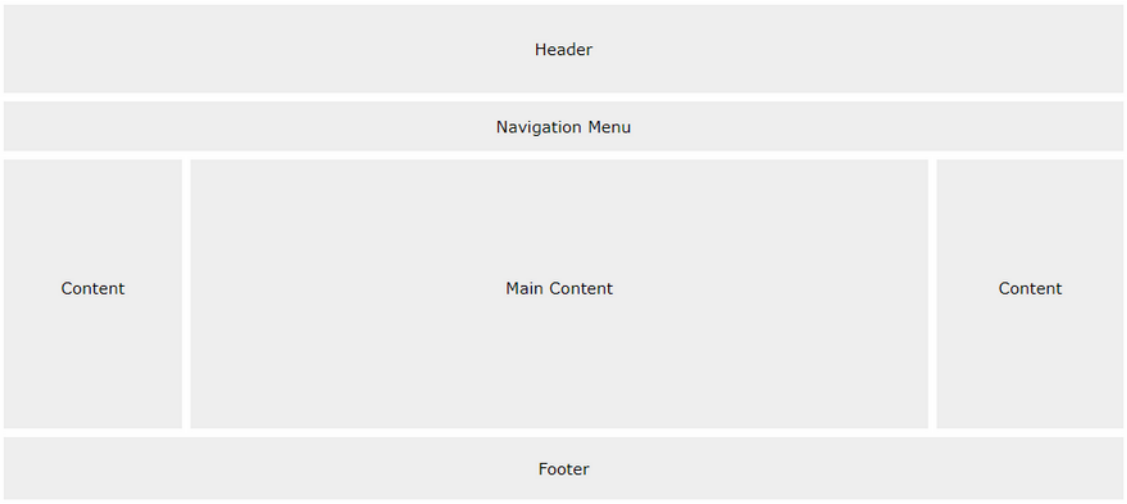
- **order:** determina a orde na que aparecerán os elementos dentro do contedor.
- **flex-grow:** indica canto poden medrar os elementos.
- **flex-shrink:** indica se os elementos poden encoller.
- **flex-basis:** permite definir o tamaño por defecto dun elemento denantes que se distribúa o espazo restante.
- **flex:** *shorthand* para as tres anteriores.
- **align-self:** permite sobreescribir a propiedade `align-items` para elementos concretos.

No seguinte enlace podemos ver unha explicación do funcionamento deste *layout* segundo o titorial oficial de CSS Tricks.

Website layout

É este un dos xeitos tradicionais de estruturar as páxinas web, para o que empregamos, principalmente as propiedades `display` (con valor *block*) e `float` (que admite os valores *left* e *right*), aínda que tamén pode ser resolto mediante a emprega de columnas, tal e coma vimos ao comezo deste apartado. Este *layout* é aplicable tanto a páxinas completas coma a elementos concretos.

Á continuación vemos unha distribución típica que emprega un *layout* deste tipo, tal e coma nos sinala a web oficial do W3C:



Por motivos de simplicidade non engadiremos exemplos aquí, pero podemos atopar unha estensa explicación do método no titorial oficial de CSS da W3C.

Grid layout

O módulo *grid layout* baséase na división conceptual da páxina ou elemento ao que queremos aplicarlle esta organización en filas e columnas. Con ese fin introduce unha nova unidade (*fr*, do inglés *fraction*) e emprega as seguintes propiedades:

- No elemento pai:
  - **display:** con valor *grid*.
  - **grid-columns:** no que se indica a cantidade e largura relativa das columnas.
  - **grid-rows:** no que se indica o número e altura das filas.
- Nos elementos fillos:
  - **grid-column**[21]**:** indica a posición entre as columnas.
  - **grid-column-span:** indica cantas columnas consecutivas ocupa o elemento.
  - **grid-row**[22]**:** indica a posición entre as filas.

No seguinte enlace pode verse un exemplo práctico de uso deste *layout*, no titorial oficial de CSS de MDN Web Docs.

A propiedade `position`

Dada a súa importancia e o habitual do seu uso, a propiedade `position` merece un punto propio dentro deste apartado relativo á organización dos elementos nas páxinas web. Dependendo do valor seleccionado para o elemento afectado, acompañarase (ou non) das propiedades `top`,

[21] Esta propiedade é shorthand para as propiedades **grid-column-start** e **grid-column-end**.

[22] Esta propiedade é shorthand para as propiedades **grid-row-start** e **grid-row-end**.

`right`, `bottom` e `left` para definir a posición desexada no plano da páxina, e da propiedade `z-index` para determinar o «amoreamento» dunhas capas sobre outras.

