



TEMA 5. CSS Avanzado

Lenguajes de Marcas
CFGS DAW 1

Autor: Pascual Ligeró

Revisado por:

Diana Bautista - d.bautistarayo@edu.gva.es

Óscar Villar - or.villarfernandez@edu.gva.es

2022/2023

Versión:260123.1102

Licencia



**[CC](#) [BY-NC-SA](#) [3.0](#) [ES](#) Reconocimiento – NoComercial –
CompartirIgual (by-nc-sa):**

No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

NOTA: Esta es una obra derivada de la original realizada por Pascual Ligeró.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención



Interesante

ÍNDICE DE CONTENIDO

1. Selectores avanzados	4
2. Transform	5
3. Position	6
3.1 Position relative	6
3.2 Position absolute	7
3.3 Position fixed	7
3.4 Position sticky	8
3.5 Z-index	8
4. CSS Flexbox	9
4.1 Declarando Flex-box	9
4.1 Distribución en columna	10
4.2 Disposición bidimensional	11
4.3 Alineamiento horizontal	11
4.4 Alineamiento vertical	12
4.5 Gestión de líneas de elementos	12
4.6 Control de crecimiento de elementos	13
4.7 Orden de elementos	13
5. CSS Grid	14
5.1 Definiendo Grid	14
5.2 Definiendo plantillas de filas y columnas	15
5.2 Definiendo áreas grid	16
6. Fondos	17
7. Animaciones	20
7.1 Concepto de transición	20
7.2 Concepto de animación	21
7.3 @keyframes	21
8. Iniciación al pensamiento RWD	23
10. BIBLIOGRAFÍA	24

UD05. CSS Avanzado

1. Selectores avanzados

Seguimos presentando aquellos selectores que no fueron incluidos en la anterior unidad:

Selector hijos directos (`div > p{ ... }`):

Este selector permite apuntar a todos y cada uno de los elementos declarados a la derecha del símbolo mayor que **estrictamente sean hijos directos de cajas div**. Únicamente aquellos elementos internos que nuevamente adquieran el mismo tipo de relación padre-hijo, y en concreto los hijos <p>, se verán de nuevo afectados por las reglas CSS aplicadas en este selector.

Selector descendientes (`div p{ ... }`):

Este selector permite ahora sí, apuntar todos los elementos hijo <p>, sea cual sea su nivel de profundidad para cuando estos tengan un elemento ancestro <div>. Este selector y el anterior con comúnmente confundidos por lo que es recomendable realizar pruebas

Selector primer contiguo (`div + p { ... }`):

Permite seleccionar un elemento adyacente a otro (declarado a continuación según el flujo HTML). Tiene su utilidad con: subtítulos, pies de tabla / imagen, etc.

Selector de atributo (`[target="_blank"]`):

Permite seleccionar aquellos elementos cuyos atributos coincidan con un patrón textual. Existen variantes de este selector de atributo. Consultar en W3Schools si se requiere uno más preciso.

Selectores de estado o pseudo clases (`div:active{ ... }`):

Determinan qué propiedades CSS cambiar en función de haberse alcanzado un determinado estado como: pasar el mouse por encima, hacer click, cambiar de estado a visitado (enlace), alcanzar un estado válido / inválido, enésimo elemento de una sucesión (lista, fila de tabla) etc.

Selector raíz (`:root{ ... } / html{ ... }`):

Generalmente se recurre a este selector con objeto de definir propiedades que sean heredadas para todos y cada uno de los elementos declarados en el documento. Algunos objetivos comunes de recurrir a este selector son: reseteo de propiedades por defecto, custom properties o variar el tamaño de fuente y aplicar medidas con las unidades em y rem a los elementos.

Selector combinado (`.clase1.clase2 { .. }`):

Selector que actúa como un Y lógico y que exige que los elementos tengan dos o más propiedades. Para este ejemplo: se requiere que el elemento defina la clases clase1 y también la clase2.

La totalidad de selectores presentada en la presente unidad y la anterior dan un gran nivel de personalización a la hora de poder definir agrupaciones de reglas CSS para conjuntos de elementos. Hay más selectores, aunque se han presentado los más relevantes.

2. Transform

Podemos efectuar una serie de transformaciones en los elementos basadas en:

- Translaciones
- Rotaciones
- Escalados
- Aplicar un factor de oblicuidad

Para la caja del ejemplo gráfico se ha aplicado:

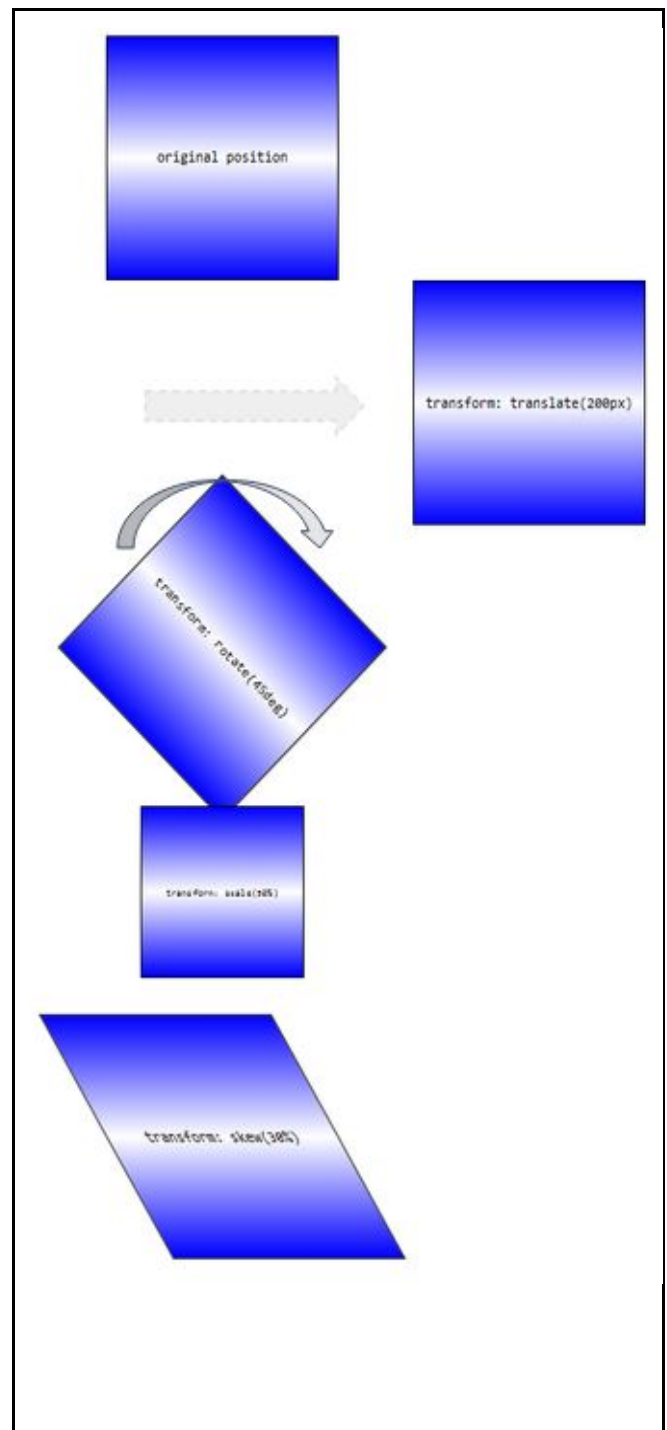
```
transform: translate(200px);  
transform: rotate(45deg);  
transform: scale(50%);  
transform: skew(30%);
```

Las variantes 3D permiten efectos más elaborados.

Estas propiedades ganan al ser animadas debidamente con @keyframes.

Puede usarse `translate` para corregir el posicionamiento puntual de un elemento.

De hecho, `translate` acepta dos argumentos también (desplazamiento en los ejes x e y). Como veremos en el apartado de posicionamiento, equivaldrá a aplicar `position: relative` y hacer variar las propiedades offset.



3. Position

Tras haber estudiado float con bastante calma en la unidad 4, os presentamos position como herramienta de posicionamiento complementaria (que no sustitutoria).

Se dice que un elemento está posicionado si es marcado con una de estas propiedades:

- `position: relative;`
- `position: absolute;`
- `position: fixed;`
- `position: sticky;`

Por defecto un elemento posee la propiedad `position: static` y este por tanto, no se considera posicionado.

