CSS: Transiciones, filtros y animaciones

Contenido

1. Aplicar filtros a las imágenes con CSS 3
1.1. Múltiples imágenes de fondo 3
1.2. Filtros sobre imágenes 3
1.3. La Etiqueta: <svg> 4
2. Transiciones 8
2.1. Propiedades que pueden ser animadas 9
3. Transformaciones y animaciones 10
3.1. Animaciones 11
3.2. Definiendo la secuencia de la animación con fotogramas 12
3.3. Ejemplos 12
4. Sobre los efectos en una web 17

1. Aplicar filtros a las imágenes con CSS

1.1. Múltiples imágenes de fondo

En CSS3 podremos aplicar varias imágenes de fondo a un mismo elemento. Para ello usaremos la propiedad ***background***, a la cual podremos añadir varias imágenes usando la sintaxis ***url(…)*** y a continuación la posición. Ejemplo:

```
#contenidoFondo {

width: 400px;
height: 200px;
background:
    url(imagen1.gif) bottom right no-repeat,
    url(imagen2.gif) center center no-repeat,
    url(imagen3.gif) repeat,
}
```

Además de la propiedad genérica *background*, podemos usar por separado las distintas propiedades para el fondo:

- background-color: Color de fondo.
- background-image: ruta de la imagen o imágenes que se usarán de fondo.
- background-repeat: repetición de la imagen de fondo (*repeat-x, repeat-y, no-repeat)*.
- background-position: posición a partir de la cual empieza la esquina ruperior izquierda de la imagen de fondo. Valores: [*top, left, bottom, right, center*], *[%x, %y], [Xpx, Ypx], initial, inheritsk*.
- background-attachment: comportamiento de la imagen de fondo cuando se hace scroll por la web. Valores: *scroll|fixed|local|initial|inherit*.

Como en otras ocasiones, podemos especificar varias propiedades de una sola vez:

```
background: #ffffff url("img_tree.png") no-repeat right top;
```

1.2. Filtros sobre imágenes

Podemos aplicar modificaciones sobre las imágenes aplicando filtros con la etiqueta ***filter***, que nos permite aplicar diferentes filtros a las imágenes. Puede aceptar los siguientes valores:

- none: valor por defecto. No se le aplica ningún filtro.
- blur(px): Difumina la imagen. Toma como valor una cantidad de pixels.
- brightness(%): Varía el brillo, toma valores entre 0% - 100%.
- contrast(%): Varía el contraste, toma valores entre 0% - 100%.
- drop-shadow(*h-shadow v-shadow [blur-radius color]*): Aplica efectos de sombra.

```
filter: drop-shadow(8px 8px 10px red);
```

- grayscale(%): Pone la imagen en escala de grises.
- hue-rotate(grados): Aplica una rotación de los tonos o matiz de la imagen el número de grados indicado.

```
filter: hue-rotate(180deg);
```

- invert(%): Invierte la escala de colores.
- opacity(%): Establece el nivel de transparencia. Se puede aplicar directamente con la propiedad ***opacity: %***, sin necesidad de ***filter***.
- saturate(%): Establece el nivel de saturación de la imagen.
- sepia(%): Pone la imagen en sepia.
- url(): toma la ubicación de un archivo XML que especifica un filtro SVG *(1) y puede incluir un ancla a un elemento de filtro específico. Ejemplo:

```
filter: url (svg-url # id-elemento)
```

Por último, están los valores *initial*: pone los valores de filtro a valores por defecto e *inherits*, que hereda esta propiedad de su elemento padre.

1.3. La Etiqueta: <svg>

SVG (Scalable Vector Graphics) es el formato estándar de imagen vectorial para gráficos bidimensionales que soporta interacción y animación con CSS o JavaScript. Entre otras ventajas:

- Es el formato estándar de gráficos vectoriales para web de la W3C.
- Es escalable e independiente de la resolución de pantalla.
- Es soportado por la inmensa mayoría de los navegadores actuales.
- Es básicamente texto, por lo que se puede comprimir consiguiendo ficheros más pequeños que sus equivalentes en imagen (JPG y PNG).
- Se pueden estilar y manipular con CSS y JavaScript.