Esta propiedade admite os seguintes valores:

- *static:* o elemento posiciónase seguindo o fluxo natural dos documentos HTML (é o valor por defecto). O uso deste valor non é compatible co uso das propiedades `top`, `right`, `bottom` e `left`.
- *relative:* o elemento sitúase con respecto á súa posición natural (coma en *static*) pero admitindo o uso das propiedades `top`, `right`, `bottom` e `left` para «recolocalo». Ese reposicionamento non afecta ao resto de elementos (aínda que si, coma é obvio, a aqueles que estean contidos no elemento reposicionado), polo que a propiedade `z-index` entra en xogo, é dicir, os elementos sitúanse en capas separadas, podendo producirse solapamento entre os mesmos.
  Supoñamos que aplicamos o seguinte fragmento de código CSS a catro `div` nomeados caixaA, caixaB, caixaC e caixaD que son todos irmáns.

```
div {
  display: inline-block;
  width: 100px;
  height: 100px;
  background-color: #cc6699;
  color: white;
}

#caixaB {
  position: relative;
  top: 20px;
  left: 20px;
  background-color: #ffcc66;
}
```

O resultado será algo como o que sigue:

- *absolute:* o elemento xa non sigue o fluxo natural do documento, posicionándose en relación ao seu ancestro posicionado máis cercano (xeralmente o seu elemento pai). A posición, polo tanto, vai estar totalmente determinada polas propiedades `top`, `right`, `bottom` e `left` no plano do documento, e por `z-index` no eixo perpendicular ao mesmo.

Na mesma liña, aplicamos o seguinte código ao mesmo documento HTML:

```
div {
  display: inline-block;
  width: 100px;
  height: 100px;
  background-color: #cc6699;
  color: white;
}

#caixaB {
  position: absolute;
  top: 20px;
  left: 20px;
  background-color: #ffcc66;
}
```

O resultado será igual ao que figura de seguido:

- *fixed:* o elemento posiciónase con respecto ao *viewport*, é dicir, o navegador *per se*. Dese xeito, cando se fai *scroll* o elemento permanece sempre na mesma posición relativa á xanela do navegador, con independencia do resto do documento HTML. Igual ca no caso anterior, a posición vai estar totalmente determinada polas propiedades `top`, `right`, `bottom` e `left` (neste caso no plano da xanela do navegador), e por `z-index` no eixo perpendicular ao mesmo.

  O código CSS aplícase a dous `div`, o primeiro deles (*pai*) contén un texto longo, sendo o seu último fillo o segundo elemento (*fillo*). É preciso cargalo nun navegador para apreciar o resultado, polo que non incluímos captura do mesmo:

```
#pai {
  width: 500px;
  height: 300px;
  overflow: scroll;
  padding-left: 150px;
}

#fillo {
  width: 100px;
  height: 100px;
  background: red;
  color: white;
  position: fixed;
  top: 80px;
  left: 10px;
  background-color: #ffcc66;
}
```

- *sticky:* o elemento comportarase coma se tivera posicionamento *static* pero mantendo a posición *fixed* sobre o seu elemento pai conforme facemos *scroll* até que este último desapareza da vista.

  Se aplicamos o código CSS seguinte a un `div` que conteña unha `dl` cun único `dt` e múltiples `dd` teremos:

```
div {
  height: 150px;
  overflow: scroll;
  padding-left: 150px;
  font-size: 20pt;
  padding: 5px;
}

dt {
  background-color: #B8C1C8;
  border: 1px solid #989EA4;
  position: sticky;
  top: 0px;
}
```

  Igual ca no caso anterior, non incluímos aquí exemplo resultado visual do uso deste valor, en tanto que para apreciala é preciso cargar a páxina nun navegador (poderiamos tomar o código anterior e substituír *fixed* por *sticky* e engadir moito contido –coma texto de enchemento- despois do elemento *pai* para poder realizar *scroll* e apreciar o efecto).

Transformacións, transicións e animacións

Antes da introdución distintos mecanismos de «efectos» visuais, practicamente o unico xeito de empregar os mesmos en páxinas web era empregando JavaScript ou tecnoloxías de terceiros (coma a agora defunta Flash). Por sorte, CSS3 introduciu tres novos modos de xerar efectos visuais, sendo estes:

- **Transformacións.**
- **Transicións.**
- **Animacións.**

Transformacións

O módulo **CSS 2D Transforms** posibilita a incorporación nas páxinas web de diversas transformacións que poden ser aplicadas aos elementos HTML sen necesidade da intercalación de terceiras tecnoloxías.

As transformacións CSS non son transitivas, isto é, afectan unicamente ao elemento indicado o que permite manter intactos tanto o *layout* coma o resto de elementos HTML do documento.

Propiedades

Para incorporar unha transformación empregamos na maioría de casos a propiedade **transform**, dándolle coma valor unha función que recibirá, ao seu tempo, unha serie de parámetros dependendo do obxectivo da mesma. Esas funcións son as que figuran a continuación:

- **none:** é un valor, non unha función, e indica que non hai transformación (é o valor por defecto).
- **matrix(a, b, c, d, tx, ty):** aplica os seis parámetros recibidos para deformar (os catro primeiros) e trasladar (os dous últimos) o elemento.
  O seguinte fragmento de código aplícase a un documento HTML cun único `div`:

```
div {
  background-image: url(lua.jpg);
  background-size: cover;
  width: 300px;
  height: 300px;
}

div:hover {
  transform: matrix(1, 2, 1, 1, 100, 200);
}
```

- **matrix3d(a1, b1, c1, d1, a2, b2, c2, d2, a3, b3, c3, d3, a4, b4, c4, d4):** semellante ao anterior, onde os primeiros tres valores de cada letra representan a deformación, e os que levan numeral 4 representan a traslación. Pódese atopar un bó exemplo do uso desta función no **Titorial oficial de CSS de MDN Web Docs**.
- **translate(x, y):** traslada no plano o elemento segundo os valores indicados nos eixos X e Y.
- **translateX(x):** traslada o elemento só no eixo X.
- **translateY(y):** traslada o elemento só no eixo Y.