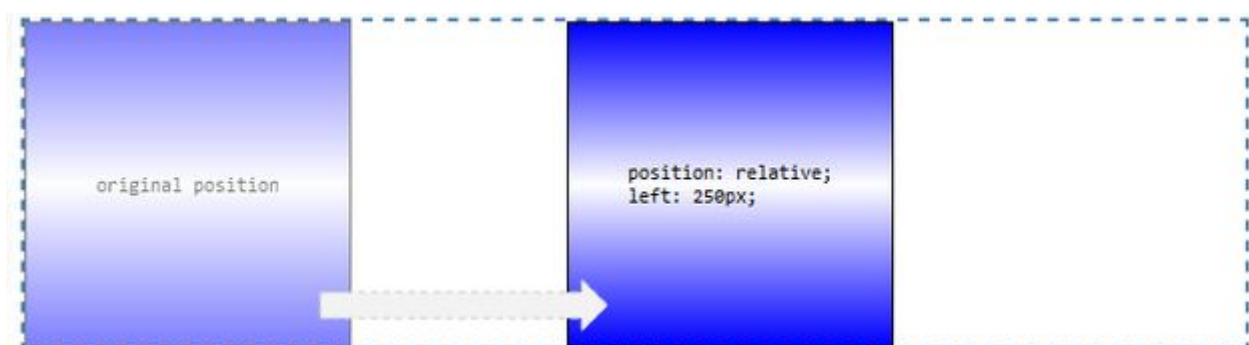
Una vez el elemento está posicionado (tiene uno de los 4 valores: relative, absolute, fixed o sticky) podemos aplicar los valores “offset” o de desplazamiento y un desplazamiento en una unidad de medida de las que pone a disposición CSS. Los valores “offset” son:

- top
- bottom
- right
- left

Solo cabe tener que mencionar que para cada uno de los valores de posicionado existe una referencia distinta y que el propósito de cada uno de ellos es diferente.

3.1 Position relative

Utilizamos **position relative** cuando queremos **variar un elemento con respecto a su posición original** (esta será su referencia). Cuando aplicamos position relative, nada ocurre. Sin embargo, al aplicar valores offset, observaremos cómo este varía con respecto a su posición original.



3.2 Position absolute

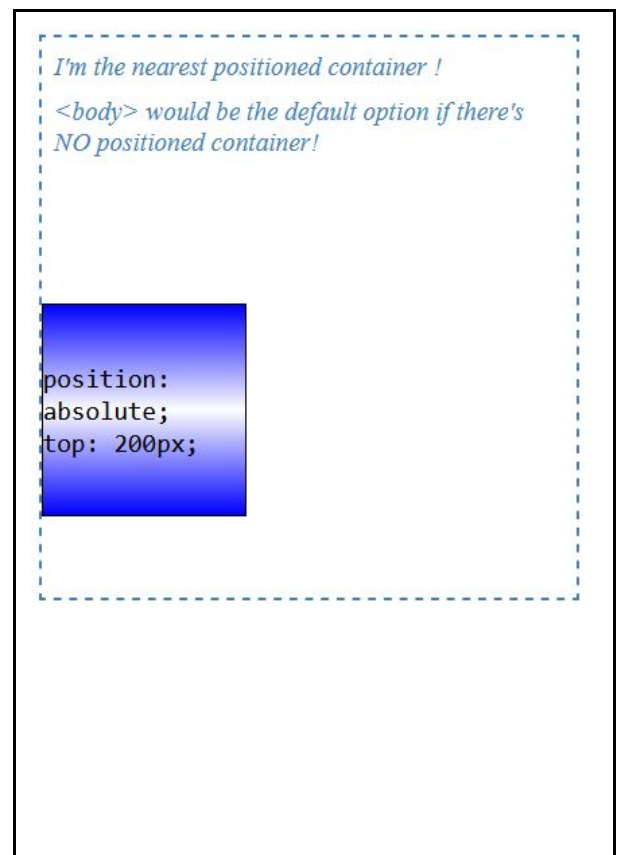
Position absolute requiere tener clara la relación entre el elemento que queremos posicionar absolutamente con su contenedor padre.

Concretamente, necesitamos que el **contenedor padre esté posicionado** con el valor: relative, absolute, fixed o sticky.

Finalmente, el elemento en cuestión debe implementar position: absolute y definir los valores offset oportunos.

Si el contenedor padre no está posicionado, se tomará en cuenta como referencia el body. Por tanto la referencia en este caso es: el contenedor posicionado más próximo. Cabe resaltar que el elemento a posicionar comenzará a dibujarse en la esquina superior izquierda del contenedor.

Otro detalle importante es que el elemento posicionado con position: absolute **pierde su espacio en el flujo** (y otros lo intentarán ocupar).

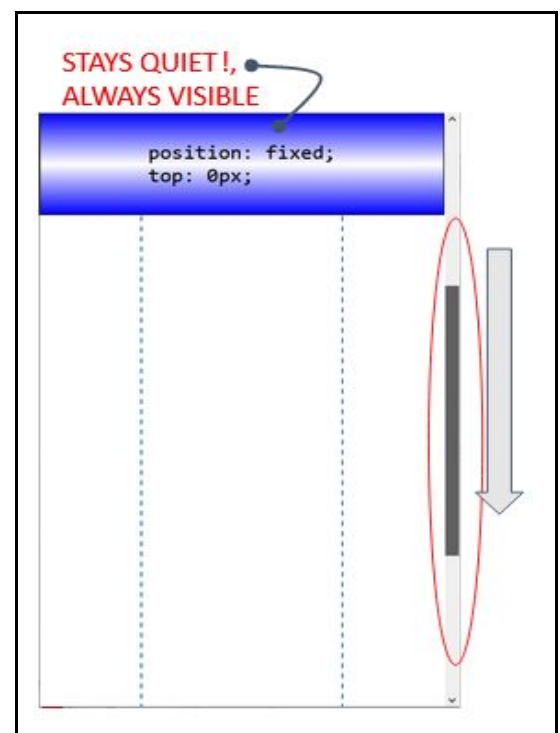


3.3 Position fixed

Position fixed permite que un elemento permanezca fijo en el visor del navegador. Puede interesar tener una barra de navegación perpetua o un pie de página o zona que sea siempre visible, independientemente del nivel de scroll que el usuario efectúe.

El elemento que se desea fijar su posición pierde también su espacio en el flujo HTML (los elementos inmediatamente declarados ocuparán su lugar).

En el caso de position: fixed, la referencia de partida es el viewport o visor del navegador. Apelar a los valores offset: top: 0 y left: 0 equivale posicionarlo a la esquina superior izquierda del visor.



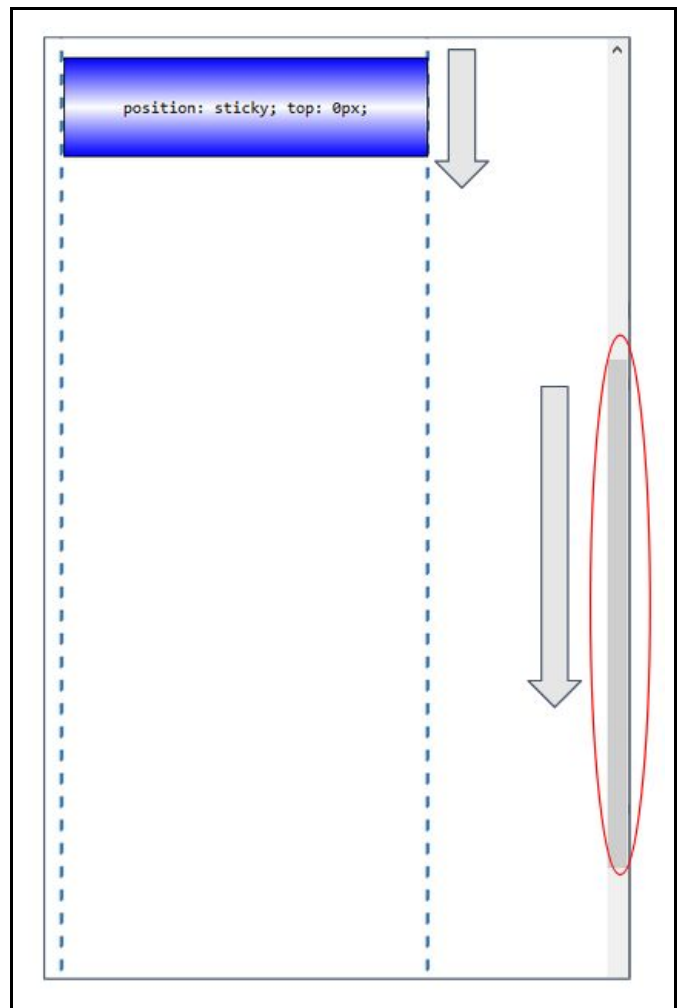
3.4 Position sticky

¿Recuerdas esos banners publicitarios que te persiguen por toda una página web aún haciendo scroll? Este efecto se puede conseguir sin Javascript y con relativa sencillez con la propiedad `position: sticky`.