Existen muchas herramientas tanto para crear como para editar ficheros SVG y después poder aplicarles estilo en CSS. Entre las herramientas actualmente más destacadas encontramos: **Inkscape**, Adobe Illustrator, SVG Editor, …

Hay varias formas de incluir los gráficos SVG en nuestra página web. Si solo se quieren para aprovechar la escalabilidad y el menor tamaño de los archivos se pueden usar en la etiqueta ***<img>*** o mediante CSS con la propiedad ***background-image***. Si se quieren estilar e interactuar con ellos o son archivos SVG complejos, se incluirán con la etiqueta ***<object>*** o ***<svg>*** directamente en el código HTML.

Diferencias de un objeto **svg** dentro de las diferentes etiquetas:

<svg>:

Pros:

- **Control total con CSS/JS:** Es la única forma que te permite estilizar las partes internas del SVG (como el <polygon> o <circle>) con la misma hoja de estilos CSS de tu página.
- **Menos peticiones HTTP:** No se necesita cargar un archivo de imagen por separado, lo que puede hacer que la página cargue marginalmente más rápido.

Contras:

- **Ensucia el HTML:** Tu archivo HTML puede volverse muy largo y difícil de leer si tienes muchos SVG complejos.

- **No se puede cachear:** El SVG no se guarda en la caché del navegador como un archivo de imagen independiente. Si usas la misma estrella en 10 páginas, se cargará 10 veces (como parte del HTML de cada página).

<img>:

Pros:

- **Sencillo y semántico:** Es la forma estándar y más limpia de mostrar una imagen. El atributo alt es genial para la accesibilidad.
- **Se puede cachear:** El navegador guardará mi-logo.svg en caché, así que si se usa en más páginas, cargará instantáneamente.

Contras:

- **Sin control de CSS.** No puedes realizar ningún tipo de modificación, es como si fuera una imagen.

<object>: punto medio entre los dos anteriores:

Pros:

- **Se puede cachear:** Al igual que <img>, el archivo .svg se guarda en caché.
- **Permite interactividad:** A diferencia de <img>, el SVG dentro de un <object> *sí* puede ser manipulado con CSS y JavaScript, pero generalmente con un CSS *interno* del propio archivo .svg o un CSS cargado *desde* el <object>, no desde el CSS principal de la página.
- **Permite "Fallback":** Puedes anidar contenido (como un <img> PNG) dentro del <object> que se mostrará si el navegador no puede cargar el SVG.

Contras:

- **Más complejo:** Es menos directo que <img> y la manipulación con CSS/JS es más complicada que con el método "inline".

La etiqueta <svg>...</svg> actúa como el **lienzo** o la **mesa de dibujo** principal. Todo lo que dibujes (líneas, círculos, texto) debe ir *dentro* de esta etiqueta.

Establece un nuevo **sistema de coordenadas**. A diferencia del flujo normal de HTML (de arriba abajo), dentro de un SVG, todo se posiciona con coordenadas e , donde es la **esquina superior izquierda**.

Atributos Clave de <svg>

- **width y height**: Definen el **tamaño físico** que ocupará el SVG en tu página web (ej. width="100px").
- **viewBox**: Este es **el atributo más importante**. Define el **sistema de coordenadas interno** del lienzo. Se escribe como viewBox="min-x min-y width height".
    - **Ejemplo:** viewBox="0 0 100 100" significa que el lienzo interno va desde la coordenada hasta la (100,100). El navegador escalará este lienzo de 100x100 para que quepa en el width y height físicos que hayas definido.
- **xmlns**: El "XML NameSpace" (xmlns="http://www.w3.org/2000/svg"). Es un atributo estándar y obligatorio que le dice al navegador que interprete el código como SVG.