  A continuación vemos un exemplo de aplicación das tres funcións anteriores sobre tres `div` contidos nun mesmo documento HTML:

```
div {
  background-image: url(lua.jpg);
  background-size: cover;
  width: 300px;
  height: 300px;
}

#cadro1:hover {
  transform: translate(20px, 50px);
}

#cadro2:hover {
  transform: translateX(30px);
}

#cadro3:hover {
  transform: translateY(-80px);
}
```

- **translateZ(z):** traslada o elemento só no eixo Z. No **Titorial oficial de CSS de MDN Web Docs** atopamos diversos exemplo do seu uso.
- **translate3d(x,y,z):** traslada no espazo o elemento segundo os valores indicados nos eixos X, Y e Z. Unha vez máis, o **Titorial oficial de CSS de MDN Web Docs** contén un exemplo axeitado para comprender o uso desta función.
- **scale(x,y)**: escala o elemento no plano (con deformación) o indicado nos eixos X e Y.
- **scaleX(x):** escala o elemento só no eixo X (con deformación).
- **scaleY(y):** escala o elemento só no eixo Y (con deformación).

  Neste exemplo podemos ver a aplicación das tres funcións anteriores aos elementos dun documento HTML:

```
div {
  background-image: url(lua.jpg);
  background-size: cover;
  width: 300px;
  height: 300px;
}

#cadro1:hover {
  transform: scale(0.8, 2);
```

```
}

#cadro2:hover {
  transform: scaleX(3);
}

#cadro3:hover {
  transform: scaleY(0.5);
}
```

- **scaleZ(z):** escala o elemento só no eixo Z (con deformación). Os exemplos do **Titorial oficial de CSS de MDN Web Docs** resultan prácticos para comprender a súa operativa.
  scale3d(x,y,z): escala o elemento no espazo (con deformación) o indicado nos eixos X, Y e Z. Obsérvese, coma nos casos anteriores referidos a función ámbito tridimensional, os exemplos de uso presentes no **Titorial oficial de CSS de MDN Web Docs**.
- **rotate(ángulo):** rota o elemento no plano sobre o seu centro de gravidade en base ao ángulo recibido coma parámetro.
- **rotateX(ángulo):** rota en 3D en torno ao eixo X en base ao ángulo recibido coma parámetro.
- **rotateY(ángulo):** rota en 3D en torno ao eixo Y en base ao ángulo recibido coma parámetro.
- **rotateZ(ángulo):** rota en 3D en torno ao eixo Z en base ao ángulo recibido coma parámetro. En páxinas bidimensionais (sen 3D) o seu comportamento é análogo a **rotate(ángulo)**.
- **rotate3d(x, y, z, ángulo):** rota o elemento no espazo sobre o seu centro de gravidade en base aos multiplicadores nos eixos X, Y e Z e o ángulo recibidos coma parámetros, é dicir, rotará tanto en cada eixo coma resulte de multiplicar o parámetro correspondente polo ángulo indicado.
  Neste exemplo, unha vez máis aplicado aos distintos div contidos nun mesmo documento HTML, podemos ver unha demostración de uso das tres últimas funcións de transformación descritas:

```
div {
  background-image: url(lua.jpg);
  background-size: cover;
  width: 300px;
  height: 300px;
}

#cadro1:hover {
  transform: rotate(0.2turn);
}

#cadro2:hover {
  transform: rotateX(30deg);
}

#cadro3:hover {
  transform: rotateY(0.6turn);
}
```

```
#cadro4:hover {
  transform: rotate3d(2,1,-1,60deg);
}
```

- **skew(ánguloX, ánguloY):** define unha deformación no plano a partires dos ángulos (sexaxesimais) indicados sobre os eixos X e Y.
- **skewX(ángulo):** define unha deformación sobre o eixo X en base ao ángulo pasado coma parámetro.
- **skewY(ángulo):** define unha deformación sobre o eixo Y en base ao ángulo pasado coma parámetro.