Una vez declarado este posicionamiento junto a un valor offset, como por ejemplo una separación del elemento con respecto al borde superior del contenedor referencia (el viewport), se escuchará el evento de scroll y persuadirá al usuario en pantalla.

Generalmente este elemento sticky se incluye dentro de un contenedor con un alto finito que hará las veces de “carril” para el recorrido de persuasión que efectuará.

Es conveniente revisar la compatibilidad existente de esta propiedad en la plataforma Can I Use, dado que ha sido de las últimas propiedades position incluidas en CSS y para algún caso particular puede no tener soporte.



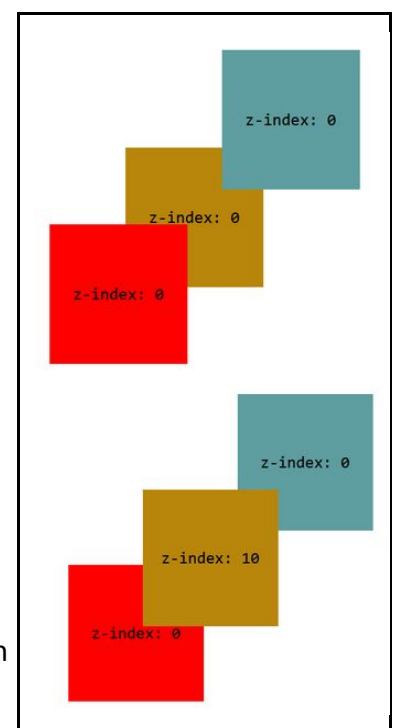
3.5 Z-index

Es importante darse cuenta que en cualquier momento dos elementos cualesquiera pudieran estar solapándose, bien por aplicar translate, alterar su posición con la propiedad position, etc.

En primera instancia CSS resuelve estos solapamientos, dejando que aquellos elementos declarados a continuación de otros en el flujo HTML se muestren al frente. Ésta es la primera consideración que hemos de tomar en cuenta.

Para manipular el apilamiento de capas de elementos se recurre a la propiedad z-index. Z-index permitirá atribuir un valor numérico a cada elemento a fin de poder mostrarlo más al frente u ocultarlo en capas posteriores. El valor z-index por defecto de todos los elementos es 0.

Se recomienda usar valores z-index en rangos de decenas (o centenas) y no declarar valores contiguos como 1, 2, 3.. De este modo se dispondrá de un margen de personalización cuando se desee resolver una situación particular donde varios elementos intervengan.



4. CSS Flexbox

Vamos con una tercera herramienta para posicionar elementos, ahora sí con algo más de autonomía que float o position.

CSS Flexbox llega para poder alinear o reordenar elementos generalmente en un eje (horizontal o vertical) en un área determinada.



Lejos de lo que muchos puedan creer, flexbox también tiene sus limitaciones y/o dificultades a la hora de realizar ciertas operaciones. La disposición bidimensional de elementos no es su fuerte, como ya se ha avanzado, para eso existe una última herramienta que será presentada en un apartado posterior.

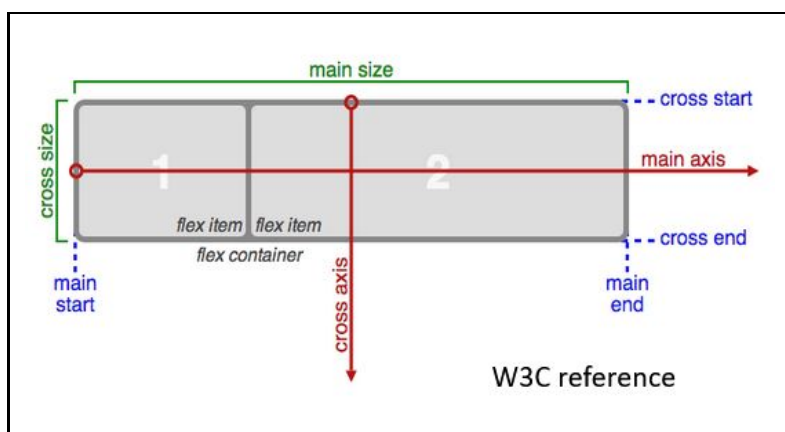
4.1 Declarando Flex-box

Sobre un contenedor cualquiera se define la propiedad **display: flex** y automáticamente sus elementos se tornan flexibles, es decir, de encogerse y agrandarse según un criterio que estudiaremos a continuación. También es posible definir **display: inline-flex**, de este modo, la caja actuará como si de un elemento de línea se tratara.

Sobre un contenedor flexbox actúan siempre dos ejes: el principal (main axis) y el secundario (cross-axis).

El eje principal, si no variamos la propiedad **flex-direction**, por defecto es la horizontal.

El comportamiento de los elementos contenidos por tanto es alinearse en el



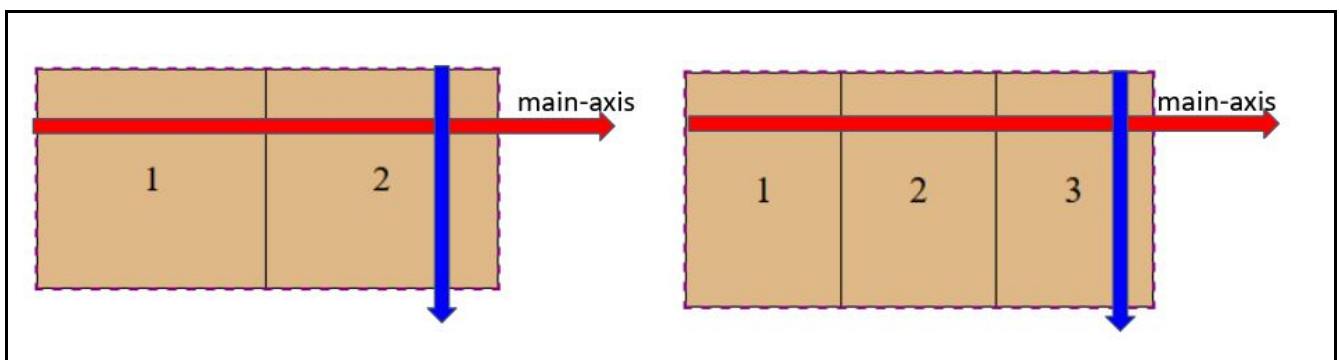
eje horizontal unos al lado de otros de forma adyacente, pese a que estos sean elementos de bloque.

El eje secundario (por defecto) se encarga de controlar el posicionamiento de elementos en la vertical.

Cada uno de los ejes tiene una zona de comienzo y otra de término, en la jerga flex-box es: start y end.

La adición de más elementos al contenedor flexbox puede originar dos comportamientos:

- **Tratar de comprimirse** en el espacio interno del contenedor (aún teniendo estos unas dimensiones definidas). Es sin duda el comportamiento más habitual y característico.
- **Desbordamiento** si entre los elementos se produce un empuje excesivo debido a margins/paddings o debido al tamaño del contenido de estos.



Contenedor CSS Flexbox al que se le añade un elemento más. Compresión de elementos.

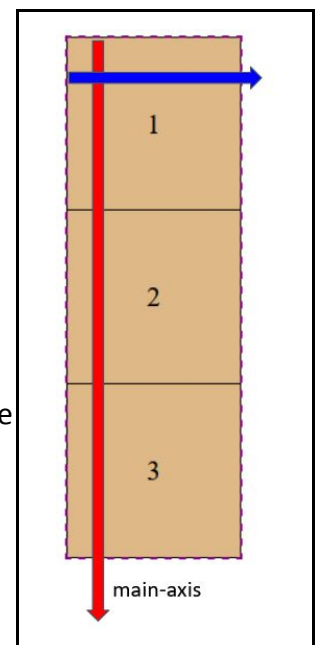
4.1 Distribución en columna

La disposición de elementos en columna se consigue con la propiedad **flex-direction** y el valor **column** (por defecto el valor es `row`).