Formas básicas:

<rect> (Rectángulo):

- x, y: Coordenadas de la esquina superior izquierda.
- width, height: Ancho y alto.
- rx, ry: (Opcional) Radio de las esquinas para hacerlas redondeadas.
- *Ejemplo:* <rect x="10" y="10" width="80" height="80" rx="5" />

<circle> (Círculo):

- cx, cy: Coordenadas del **centro** del círculo.
- r: Radio del círculo.
- *Ejemplol:* <circle cx="50" cy="50" r="40" />

<ellipse> (Elipse):

- cx, cy: Coordenadas del **centro**.
- rx: Radio horizontal.
- ry: Radio vertical.
- *Ejemplo:* <ellipse cx="50" cy="50" rx="40" ry="20" />

<line> (Línea):

- x1, y1: Coordenadas del punto de inicio.
- x2, y2: Coordenadas del punto final.
- *Ejemplo:* <line x1="0" y1="0" x2="100" y2="100" />

<polygon> (Polígono):

- Dibuja una forma de múltiples lados que **se cierra sola**.
- points: Una lista de pares de coordenadas separadas por espacios.
- *Ejemplo:* <polygon points="50,5 95,35 5,35" /> (un triángulo)

<polyline> (Polilínea):

- Igual que <polygon>, pero **no se cierra sola** (es solo un conjunto de líneas conectadas).
- *Ejemplo:* <polyline points="10,10 50,20 90,10" /> (una forma de "V")

Otras etiquetas dentro de <svg>:

<path> (trazado): es el elemento más poderoso. Puede dibujar todo lo que hacen las formas anteriores y mucho más, como curvas complejas, usando el atributo d, con los siguientes comandos:

- M x,y: Mover a (inicio).
- L x,y: **L**ínea hasta.
- H x: Línea **H**orizontal hasta .
- V y: Línea **V**ertical hasta .
- C..., Q...: Comandos para dibujar **C**urvas.
- A...: Dibuja un **A**rco.
- Z: Cerrar el Trazado (volver al punto M inicial).
- *Ejemplo (un corazón simple):* <path d="M50 30 Q 25 10, 10 30 C 0 50, 50 80, 50 80 C 50 80, 100 50, 90 30 Q 75 10, 50 30 Z" />

<g> para agrupar los svg. Es como el para engoblarlos. Se usa para aplicar un estilo o transformación que afectará a todo los svg que incluya.

<text>: Incluir texto dentro del svg, sus atributos son:

- x, y: Coordenadas de anclaje del texto.
- text-anchor="middle": (Opcional) Centra el texto horizontalmente en la coordenada x.
- Ejemplo conceptual: <text x="50" y="50" text-anchor="middle">Hola</text>

<title>: como alt de <img>.

<desc>: para una descripción más larga y detallada.

Atributos (directamente en la etiqueta o en CSS):

- **fill**: El color de **relleno**.
- **stroke**: El color del **borde**.
- **stroke-width**: El **grosor** del borde.
- **fill="none"**: Se usa (como en <polyline>) para indicar que la forma no tiene relleno, solo borde.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Ejemplos de Formas SVG</title>
    <style>
        body { font-family: sans-serif; text-align: center; }
        svg {
            border: 2px solid #1a73e8;
            background-color: #f9f9f9;
            width: 300px; /* Tamaño físico en la página */
            height: 300px;
        }
    </style>
</head>
<body>

    <h1>Ejemplos de Formas Básicas SVG</h1>

    <svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">

        <rect x="5" y="5" width="30" height="15" rx="3" fill="blue" />

        <circle cx="50" cy="15" r="10" fill="red" />

        <line x1="70" y1="10" x2="90" y2="25" stroke="black" stroke-width="3" />

        <ellipse cx="50" cy="40" rx="20" ry="8" fill="green" />

        <polyline points="5,60 15,75 25,60 35,75 45,60"
                  fill="none" stroke="orange" stroke-width="2" />