  A continuación figura o código CSS correspondente ao exemplo de aplicación destas tres últimas funcións:

```
div {
  background-image: url(lua.jpg);
  background-size: cover;
  width: 300px;
  height: 300px;
}

#cadro1:hover {
  transform: skew(0.2turn, 20deg);
}

#cadro2:hover {
  transform: skewX(40deg);
}

#cadro3:hover {
  transform: skewY(0.6turn);
}
```

Asociadas tamén ao concepto de transformación atopamos outras propiedades que poden combinarse coas anteriores, como por exemplo:

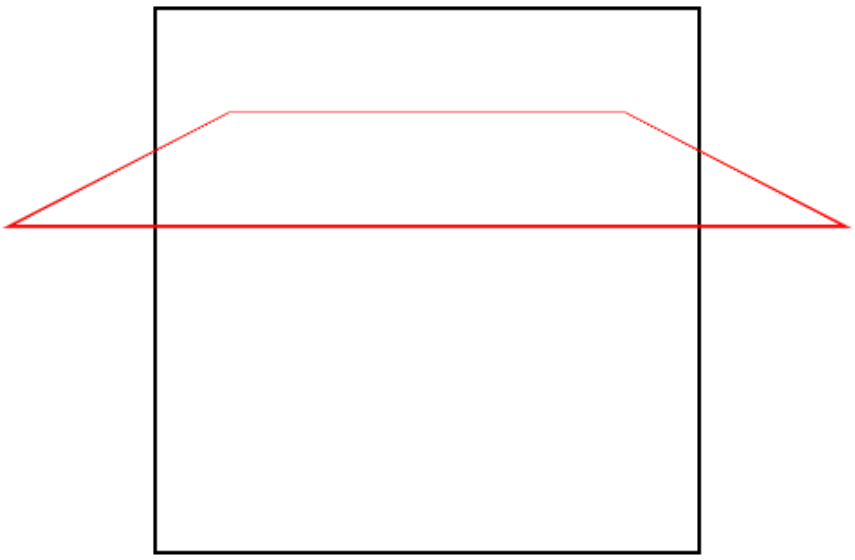
- **perspective:** permite determinar a distancia visual entre o usuario e o plano Z=0 no espazo. Iso vai posibilitar mudar a perspectiva que o usuario teña dos obxectos tridimensionais.

  O fragmento de código CSS que figura a continuación aplícase a un documento HTML con dous div, un deles contido dentro do outro:

```
#pai {
  height: 300px;
  width: 300px;
  margin: 200px;
  border: 2px solid black;
  perspective: 150px;
}

#fillo {
  padding: 25%;
  border: 2px solid red;
```

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

E producirá un resultado semellante ao que figura a continuación:

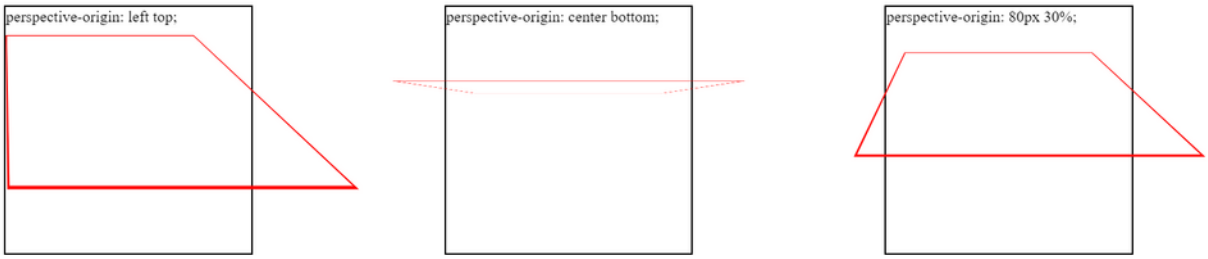


- **perspective-origin:** complementa á propiedade anterior definindo o punto de vista do usuario, é dicir, dende onde está a verse o elemento nun contexto tridimensional. Pode recibir un valor (os habituais *initial* e *inherit*) ou dous valores, en cuxo caso:
    - O primeiro valor representa o punto de vista na horizontal, e pode valer:
        - *left:* o usuario observa dende a esquerda.
        - *center:* o usuario observa dende o centro.
        - *right:* o usuario observa dende a dereita.
        - *lonxitude:* indícase en calquera das unidades de lonxitude propias de CSS.
        - *porcentaxe:* sendo o seu valor por defecto 50%.
    - O segundo valor representa o punto de vista na vertical, podendo ser:
        - *top:* o usuario observa dende arriba.
        - *center:* o usuario observa dende o centro.
        - *bottom:* o usuario observa dende abaixo.
        - *lonxitude:* indicada en calquera das unidades de lonxitude propias de CSS.
        - *porcentaxe:* sendo o seu valor por defecto 50%.