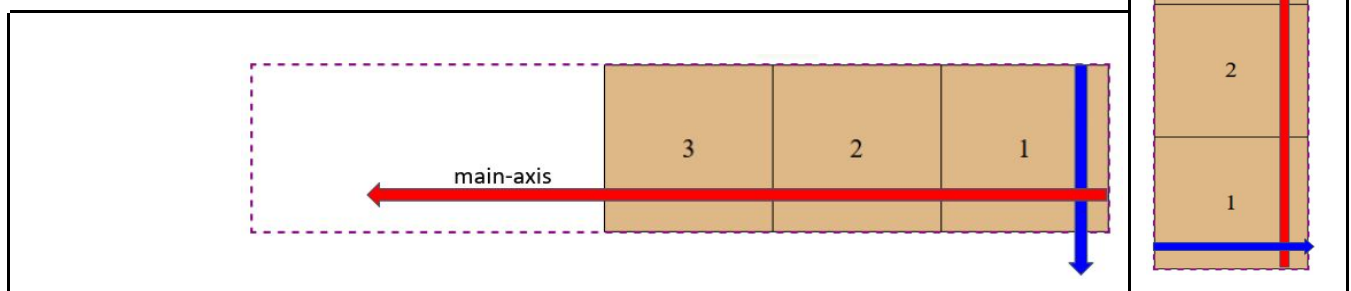
Nótese que ahora el eje principal es el vertical y el secundario el horizontal. Es importante controlar este hecho, dado que a posteriori haremos uso de propiedades para alinear elementos cuyo fundamento será el eje principal y secundario.

Esta distribución altera la naturaleza de elementos de línea o línea-bloque, que como ya sabemos se posicionan de forma contigua en la misma línea hasta que no caben en la misma.

Los elementos por defecto intentarán acomodarse dentro del contenedor.



Tanto si estamos con una configuración de alinear elementos en filas (`row`) o columnas (`column`) se admiten los valores `row-reverse` y `column-reverse` a fin de poder, además de invertir los ejes, variar el punto de comienzo de renderizado de estos (`start` y `end`).

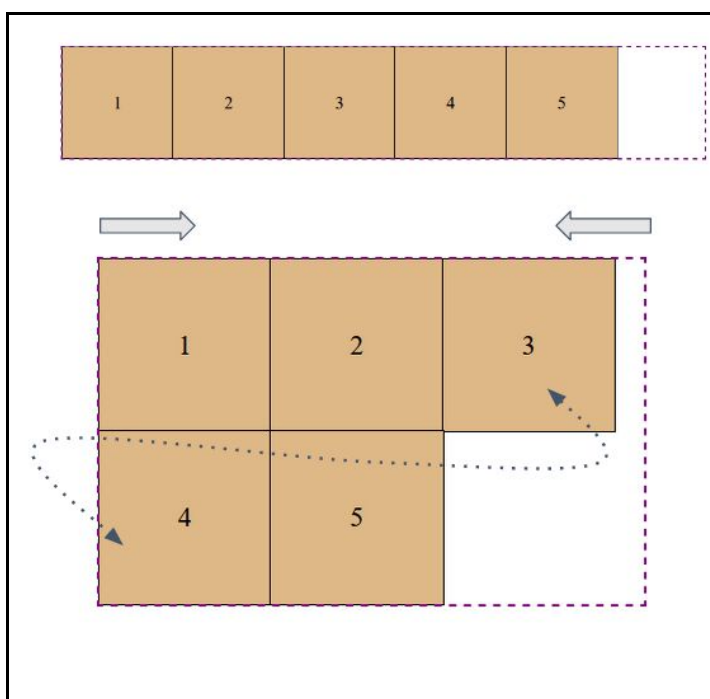


4.2 Disposición bidimensional

Podemos hacer que los elementos ocupen varias filas en el contenedor, para cuando estos ya no quepan en una única. De este modo no se producen desbordamientos o compresiones de elementos si es que estos pueden caber o ocupar nuevas líneas.

Por contra, la gestión de elementos se torna más compleja que si únicamente gestionamos en un único eje.

La propiedad que permite implementar esta disposición es **flex-wrap: wrap** (valor por defecto **no-wrap**). La idea es justamente que el contenedor “abraza” todos los elementos declarados internamente en el flujo HTML.

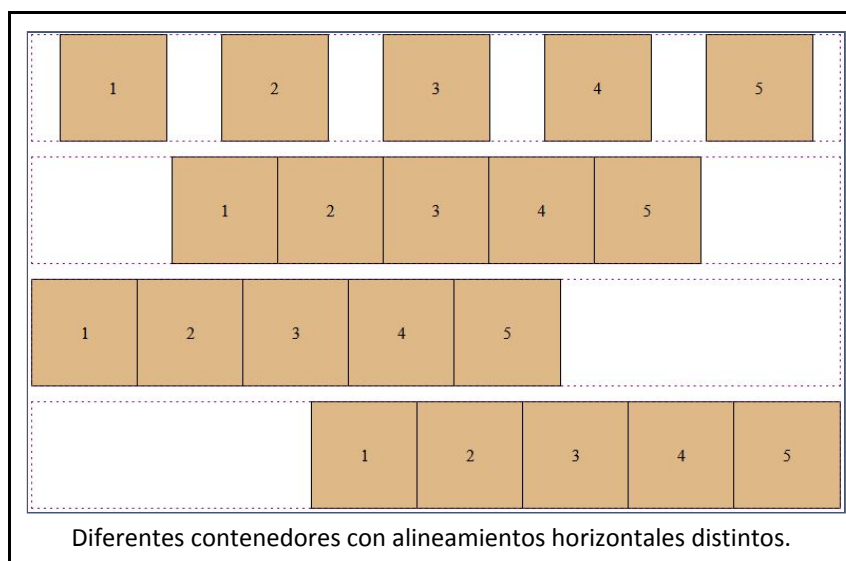


Presentadas las propiedades flex-direction y flex-wrap, existe para aquellos que quieran ahorrar líneas de código un **abreviado o shorthand** llamado **flex-flow** que permite declarar ambas propiedades en una sola sentencia.

4.3 Alineamiento del eje primario (main-axis)

La propiedad **justify-content** permite alinear los elementos en la horizontal si previamente no hemos invertido los ejes con la propiedad **flex-direction: column**.

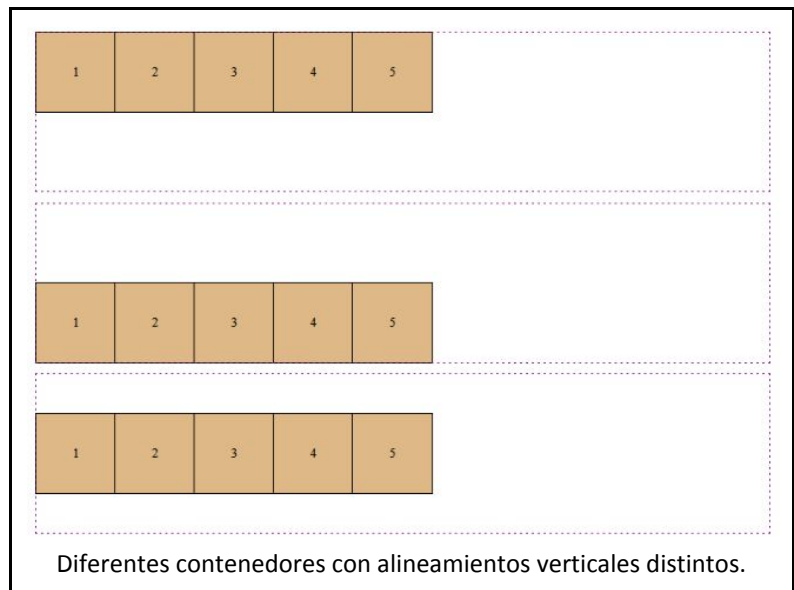
Los valores posibles son: **start**, **end**, **space between** (entre medias), **space around**, **space even** (repartición por igual).



4.4 Alineamiento del eje secundario (cross axis)

La propiedad **align-items** permite alinear los elementos en la **vertical** si previamente no hemos invertido los ejes con la propiedad **flex-direction: column**.