        <polygon points="60,60 85,60 85,85 72,70 60,85" fill="purple" />

        <text x="50" y="95" text-anchor="middle" font-size="8">
```

```
            ¡Texto en SVG!
        </text>

    </svg>

</body>
</html>
```

Herramientas en línea para crear un filter para cambiar el color a un svg:

- https://codepen.io/sosuke/pen/Pjoqqp
- https://imagecolorpicker.com/es

2. Transiciones

CSS3 trae efectos espectaculares. Como los **efectos con CSS3 para imágenes** que hemos visto.

A la hora de aplicar cambios en el aspecto de un componente HTML, para que el cambio no sea brusco, conviene hacer uso de las transiciones. La velocidad de transición también se la debemos a CSS3, y se obtiene mediante la propiedad ***transition***. Los tipos de movimiento de las transiciones pueden ser los siguientes:

- **linear: la velocidad de la animación es uniforme en todo el recorrido.**
- **ease: la velocidad se acelera al inicio, luego se retarda un poco y se acelera al final de nuevo.**
- **ease-in: la animación empieza lentamente y va aumentando progresivamente.**
- **ease-out: la animación empieza muy rápida y va descendiendo progresivamente.**
- **ease-in-out: la animación empieza y acaba lentamente, y es en la parte central del recorrido donde la velocidad es más rápida.**
- **cubic-bezier(n,n,n,n): establece una curva de bezier que podemos configurar.**
- **…**

Las transiciones CSS te permiten cambiar los valores de las propiedades durante una duración determinada. Usaremos la propiedad ***transition***. Habrá que especificar dos valores: los cambios a realizar y la duración de la transición.

```
transition: width 2s, height 4s;
```

Podemos poner todos los valores separados por espacios en blanco en la propiedad ***transition***:

`transition:[propiedad a modificar] [Duración] [Tipo de animación] [Retardo];`

```
transition: all .5s ease-in-out;
```

o poner los valores de la transición en propiedades diferentes:

- **transition-property:** especifica el nombre o nombres de las propiedades CSS a las que deberían aplicarse las transiciones. Sólo las propiedades que se enumeran **aquí** y también **aquí** son animadas durante las transiciones; los cambios en el resto de las propiedades suceden de manera instantánea como siempre. Permite la opción ***all*** para que cambien todas las propiedades.

- **transition-duration:** especifica la duración en la que sucederán las transiciones. Puedes especificar una única duración que se aplique a todas las propiedades durante la transición o valores múltiples que permitan a cada propiedad de transición un período de tiempo diferente.
- **transition-timing-function:** especifica cómo se realiza la transición. Podemos usar la curva cúbica Bézier, que se usa para definir cómo se computan los valores intermedios para las propiedades.
- **transition-delay:** define el tiempo de espera entre el momento en que se cambia una propiedad y el inicio de la transición.

```
div {
    transition-property: opacity, left, top, height;
    transition-duration: 3s, 5s, 3s, 5s;
}
```

El efecto de la transición empezará cuando los valores de las propiedades CSS que hemos especificado cambien de valor.

2.1. Propiedades que pueden ser animadas

background-color	flood-color	-moz-outline-radius
background-image	font-size	outline-width
background-position	font-size-adjust	padding
background-size	font-stretch	right
border-color	font-weight	stop-color
border-radius	height	stop-opacity
border-width	-moz-image-region	stroke
border-spacing	left	stroke-dasharray
bottom	letter-spacing	stroke-dashoffset
-moz-box-flex	lighting-color	stroke-miterlimit
box-shadow	line-height	stroke-opacity
color	margin	stroke-width
-moz-column-count	marker-offset	text-indent
-moz-column-gap	max-height	text-shadow
-moz-column-rule-color	max-width	top
-moz-column-rule-width	min-height	-moz-transform-origin
-moz-column-width	min-width	-moz-transform
clip	opacity	vertical-align
fill	outline-color	visibility
fill-opacity	outline-offset	width

word-spacing	z-index

3. Transformaciones y animaciones

CSS3 provee también de una serie de efectos que imprimen movimiento 2D o 3D, transformando y animando los componentes. Combinándolas, además, se pueden conseguir efectos espectaculares. Algunos de ellos implican usar un contenedor, un div que contiene al componente a transformar o animar. Algunas propiedades que podremos cambiar es la posición, el tamaño y la inclinación, además, podremos realizar transformaciones 2D y 3D.

Para empezar, pondremos los valores posibles para el atributo ***transform***:

- **Trasladar un elemento:** translate(x, y). Mueve un elemento a la posición especificada.
- **Zoom y cambios de tamaño:** scale(). Si usamos un valor para *scale* menor que 1, el componente se contrae. Se usa sobre todo en imágenes.