Velaquí algúns exemplos de aplicación desta propiedade partindo do exemplo anterior (supoñamos agora que temos tres `div` con cadanseu `div` fillo:

```
.pai {
  position: absolute;
  top: 100px;
  height: 250px;
  width: 250px;
  border: 2px solid black;
  perspective: 150px;
}

#pai1 {
  left: 100px;
  perspective-origin: left top;
}

#pai2 {
  left: 550px;
  perspective-origin: center bottom;
}

#pai3 {
  left: 1000px;
  perspective-origin: 80px 30%;
}

.fillo {
  padding: 25%;
  border: 2px solid red;
  transform: rotateX(45deg);
}
```

Ficando o resultado semellante a isto:



- **transform-origin:** esta propiedade posibilita cambiar a posición dos elementos transformados. Pode recibir dous ou tres valores:
    - Dous valores:
        - O primeiro representa a transformación no eixo X, en forma de lonxitude ou porcentaxe (o seu valor por defecto é 50%).
        - O segundo representa a transformación no eixo Y, en forma de lonxitude ou porcentaxe (o seu valor por defecto é 50%).
    - Tres valores: coincidindo os dous primeiros e engadindo un terceiro que representa a transformación no eixo X, en forma de lonxitude ou porcentaxe (o seu valor por defecto é 0).

O ideal para apreciar o comportamento desta propiedade é tomar unha mesma transformación e aplicarlle distintas orixes. Neste caso partimos do exemplo relativo á función **skew()**:

```
div {
  background-image: url(lua.jpg);
  background-size: cover;
  width: 300px;
  height: 300px;
}

#cadro1:hover {
  transform: skew(0.2turn, 20deg);
  transform-origin: 50% 50%;
}

#cadro2:hover {
  transform: skew(0.2turn, 20deg);
  transform-origin: 20px 130px;
}

#cadro3:hover {
  transform: skew(0.2turn, 20deg);
  transform-origin: 20% 80%;
}
```

- **transform-style:** determina se os elementos fillos preservarán ou non a súa posición tridimensional. Admite, ademais dos valores típicos, estes dous:
    - *flat:* os elementos fillos non preservarán a posición tridimensional (é o valor por defecto).
    - *preserve-3d:* os elementos fillos preservarán a posición tridimensional.

  O resultado de aplicar esta propiedade sobre obxectos tridimensionais podémolo apreciar no **Titorial oficial de CSS de MDN Web Docs**.
- **transform-box:** define a «caixa de referencia» á cal aplícanse as propiedades **transform** e **transform-origin**, isto é, a súa área de actuación. Admite os valores seguintes:
    - *content-box:* a área de actuación é a caixa de contidos.
    - *border-box:* a área de actuación é a caixa definida polos bordes.
    - *fill-box:* a área de actuación é a caixa de enchemento, é dicir, aquela que contén a súa forma xeométrica.
    - *stroke-box:* a área de actuación é a caixa que delimita a súa forma xeométrica (incluíndo os bordes).
    - *view-box:* a área de actuación é o *viewport*, é dicir, a xanela do navegador.

  O **Titorial oficial de CSS de Quackit** contén un bó exemplo de uso desta propiedade no cal podemos probar o resultado dos distintos valores da mesma.

Transicións

As **transicións** de CSS, introducidas a través do módulo **CSS Transitions**, permítenos crear efectos visuais que ocorren de xeito gradual ao longo do tempo, en contraposición cos cambios bruscos que suceden cando empregamos transformacións ou *hover*, por exemplo, sen ningún tipo de suavizado. A aplicacións de transicións ten efecto na maioría de propiedades que teñen a ver co aspecto visual dos elementos.

Propiedades

En CSS as transicións defínense a partir do seguinte xogo de propiedades:

- **transition-delay:** especifica a cantidade de tempo que se vai esperar entre a solicitude do cambio de estado e o comezo do efecto de transición. O seu valor por defecto é *0s*, o que implica que os efectos inician no intre exacto do evento que os desencadea namentres non indiquemos o contrario. Esta propiedade admite múltiples valores separados por comas, no caso de que esteamos a aplicar retardos distintos a diferentes efectos.
  Por exemplo, se aplicamos o seguinte código CSS a un documento HTML con tres `div` irmáns con cadanseu identificador:

```
div {
  background-image: url('lua.jpg');
  background-size: cover;
  height: 300px;
  width: 300px;
  margin: 20px;
  display: inline-block;
}

#caixa1:hover {
  opacity: 0;
  transition-delay: 500ms;
}

#caixa2:hover {
  opacity: 0;
  transition-delay: 2s;
}

#caixa3:hover {
  opacity: 0;
  transition-delay: 4s;
}
```

  Observaremos como os elementos van «desaparecendo» (asumindo unha transparencia total) ao cabo de 500ms, 2s e 4s respectivamente dende o intre en que colocamos o cursor do rato enriba deles.
- **transition-duration:** esta propiedade determina o tempo que tarda o efecto en completarse dende que é desencadeado. O seu valor por defecto é *0s*, o que implica

que non se aprecia efecto ningún, senón que o cambio é instantáneo. Do mesmo xeito ca anterior, esta propiedade admite múltiples valores separados por comas, a aplicar ás distintas duracións dos efectos implicados.