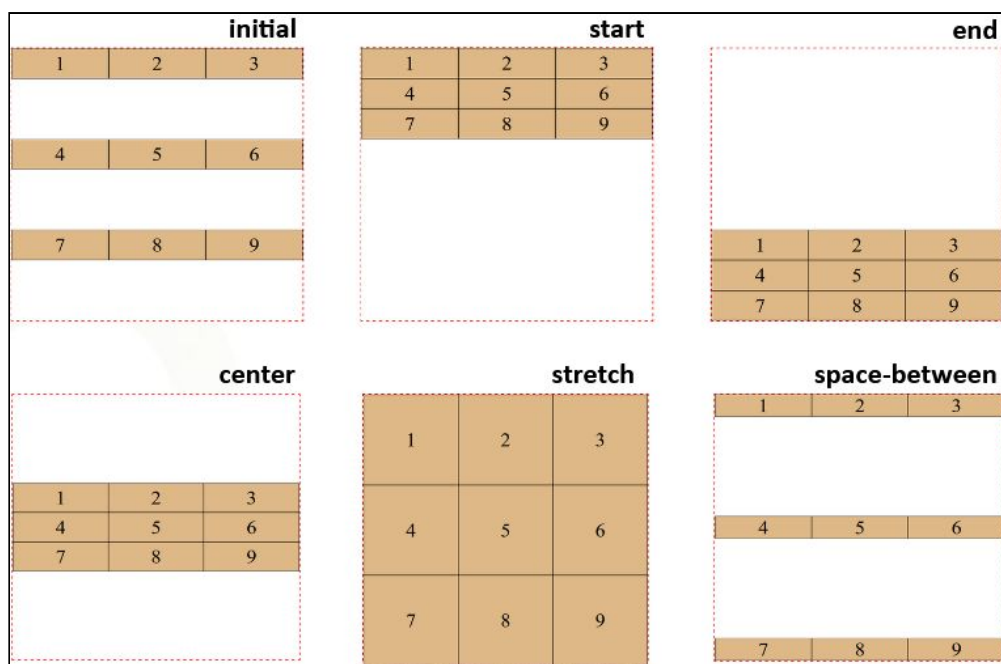
Los valores posibles en este caso son: **start**, **end**, **center**.



4.5 Gestión de líneas de elementos

Para cuando tenemos activa la regla CSS **flex-wrap: wrap** podemos gestionar la distribución de espacio de las líneas de elementos contenidas en una caja flex-box.

Los valores posibles en este caso son: **initial**, **start**, **end**, **center**, **stretch** (es preciso deshabilitar el alto de los elementos) o **space around/between**.



Diferentes distribuciones de filas de elementos para un contenedor con flex-wrap.

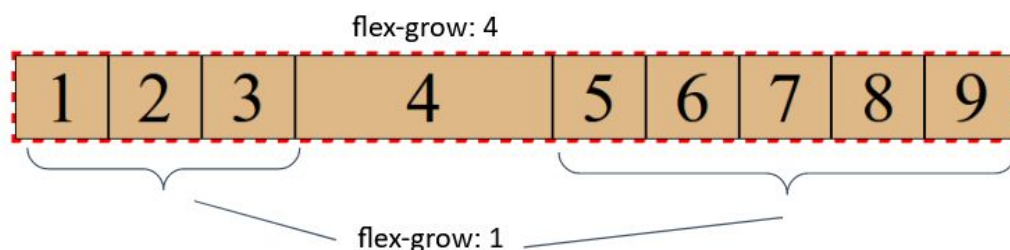
4.6 Control de crecimiento de elementos

Es posible definir valores a cada uno de los elementos a fin de controlar su crecimiento, decrecimiento y dimensiones iniciales (justo antes de participar en una operación de crecimiento o decrecimiento).

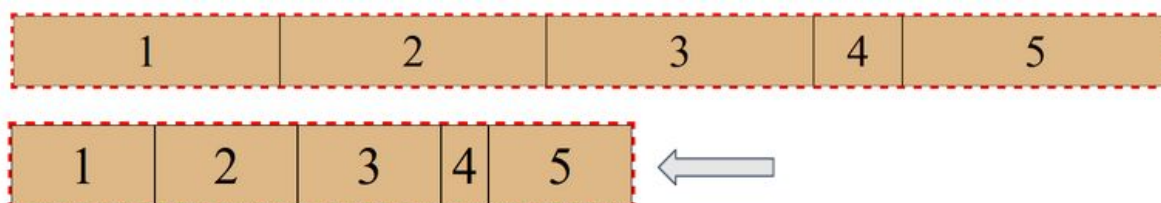
Los valores a aplicar a los elementos en este caso corresponden a `flex-grow`, `flex-shrink` y `flex-basis` respectivamente.

flex-grow permite definir un valor numérico (positivo) a cada elemento. Generalmente, elementos con un mayor número `flex-grow` declarado ocupan una mayor parcela de espacio dentro del contenedor padre con respecto al resto de elementos.

Por defecto los elementos participan con un valor `flex-grow` idéntico y por tanto crecen de forma homogénea.



flex-shrink permite al igual que `flex-grow` definir un valor numérico (positivo) a cada elemento. En este caso se trata de un factor de decrecimiento.



flex basis actúa como propiedad que define el tamaño inicial de un elemento flexible, justo antes de aplicarle factores de crecimiento o decrecimiento. Puede verse como la propiedad `width`, pero en el ámbito de `flex-box`. Esta propiedad es aplicada conjuntamente con `flex-grow` y `flex-shrink`.

A partir del valor `flex-basis` declarado el elemento, este escogerá / disminuirá su tamaño en función del crecimiento o decrecimiento del contenedor padre, el cual implementa `flex-box`.

4.7 Orden de elementos

La propiedad **order** aplicada a los elementos internos a una caja `flex-box` permite variar el orden de visualización de los mismos en el renderizado del navegador (por defecto todos tienen el valor 0).

Podemos puntualmente posicionar un elemento al principio asignando un valor negativo, de hecho cuanto más bajo sea este valor, ocupará posiciones anteriores). La reordenación de un número de elemento implica la definición concreta de la propiedad `order` de estos (y seguramente del resto).

5. CSS Grid

CSS Grid permite definir un layout bidimensional con gran cantidad de elementos en un área amplia como por ejemplo una página HTML entera con relativa sencillez.

El uso de CSS Grid no está reñido con flexbox, position o float. Simplemente es una herramienta más avanzada para lograr cómodamente lo que otras no permiten sin tener que trabajar muy muy duro.

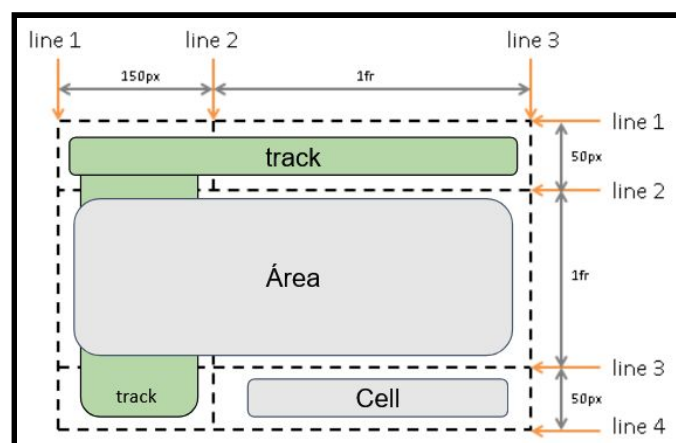
La filosofía es justamente que según el contexto del problema a resolver elijamos la herramienta de posicionamiento más conveniente, permitiendo así combinarlas bajo demanda.

5.1 Definiendo Grid

Declararemos la regla CSS `display: grid;` a un contenedor para que este adopte la capacidad de acomodar elementos. A priori no sucede nada visualmente (internamente sí).

La terminología de CSS grid será:

- **grid-container / grid-items (Existen en el DOM):** contenedor padre y contenedores hijo.
- **grid-tracks:** representan cada una de las filas o columnas que forman el grid.
- **grid-cells:** cada una de las intersecciones entre una fila y una columna.
- **grid-áreas:** una o grupo de celdas contiguas .
- **grid-lines:** margen declarado entre grid-cells y grid-áreas



Terminología CSS Grid

Seguiremos principalmente dos pautas de trabajo con Grid:

- La definición de plantillas (templates) para filas / columnas.
- La definición de áreas grid

5.2 Definiendo plantillas de filas y columnas

Definimos plantillas de filas y de columnas en el contenedor padre (quien implementa grid). Definir las dimensiones para cada uno de estos elementos implica su creación.

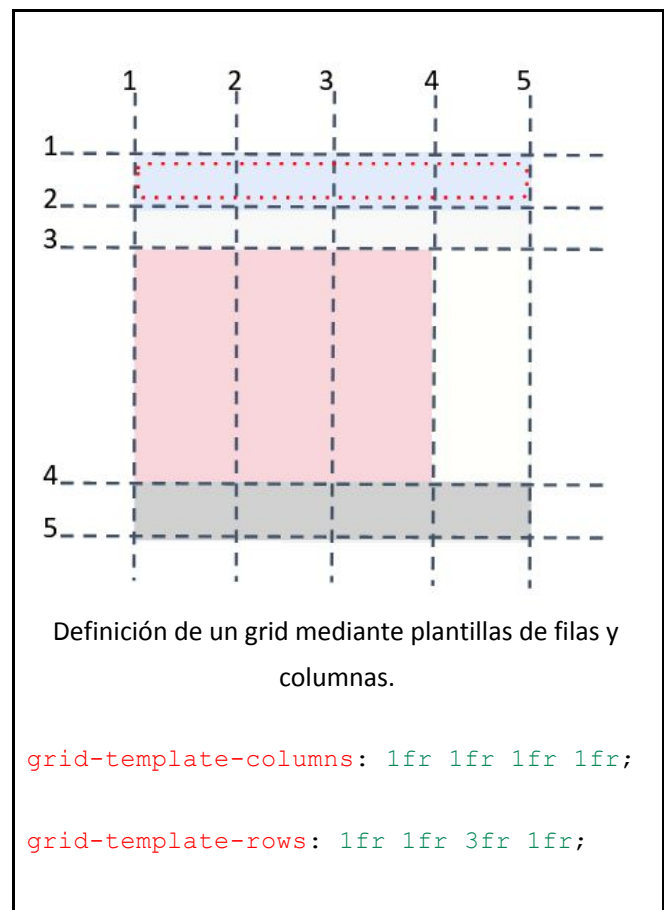
Las propiedades que permiten su definición son:

grid-template-columns y

grid-template-rows.