```
transform:scale(1.3);
```

- **Giros y rotaciones:** rotate(X deg).Gira los grados especificados en sentido de la aguja del reloj o al contrario si el valor en grados es negativo.
- **Inclinaciones:** skew(X deg). Inclina el elemento el valor dado en grados.
- **Varias transformaciones a la vez:** matrix(). Combina todas las transformaciones. La sintaxis es:

```
matrix(scaleX(), skewY(), skewX(), scaleY(), translateX(), translateY())
```

Para cada transformación de las vistas anteriormente, existe la misma solamente para el eje X y el eje Y. Por ejemplo: translateX(), translateY(), scaleX(), scaleY(), skewX(), skewY().

Otra forma de combinar varias transformaciones en un solo elemento es separando con espacios cada una de esas transformaciones:

```
transform: translateX(200%) scale(2,2);
```

Transformaciones 3D: Existen transformaciones muy parecidas, pero incluyendo el eje Z. Para poder realizar transformaciones en el eje Z, tendremos que usar la propiedad ***perspective*** en el elemento padre de los elementos a transformar y ***transform-style: preserve-3d:***

```
perspective: 30em;
transform-style: preserve-3d;
```

Otros valores para transform:

Valor	Descripción

none	Sin transformaciones
matrix(*n,n,n,n,n,n*)	Define una transformación 2D, usando una matriz de 6 valores.
matrix3d (*n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n*)	Define una transformación 3D, usando una matriz de 4x4, 16 valores.
translate(*x,y*)	Define una transformación 2D.
translate3d(*x,y,z*)	Define una transformación 3D.
translateX(*x*)	Define una traslación en el eje X.
translateY(*y*)	Define una traslación en el eje Y.
translateZ(*z*)	Define una traslación en el eje Z.
scale(*x,y*)	Transformación de escalado en 2D
scale3d(*x,y,z*)	Transformación de escalado en 3D
scaleX(*x*)	Escala solamente el eje X.
scaleY(*y*)	Escala solamente el eje Y.
scaleZ(*z*)	Transformación 3D usando solo el eje Z.
rotate(*angle*)	Rotación 2D especificando el ángulo.
rotate3d(*x,y,z,angle*)	Define una rotación 3D.
rotateX(*angle*)	Rotación 3D sobre el eje X.
rotateY(*angle*)	Rotación 3D sobre el eje Y.
rotateZ(*angle*)	Rotación 3D sobre el eje Z.
skew(*x-angle,y-angle*)	Tuerce el componente sobre el eje X e Y.
skewX(*angle*)	Tuerce el componente sobre el eje X.
skewY(*angle*)	Tuerce el componente sobre el eje Y.
perspective(*n*)	Vista en perspectiva del componente.
initial	Establece la propiedad a su valor por defecto.
inherit	Hereda esta propiedad de su elemento padre

3.1. Animaciones

Para crear una secuencia de animación CSS, usaremos la propiedad ***animation*** y sus sub-propiedades. Con ellas podemos no solo configurar el ritmo y la duración de la animación sino otros detalles sobre la secuencia de la animación. Con ellas **no** configuramos la apariencia actual de la animación, para ello disponemos de ***@keyframes*** como describiremos más adelante.

Las sub-propiedades de ***animation*** son:

`animation-delay:` Tiempo de retardo entre el momento en que el elemento se carga y el comienzo de la secuencia de la animación. Ejemplo: `animation-delay: 2s;`.

`animation-direction:` Indica si la animación debe retroceder hasta el fotograma de inicio al finalizar la secuencia o si debe comenzar desde el principio al llegar al final. Valores: ***normal, reverse, alternate, alternate-reverse***.

`animation-duration:` Indica la cantidad de tiempo que la animación consume en completar su ciclo (duración). Ejemplo: `animation-duration: 2s;`. Si no ponemos duración, la animación no se ejecutará, ya que tiene como valor por defecto 0.

`animation-iteration-count:` El número de veces que se repite. Podemos indicar ***`infinite`*** para repetir la animación indefinidamente.