Partindo do exemplo anterior, tomamos o mesmo documento HTML e aplicámoslle o código CSS seguinte:

```
div {
  background-image: url('lua.jpg');
  background-size: cover;
  height: 300px;
  width: 300px;
  margin: 20px;
  display: inline-block;
}

#caixa1:hover {
  opacity: 0;
  transition-duration: 100ms;
}

#caixa2:hover {
  opacity: 0;
  transition-duration: 1s;
}

#caixa3:hover {
  opacity: 0;
  transition-duration: 2s;
}
```

O resultado, como apreciaremos se construímos o tal documento, será un mesmo efecto de transparencia ao pasar o rato por riba dos elementos, e aplicado a tres velocidades distintas, sucesivamente máis rápidas.

- **transition-property:** permite indicar o nome da propiedade ou propiedades ás cales o efecto de transición debe aplicarse. Admite, polo tanto, múltiples valores, entre os que destacamos:
    - Listaxe das propiedades afectadas, separados por comas.
    - *none:* se non afecta a ningunha propiedade.
    - *all:* se afecta por igual a todas.

  O seguinte fragmento de código CSS aplícase a un único `div`, combinando as tres propiedades anteriores para unha mellor comprensión do seu impacto combinado sobre múltiples propiedades en transformación:

```
div {
  background-image: url('lua.jpg');
  background-size: cover;
  height: 300px;
  width: 300px;
  margin: 20px;
  display: inline-block;
}
```

```
#caixa:hover {
  opacity: 0;
  height: 400px;
  width: 400px;
  transition-delay: 0s, 0s, 0.5s;
  transition-duration: 1s, 1s, 2s;
  transition-property: height, width, opacity;
}
```

O resultado, polo tanto, será un `div` que, sen retardo, mudará de dimensións durante 1 segundo e que, ao cabo de medio segundo, tornará completamente transparente de xeito gradual ao longo de 2 segundos.

- **transition-timing-function:** sinala a función do tempo que seguirá o efecto, isto é, o xeito en que se desenvolverá o mesmo ao transcorrer a súa duración. Admite múltiples valores separados por comas (para múltiples efectos) e, cada un deles, pode posuír un dos seguintes valores:
  - *ease:* é o valor por defecto. Consiste en acelerar o efecto cara á metade da transición, suavizando contra o final.
  - *linear:* a transición prodúcese linearmente, isto é, con velocidade constante a través do tempo.
  - *ease-in:* a transición inicia a modo acelerando conforme avanza o tempo.
  - *ease-out:* a transición inicia rápido e vai aminorando ao longo do tempo.
  - *ease-in-out:* o efecto inicia a modo, acelera contra a metade e decelera novamente conforme vai rematar.
  - *cubic-bezier(p1, p2, p3, p4):* determina a curva de Bezier cúbica que seguirá o efecto, onde os parámetros p1 e p3 terán un valor entre 0 e 1, podendo p2 e p4 tomar calquera valor real.
  - *steps(n, tipo_de_salto):* a transición desenvólvese ao longo de n pasos, deténdose durante a mesma cantidade de tempo entre paso e paso. A maiores, o parámetro *tipo_de_salto* determina o comportamento deses pasos e pausas, podendo valer:
    - *jump-start* ou *start:* o primeiro salto prodúcese tan pronto inicia a transición.
    - *jump-end* ou *end:* o último salto prodúcese cando remata a transición.
    - *jump-none:* non hai pausas nos extremos, senón que o primeiro paso comeza coa transición tal e coma o último remata con ela. Hai, polo tanto, unha pausa de 1/n da duración do efecto entre cada par de pasos.
    - *jump-both:* inclúe pausas antes do primeiro paso e despois do último.
  - *step-start:* equivale a **`steps(1, jump-start)`**.
  - *step-end:* equivale a **`steps(1, jump-end)`**.

Podemos aplicar o exemplo CSS seguinte a un documento HTML con dez `div`, cada un con cadanseu identificador, para comprobar o funcionamento das distintas funcións de tempo para animar os efectos de transición:

```
div {
  background-color: #9999ff;
  height: 100px;
  width: 100px;
  margin: 10px;
  display: inline-block;
  font-size: 8pt;
  text-align: center;
}

#caixa1:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: ease;
}

#caixa2:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: linear;
}

#caixa3:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: ease-in;
}

#caixa4:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: ease-out;
}

#caixa5:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: ease-in-out;
}

#caixa6:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: cubic-bezier(0.2,-2,0.8,2);
}

#caixa7:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: steps(5,jump-start);
}

#caixa8:hover {
```

```
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: steps(5,jump-end);
}

#caixa9:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: steps(5,jump-none);
}

#caixa10:hover {
  background-color: #3333cc;
  height: 200px;
  transition-duration: 1s;
  transition-timing-function: steps(5,jump-both);
}
```

Para o traballo con curvas Bezier personalizadas é recomendable a emprega de xeradores de curvas como o que podemos atopar en **Cubic Bezier.com**.

- **transition:** actúa coma *shorthand* do resto, permitindo definir as características das transicións a través dunha única propiedade. Recibe, pois, catro valores que representan ás catro propiedades anteriormente vistas:
    - Primeiro valor: correspondente a **transition-property**.
    - Segundo valor: correspondente a **transition-duration**.
    - Terceiro valor: correspondente a **transition-timing-function**.
    - Cuarto valor: correspondente a **transition-delay**.