Se definen tantos valores como en tantas filas o columnas deseemos particionar el contenedor grid.

Se adjunta distribución de 4 columnas de igual tamaño (25%) y 4 filas con tamaños diferentes.



Tenemos posibilidad de definir estas anchuras de filas y columnas mediante valores referidos en medidas absolutas o relativas ya conocidas o mediante el uso de unidades de fracción (fr). En este último caso, pueden lograrse cómodamente fracciones exactas mediante la definición de un conjunto de filas/columnas cuyas dimensiones vengan referidas todas en unidades fr.

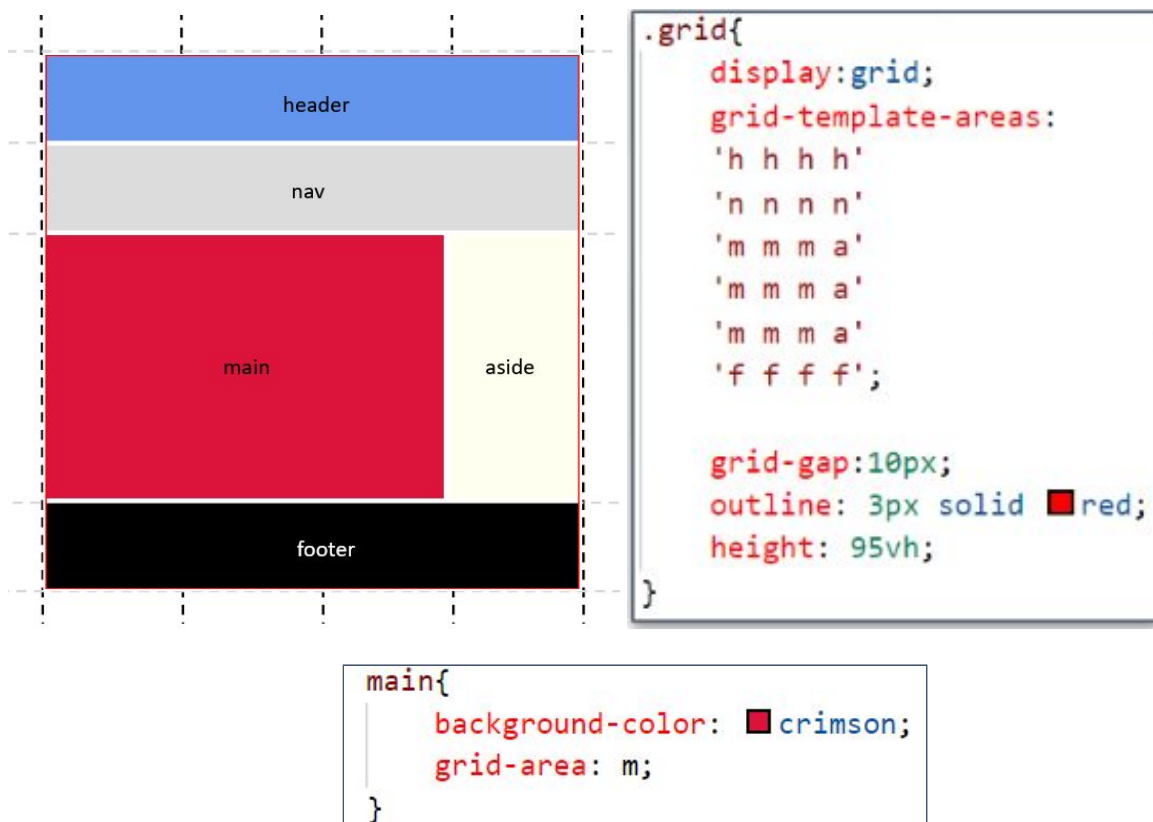
5.2 Definiendo áreas grid

La propiedad **grid-template-areas** permite hacer una declaración de cómo va a ser nuestro layout mediante una definición explícita de un String en el contenedor que implemente grid donde:

- Digamos que áreas intervendrán
- Expresar sus dimensiones en función de la repetición de una etiqueta textual que simboliza un área y su extensión.

Finalmente y tras haber declarado el String que define las dimensiones de cada una de las áreas del Grid, solo queda atribuir a cada elemento contenedor, en qué área va a participar con la propiedad **grid-area** y la asignación de la correspondiente etiqueta textual incluida en el String de definición.

¡Nunca antes fue tan fácil ordenar un layout con total independencia de la ordenación de elementos según el flujo HTML donde han sido inicialmente definidos!



Ejemplo de definición de áreas Grid.

6. Fondos

Hasta ahora hemos declarado coloreados de fondos para elementos, generalmente para su identificación cuando estábamos tratando de posicionarlos o para dotarlos de un color concreto.

También es posible poner de fondo una imagen o un gradiente de colores. Nótese que el poner de fondo una imagen, no tiene porque ser el equivalente a definir una imagen con el elemento ``.

Con `` seguramente queremos ver una imagen al completo y esta es la etiqueta más adecuada para ello. Poner de fondo una imagen puede o no tener como fin mostrar la totalidad de la imagen.

La propiedad `background` es un abreviado que engloba otras muchas otras propiedades:

- **background-color**: declaración de un color de fondo.
- **background-image**: declaración de una imagen de fondo
- **background-repeat**: permitir la repetición de un patrón de fondo
- **background-position**: definición de posicionamiento del fondo
- **background-size**: definir el tamaño del patrón de fondo.
- **background-clip**: definir dónde comienza a dibujarse el fondo según el modelo de caja.
- **background-attachment**: permitir fijar un fondo con respecto al scroll efectuado.

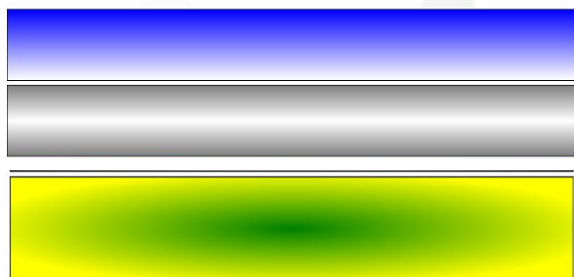
Es preferible la declaración de cada una de ellas cuando se precise pese a que esto implica tener más líneas código CSS. A continuación se mencionan las características más representativas:

background-color permite definir un color de fondo. Este color intentará ocupar toda el área del elemento donde es definido. Si hay un color y una imagen de fondo declarados, la imagen será presentada de frente (el color permanecerá oculto justo detrás).

Es posible declarar colores mediante las diferente opciones estudiadas en Unidad 4. Estas son: valores textuales, rgb/rgba, hexadecimal, hsl/hsla.

background-image permite definir una imagen de fondo mediante el atributo url y una ruta relativa y absoluta. Hemos de lidiar con el tamaño del contenedor y el propio de la imagen hasta lograr conseguir el efecto deseado.

`background-image` permite además definir gradientes de colores (lineales y radiales). Es posible declarar varios colores, que porcentaje ocuparan (paradas de color) y dirección y sentido angular para un mayor grado de personalización. Dejamos la guía de referencia W3Schools para más información.



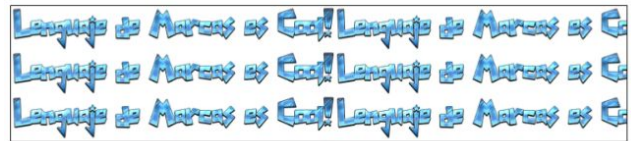
```
.caja:nth-child(1){
  background-image: linear-gradient(■blue, □white);
}

.caja:nth-child(2){
  background-image: linear-gradient(to bottom, ■grey, □white, ■grey );
}

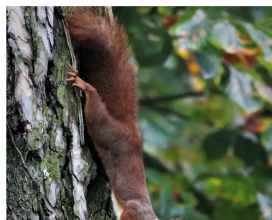
.caja:nth-child(3){
  background-image: radial-gradient( ■green, ■yellow 80% );
}
```

background-repeat permite especificar si el patrón de color, bien sea un gradiente o una imagen, se vaya a repetir en un eje, ambos o en ninguno.