`animation-name:` Especifica el nombre de la regla ***`@keyframes`*** que describe los fotogramas de la animación.

`animation-play-state:` Permite pausar y reanudar la secuencia de la animación. Valores: *paused, running, initial, inherit*.

`animation-timing-function:` Indica el ritmo de la animación, es decir, como se muestran los fotogramas de la animación, estableciendo curvas de aceleración. Valores: *linear, ease, ease-in, ease-out, ease-in-out, cubic-bezier(n,n,n,n).*

`animation-fill-mode:` Especifica qué valores tendrán las propiedades después de finalizar la animación (los de antes de ejecutarla, los del último fotograma de la animación o ambos). Valores: *none, forwards, backwards, both*.

3.2. Definiendo la secuencia de la animación con fotogramas

Una vez configurado el tiempo de la animación, necesitamos definir su apariencia. Esto lo haremos estableciendo, como mínimo, dos fotogramas usando la regla ***`@keyframes`***. Cada fotograma describe cómo se muestra cada elemento animado en un momento dado durante la secuencia de la animación. Desde que se define el tiempo y el ritmo de la animación, el fotograma usa ***`percentage`*** para indicar en qué momento de la secuencia de la animación tiene lugar. 0% es el principio, 100% es el estado final de la animación. Debemos especificar estos dos momentos para que el navegador sepa dónde debe comenzar y finalizar; debido a su importancia, estos dos momentos tienen alias especiales: ***`from`*** y ***`to`***.
Además, puedes, opcionalmente, incluir fotogramas que describan pasos intermedios entre el punto inicial y final de la animación.

3.3. Ejemplos

Haciendo que un texto se deslice por la ventana del navegador

Este sencillo ejemplo da estilos al elemento `<p>` para que el texto se deslice por la pantalla entrando desde el borde derecho de la ventana del navegador.
Animaciones como esta pueden hacer que la página se vuelva más ancha que la ventana del navegador. Para evitar este problema, pon el elemento que será animado en un contenedor y agrega ***`overflow:hidden`*** en el contenedor.

```
p {
```

```
    animation-duration: 3s;
    animation-name: slidein;
}

@keyframes slidein {
    from {
      margin-left: 100%;
      width: 300%
    }

    to {
      margin-left: 0%;
      width: 100%;
    }
}
```

El estilo del elemento **<p>** especifica, a través de la propiedad **animation-duration**, que la animación debe durar 3 segundos desde el inicio al fin y que el nombre de los **@keyframes** que definen los fotogramas de la secuencia de la animación es "*slidein*".
Los fotogramas se definen usando la regla **@keyframes**. En nuestro ejemplo, tenemos solo dos fotogramas. El primero de ellos sucede en el 0% (hemos usado su alias **from**). Aquí, configuramos el margen izquierdo del elemento, poniéndolo al 100% (es decir, en el borde derecho del elemento contenedor), y su ancho al 300% (o tres veces el ancho del elemento contenedor). Esto hace que en el primer fotograma de la animación tengamos el texto fuera del borde derecho de la ventana del navegador.
El segundo (y último) fotograma sucede en el 100% (hemos usado su alias **to**). Hemos puesto el margen derecho al 0% y el ancho del elemento al 100%. Esto produce que el texto, al finalizar la animación, esté en el borde derecho del área de contenido.

Añadiendo otro fotograma

Vamos a añadir otro fotograma a la animación de nuestro ejemplo anterior. Pongamos que queremos que el tamaño de fuente del texto aumente a medida que se mueve durante un tiempo y que después disminuye hasta su tamaño original. Esto es tan sencillo como añadir este fotograma:

```
    75% {
        font-size: 300%;
        margin-left: 25%;
        width: 150%;
    }
```

Esto le dice al navegador que en el 75% de la secuencia de la animación, el texto tiene un margen izquierdo del 25%, un tamaño de letra del 300% y un ancho del 150%.

Haciendo que se repita

Para hacer que la animación se repita, solo hay que usar la propiedad **animation-iteration-count** e indicarle cuántas veces debe repetirse. En nuestro caso, usamos ***infinite*** para que la animación se repita indefinidamente:

```
p {
```

```
    animation-duration: 3s;
    animation-name: slidein;
    animation-iteration-count: infinite;
}
```

Moviéndolo hacia adelante y hacia atrás

Hemos hecho que se repita, pero queda un poco raro que salte al inicio de la animación cada vez que ésta comienza. Queremos que se mueva hacia adelante y hacia atrás en la pantalla. Esto lo conseguimos fácilmente indicando que **`animation-direction`** es *`alternate`*:

```
p {
    animation-duration: 3s;
    animation-name: slidein;
    animation-iteration-count: infinite;
    animation-direction: alternate;
  }
```

Usando la versión abreviada animation

La versión abreviada **`animation`** es usado para ahorrar espacio. Por ejemplo, la regla que hemos usado antes se puede reemplazar por:

```
p {
    animation: 3s infinite alternate slidein;
}
```

Estableciendo múltiples valores de propiedades animation

Las propiedades de la versión larga de **`animation`** pueden aceptar múltiples valores, separados por comas - esta característica puede ser usada cuando quieres aplicar múltiples animaciones en una solo regla, y establecer por separado **duration**, **iteration-count**, etc. para diferentes animaciones.

Vamos a ver algunos ejemplos rápidos para explicar las diferentes combinaciones:

En el primer ejemplo, tenemos tres nombres de animación establecidos, pero solo una duración (**duration**) y número de iteraciones (**iteration-count**). En este caso, a las tres animaciones se les da la misma duración y número de iteraciones:

```
p {
    animation-name: fadeInOut, moveLeft300px, bounce;
    animation-duration: 3s;
    animation-iteration-count: 1;
}
```

En el segundo ejemplo, tenemos tres valores establecidos en las tres propiedades. En este caso, cada animación se ejecuta con los valores correspondientes en la misma posición en cada propiedad, así por ejemplo `fadeInOut` tiene una duración de 2.5s y 2 iteraciones, etc.

```
p {
    animation-name: fadeInOut, moveLeft300px, bounce;
```

```
    animation-duration: 2.5s, 5s, 1s;
    animation-iteration-count: 2, 1, 5;
}
```

En el tercer caso, hay tres animaciones especificadas, pero solo dos duraciones y número de iteraciones. En los casos en donde no hay valores suficientes para dar un valor separado a cada animación, los valores se repiten de inicio a fin. Así, por ejemplo, `fadeInOut` obtiene una duración de 2.5s y `moveLeft300px` obtiene una duración de 5s. Ahora tenemos asignados todos los valores de duración disponibles, así que empezamos desde el inicio de nuevo - por lo tanto `bounce` tiene una duración de 2.5s. El número de iteraciones (y cualquier otra propiedad que especifiques) será asignados de la misma forma.