  E, ao igual ca nos casos anteriores, admite separar os conxuntos mediante comas para afectar a distintas propiedades.

  Vemos, seguidamente, un exemplo CSS de aplicación de múltiples transicións en *shorthand*, a un mesmo div:

```
div {
  margin-top: 10px;
  margin-left: 100px;
  background-color: #9999ff;
  border: 4px solid black;
  height: 200px;
  width: 200px;
}
```

```
#caixa {
  background-color: #3333cc;
  border-color: red;
  margin-top: 250px;
  margin-left: 400px;
  opacity: 0.7;
  transform: rotate(30deg);
  transition: background-color 2s linear 500ms,
                       border-color 1s ease-in 2.5s,
                       margin-top 3s steps(8, jump-none) 3.5s,
                       transform 1.5s cubic-bezier(0.2,-2,0.8,2) 6.5s,
                       opacity 2s ease-out 8s,
                       margin-left 4s ease-in-out 6s;
}
```

Animacións

Para resolver certas limitacións das transicións, como o feito de só poder afectar a distintas propiedades por nome, pero non por valor (o que impide, por exemplo, actuar sobre múltiples transformacións en base á función empregada), resolveuse incorporar máis un módulo ao estándar, o chamado **CSS Animations**, e con el un novo tipo de propiedades: as **animacións**.

Para crear animacións precisamos dúas compoñentes:

- Unha definición do efecto, realizada en base ás propiedades específicas do módulo e a un conxunto de *keyframes*, é dicir, unha secuencia de puntos que definirá a nosa animación.
- Unha chamada á animación dende o elemento que será obxecto da mesma.

Propiedades

A continuación veremos as propiedades mediante as cales, en combinación coas *keyframes*, podemos definir unha animación vía CSS. No canto de incluír un exemplo específico de cada unha, coma en casos anteriores, introduciremos un exemplo global no remate do apartado, xa que estas propiedades compréndense mellor cando colaboran para xerar un mesmo efecto. As propiedades relativas ás animacións son as seguintes:

- **animation-name:** especifica o nome da animación que usaremos, o cal será empregado para identificala na correspondente secuencia de *keyframes*.
  A continuación un exemplo simplificado da súa sintaxe, representando o uso dunha secuencia de *keyframes* chamada «movemento»:

```
animation-name: movemento;
```

- **animation-duration:** determina a duración (en unidades de tempo) de cada ciclo da animación.
  Velaquí temos un fragmento de código definindo un ciclo de animación cunha duración de 7 segundos:

```
animation-duration: 7s;
```

- **animation-timing-function:** análogo ao que ocorre coa propiedade **transition-timing-function** no contexto das transicións, esta propiedade define a función do tempo que seguirá o efecto de animación. Admite os mesmos análogos que aquela, polo que xa non nos estenderemos aquí en enumeralos.
  De seguido, e coma nos casos anteriores, un exemplo de uso desta propiedade, neste caso, con valor `ease-in`:

```
animation-timing-function: ease-in;
```

- **animation-delay:** sinala o retardo (en unidades de tempo) que terá o arrinque do efecto dende que se produza o seu evento desencadedor.
  Eis, un exemplo do seu uso, para definir un retardo de 800 milisegundos no inicio da animación:

```
animation-delay: 0.8s;
```

- **animation-iteration-count:** define o número de ciclos (repeticións) da animación a realizar. Pode recibir coma valor unha das seguintes cousas:
  - Un número natural sen unidades asociadas.
  - *infinite:* para indicar que a animación repetirá indefinidamente.

  A seguinte liña de código CSS amosa o uso desta propiedade para definir unha animación que repite 5 veces denantes parar:

```
animation-iteration-count: 5;
```

- **animation-direction:** permite establecer o sentido (no tempo) no que se desenvolve a animación. Admite os seguintes valores:
  - *normal:* a animación desenvólvese normalmente, isto é, cara adiante no tempo (é o valor por defecto).
  - *reverse:* a animación desenvólvese «marcha atrás» no tempo.
  - *alternate:* a animación reprodúcese primeiro cara adiante e despois cara atrás.

- *alternate-reverse:* a animación reprodúcese primeiro cara atrás e logo cara adiante.

Aquí temos un exemplo de uso desta propiedade implicando que a animación vai desenvolverse de atrás cara adiante no tempo:

```
animation-direction: reverse;
```

- **animation-fill-mode:** especifica o estilo que aplicará a animación ao elemento obxectivo da mesma denantes e despois de ser executada. Pode tomar os seguintes valores:
  - *none:* non aplica ningún estilo adicional (é o valor por defecto da propiedade), senón que o estilo virá determinado polo resto de regras CSS aplicadas ao elemento.
  - *forwards:* o elemento terá coma estilo o correspondente ao último *keyframe* (o cal vai depender do valor das propiedades **animation-direction** e **animation-iteration-count**[23]).
  - *backwards:* o elemento terá coma estilo o correspondente ao primeiro *keyframe* e reteráo durante toda a duración do retardo (determinado por **animation-delay**). O primeiro *keyframe* vai depender do valor da propiedade **animation-direction**:
    - Se o valor é *normal* ou *alternate*: o estilo corresponderá co establecido no frame *0%* ou en *from*.
    - Se o valor é *reverse* ou *alternate-reverse*: o estilo corresponderá co establecido no frame *100%* ou en *to*.
  - *both:* o elemento terá un estilo correspondente ás características comúns de *forwards* e *backwards*.