Los valores posibles son: `repeat` (valor por defecto), `no-repeat`, `repeat-x` o `repeat-y`.



background-position permite alterar el posicionamiento de una imagen de fondo para así poder mostrar una zona de interés.



`background-position: center 10%;`



`background-position: center 90%;`

Otra perspectiva de esta herramienta puede ser el posicionar una imagen cuyas dimensiones sean inferiores a las del elemento donde está declarada como fondo. Esta propiedad trabaja con dos valores los cuales corresponden al eje x y eje y respectivamente. Es posible usar los valores textuales: `center`, `top`, `bottom`, `right` y `left`.



`background-position: left center;`



`background-position: center center;`



`background-position: right center;`



`background-position: left bottom;`



`background-position: center bottom;`

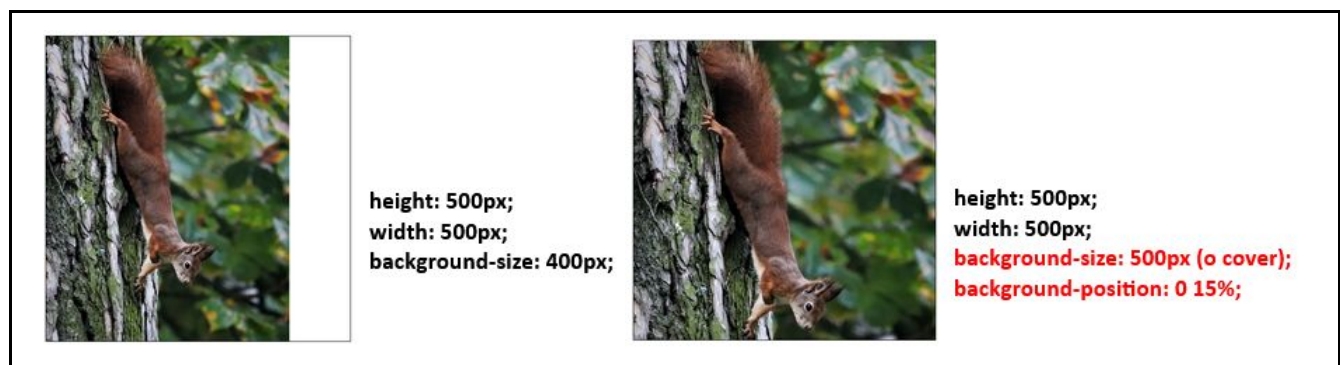


`background-position: right bottom;`

background-size permite definir el tamaño de la imagen de fondo declarada. Hay que resolver en muchas ocasiones el problema de desfase de tamaños entre el contenedor y las dimensiones originales de la imagen. Para conseguir escalados (sin deformaciones en la imagen) se precisa únicamente declarar una medida de las dos permitidas, lo que equivaldrá a definir su ancho (el alto se calculará de forma automática).

Con `background-size` con el valor `cover` permitirá redimensionar la imagen hasta cubrir por completo el contenedor con ella.

El trabajo combinado de `background-size` y `background-position` pueden hacer que te ahorres alguna edición de imagen para lograr el plano deseado.



Ajuste de una imagen de fondo con `background-size` y `background-position`.

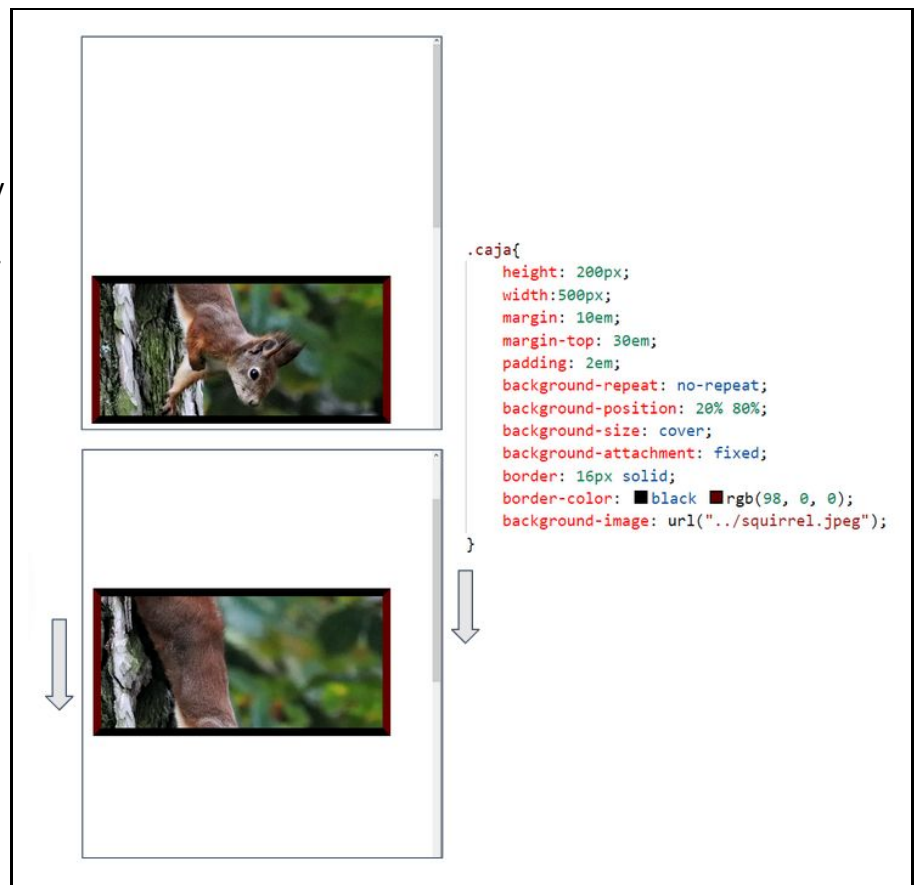
background-clip permite definir la extensión de un fondo en un elemento, bien sea este un color o una imagen. Esto es referido al modelo de caja y por tanto a si se han definido los valores: `content-box`, `padding-box` o `border-box` y respectivamente: el tamaño del contenido, la cantidad de padding o grosor de bordes declarados.



Aplicando diferentes valores para `background-clip`: `content-box` y `border-box`.

background-attachment

permite fijar una imagen con respecto al movimiento del scroll practicado por el usuario y lograr un efecto de “ventana de submarino” para un contenedor determinado.

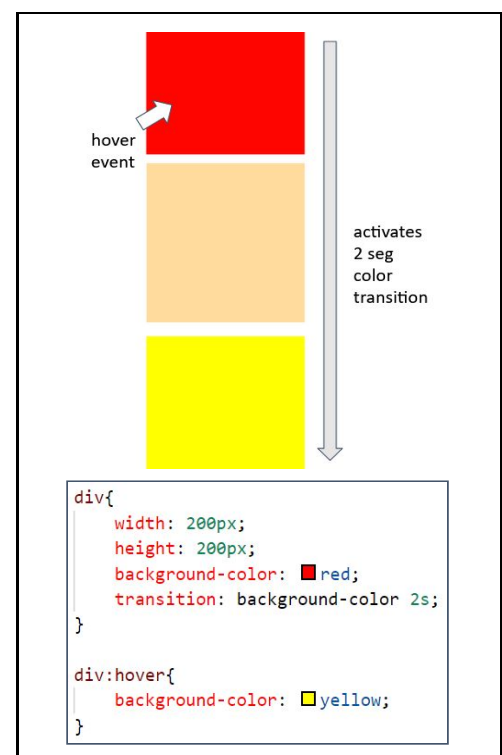


7. Animaciones

7.1 Concepto de transición

Podemos lograr efectos de transición variando una o varias propiedades CSS una vez suceda un evento. La brusquedad con la que sucede este cambio puede suavizarse aplicando la propiedad **transition** y definiendo un sutil espacio de tiempo (se pueden programar más aspectos).

No todas las propiedades CSS se prestan a ser transitorias. Sólo aquellas que puedan definirse mediante valores en un rango. El navegador se encargará de renderizar los estados intermedios de un extremo de la transición al otro.



7.2 Concepto de animación

En CSS podemos aplicar efectos de animación declarando una sucesión de estados/puntos intermedios que en conjunto proporcionan una sensación de cambio o movimiento. Puede verse como la suma de un conjunto de transiciones ejecutadas una a continuación de la otra.