```
p {
    animation-name: fadeInOut, moveLeft300px, bounce;
    animation-duration: 2.5s, 5s;
    animation-iteration-count: 2, 1;
}
```

Usando eventos de animación

Podemos tener un control mayor sobre las animaciones (así como información útil sobre ellas) haciendo uso de eventos de animación. Dichos eventos, representados por el objeto ***AnimationEvent***, se pueden usar para detectar cuándo comienza la animación, cuándo termina y cuándo comienza una iteración. Cada evento incluye el momento en el que ocurrió, así como el nombre de la animación que lo desencadenó.

Vamos a modificar el ejemplo del texto deslizante para recoger información sobre cada evento cuando suceda y así podremos echar un vistazo a cómo funcionan.

3.3.1. Agregando CSS

Empezamos creando el CSS para la animación. Esta animación durará 3 segundos, se llama slidein, se repite 3 veces, y alterna de dirección cada vez. En **@keyframes**, width y margin-left son manipulados para hacer que el elemento se deslice por la pantalla.

```
.slidein {
    animation-duration: 3s;
    animation-name: slidein;
    animation-iteration-count: 3;
    animation-direction: alternate;
}

@keyframes slidein {
    from {
      margin-left:100%;
      width:300%
    }

    to {
     margin-left:0%;
     width:100%;
   }
```

```
}
```

3.3.2. Añadiendo detectores de eventos a la animación

Usaremos un poco de Javascript para escuchar los tres posibles eventos de animación. Este código configura nuestros detectores de eventos (event listeners); los llamamos cuando el documento carga por primera vez para configurar todo.

```
var e = document.getElementById("watchme");

e.addEventListener("animationstart", listener, false);
e.addEventListener("animationend", listener, false);
e.addEventListener("animationiteration", listener, false);

e.className = "slidein";
```

3.3.3. Recibiendo los eventos

Los eventos, al irse disparando, llamarán a la función `listener()`.

```
function listener(e) {
    var l = document.createElement("li");
    switch(e.type) {
      case "animationstart":
        l.innerHTML = "Iniciado: tiempo transcurrido " + e.elapsedTime;
        break;
      case "animationend":
        l.innerHTML = "Finalizado: tiempo transcurrido " + e.elapsedTime;
        break;
      case "animationiteration":
        l.innerHTML = "Nueva iteración comenzó a los " + e.elapsedTime;
        break;
    }
    document.getElementById("output").appendChild(l);
  }
```

Este código también es muy sencillo. Miramos en **`e.type`** para saber qué tipo de evento se ha disparado y, en función del tipo de evento, añadimos su correspondiente texto al elemento **`<ul>`** que usaremos para registrar la actividad de nuestros eventos.

El resultado, si todo ha ido bien, será algo parecido a esto:

- Iniciado: tiempo transcurrido 0
- Nueva iteración comenzó a los 3.01200008392334
- Nueva iteración comenzó a los 6.00600004196167
- Finalizado: tiempo transcurrido 9.234000205993652

Fijémonos en que después de la iteración final de la animación, el evento **`animationiteration`** no se envía, en su lugar se lanza **`animationend`**.

4. Sobre los efectos en una web

A veces uno quiere que su web sea «la más chula del barrio», pero ten siempre presente cuál es el objetivo primordial de tu web, y no lo sacrifiques ni un ápice ante el diseño, y mucho menos ante un efecto visual.

Ten en cuenta que en un ecommerce el objetivo número uno es vender, y en una web corporativa o una de marca personal es generar contactos comerciales. Así que eso es lo primero y toda la web debe estar pensada (en conjunto y también cada uno de los elementos de forma individual) para servir a ese objetivo.

Después de eso está la usabilidad, o más bien como parte de eso, ya que una web con usabilidad pobre no va a vender ni a convertir. Sólo una vez que ya se han tenido en cuenta esos aspectos queda lugar para florituras de diseño y efectos llamativos, si estamos seguros que no van a interferir.

Ten eso en cuenta a la hora de implementar estos efectos de imágenes.