Propoñémos un exemplo para ilustrar a emprega desta propiedade, neste caso con valor *backwards*:

```
animation-fill-mode: backwards;
```

- **animation-play-state:** esta propiedade permite indicar o estado actual de reprodución da animación[24]. Admite, ademais dos valores comúns típicos, os dous seguintes:
  - *running:* implica que a animación está activa, é dicir, que vai executarse cando lle corresponda (é o valor por defecto).

[23] Convén mirar o **Titorial oficial de CSS de MDN Web Docs** ao respecto para entender o impacto de ambas propiedades en **animation-fill-mode**.

[24] Esta propiedade foi ideada para ser manexada dende outras linguaxes (principalmente JavaScript) de xeito que a animación poida ser condicionada a certos eventos externos.

- *paused:* a animación está pausada, isto é, non vai executarse.

Aínda que, polo xeral, non imos dar especial uso a esta propiedade (dada a súa particularidade), amosamos igualmente un exemplo de uso da mesma, neste caso indicando que a animación está activa:

```
animation-play-state: running;
```

- **animation:** esta propiedade actúa coma *shorthand* das anteriores. Recibe, polo tanto, oito valores consecutivos (na mesma orde na que foron tratados) separados por espazos en branco:
  - *Nome da animación*.
  - *Duración*.
  - *Función do tempo*.
  - *Retardo*.
  - *Iteracións*.
  - *Sentido no tempo*.
  - *Estilo namentres non executa*.
  - *Estado de reprodución*.

  O exemplo que figura a continuación é equivalente ao establecido nos exemplos propostos para as oito propiedades anteriores:

```
animation: movemento 7s ease-in 0.8s 5 reverse backwards running;
```

Keyframes

A regra *keyframes* de CSS permítenos especificar os pasos que tomará a nosa animación definindo o estilo correspondente a cada un deles. Con este fin empréganse as palabras claves *from* (para representar o estilo do paso inicial) e *to* (para representar o estilo do paso final), as cales poden levar outros pasos intercalados representados pola porcentaxe de animación ocorrida até o inicio dos mesmos.

A regra toma a seguinte sintaxe xeral:

```
@keyframes nome_da_secuencia {
  from {
    estilo1
  }
  porcentaxe1 {
    estilo2
  }
  …
  porcentaxen {
    estilon
  }
  to {
```

```
    estilo(n+1)
  }
}
```

Vexamos, pois, un exemplo dunha animación que muda diversas características dun `div` en 5 pasos:

```
div {
  margin-top: 100px;
  margin-left: 100px;
  background-color: #9999ff;
  border: 4px solid black;
  height: 200px;
  width: 200px;
  animation-name: anima_exemplo;
  animation-duration: 10s;
  animation-timing-function: ease-in;
  animation-delay: 1s;
  animation-iteration-count: infinite;
  animation-direction: alternate-reverse;
  animation-fill-mode: none;
  animation-play-state: running;
}

@keyframes anima_exemplo {
  from {
         background-color: #00ff99;
         height: 250px;
         margin-left: 150px;
  }
  25% {
         background-color: #cc6699;
         width: 250px;
         margin-top: 400px;
  }
  50% {
         background-color: #ffff66;
         height: 400px;
         width: 400px;
         margin-left: 600px;
  }
  75% {
         background-color: #996633;
         width: 100px;
         margin-top: 450px;
  }
  to {
         background-color: #6666ff;
         height: 200px;
         width: 200px;
         margin-top: 100px;
         margin-left: 100px;

  }
}
```

Bibliografía

- Chow, M., (2019). *COMP 20: Web Programming - Document Object Model (DOM)*. [online] Dispoñible en: https://tuftsdev.github.io/WebProgramming/notes/dom.html [Accedido o 20 outubro de 2020].
- *CSS*. (2021). Dispoñible en: https://www.quackit.com/css/ [Accedido o 19 de xaneiro de 2021].
- *CSS Reference*. (2020). Dispoñible en: https://www.w3schools.com/cssref/ [Accedido o 18 de novembro de 2020].
- *CSS Tricks*. (2021). Dispoñible en: https://www.css-tricks.com/ [Accedido o 9 de xaneiro de 2021].
- *Referencia de la API Web*. (2020). Dispoñible en: https://developer.mozilla.org/es/docs/Web/API [Accedido o 18 de novembro de 2020].
- Remoaldo, P. (2011). *CSS3* (1ª ed.). Lisboa: FCA.

ÍNDICE