Todas las propiedades CSS no son animables. Sólo aquellas que puedan definirse mediante valores en un rango (principalmente numérico). El navegador se encargará de renderizar los estados intermedios de un extremo de la animación al otro.

Las animaciones suponen un trabajo extra al navegador. Debemos tener presente esto para lograr diseños que puedan ejecutarse con relativa suavidad en navegadores que corren en equipos con bajos recursos computacionales.

7.3 @keyframes

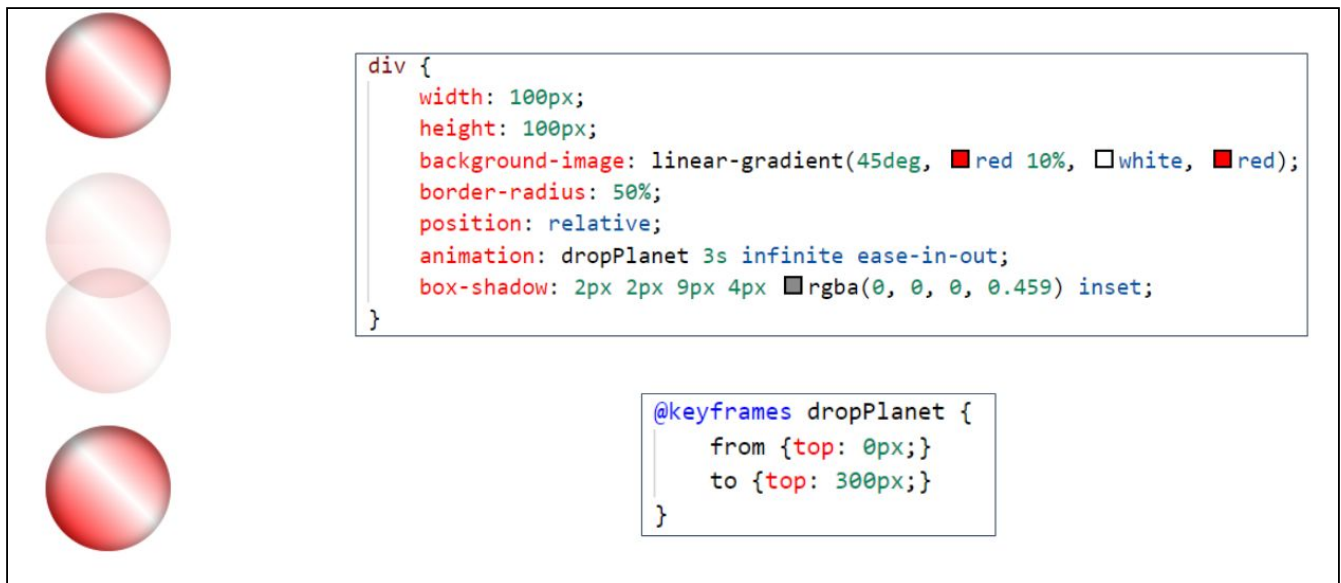
Podemos programar una transición con la anotación **@keyframes** y asignarle a esta un identificador para su invocación.

Declaramos los estados intermedios con:

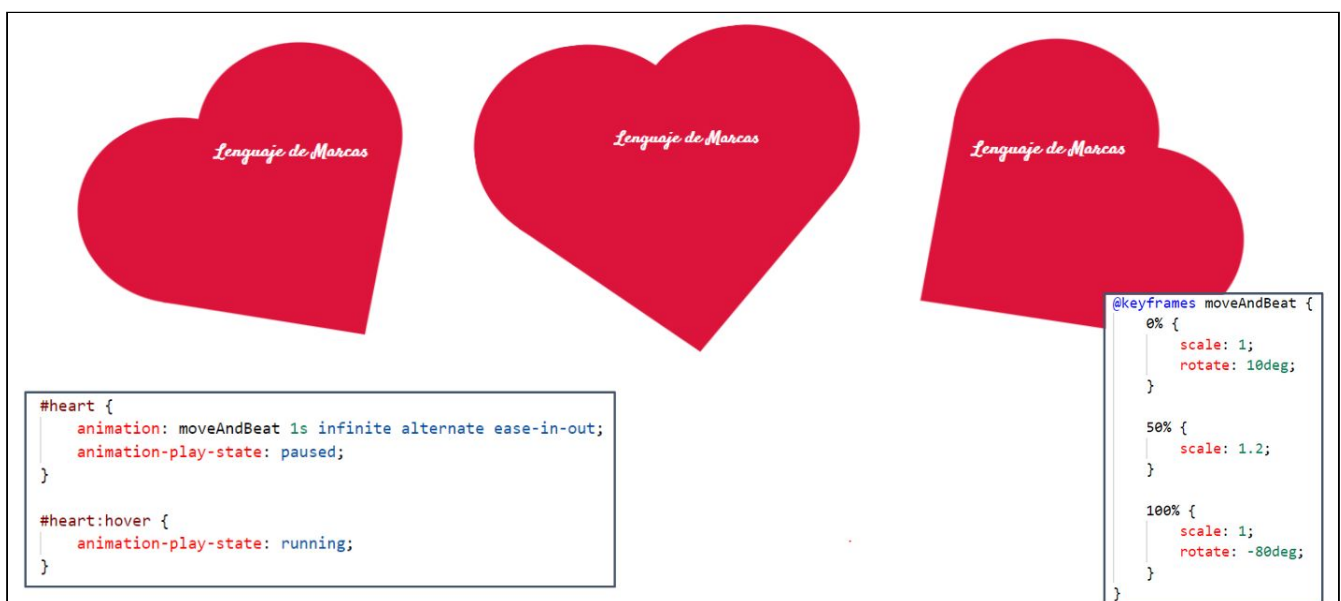
- **from / to:** animación simple que se compone de dos estados inicio y fin.
- **sucesión de bloques de instantáneas** donde se producen cambios **en base al porcentaje** que estas ocupan respecto a la duración total de la animación.

Aplicamos esta animación con la propiedad **animation** y programamos los aspectos: tiempo de duración, función de ida y venida, nº de veces ejecutada, tiempo de espera inicial, etc.

Recordad: La animación se desencadena tras un evento. Bien al satisfacer el evento que concuerda con un selector CSS basado en pseudoclases de estado o al escucharse un evento programado mediante Javascript.



Animación de tipo from-to.



Animación en base a sucesión de instantáneas en el tiempo.

Para más información consultar la guía de W3Schools. Recordad también que este tema se verá con más detalle en el módulo profesional de segundo curso: Diseño de Interfaces Web.

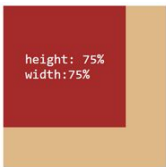


8. Iniciación al pensamiento RWD

El diseño web responsivo o adaptable es una disciplina que abarca algo más que:

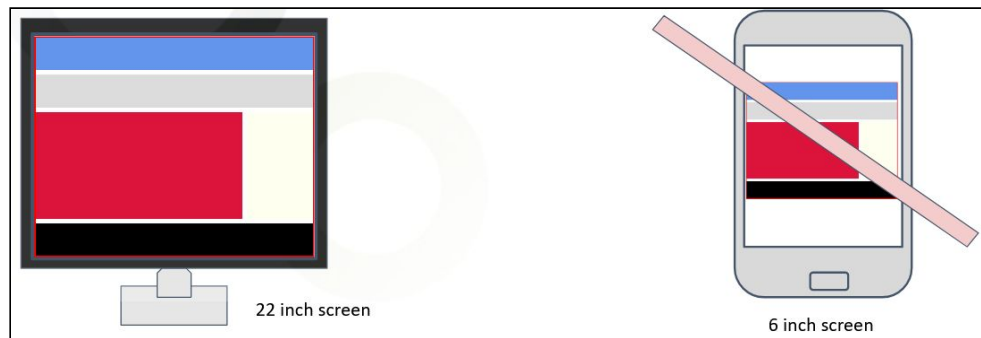
- El uso de medidas relativas: %, em, rem, vh, vw, min-height, min-width.
- El uso de CSS Flex-box / CSS Grid o cualquier herramienta de posicionamiento que se comporte en un determinado caso de forma dinámica con respecto a sus elementos gestionados. Hemos estudiado brevemente estas herramientas y sabemos de la posibilidad de crecimiento de sus elementos.
- El uso de media-queries (condicionales CSS).
- Lograr un diseño adaptativo para un único tipo de pantalla.

Concretamente, el RWD comporta la toma de decisiones para lograr adaptar todos y cada uno de los elementos de mi página haciendo uso de las herramientas anteriormente citadas, teniendo presente el mayor número de pantallas donde finalmente va a ser renderizada (móviles, tablets, smart tv, laptops, full HD Screens, etc).

El uso de medidas relativas como el porcentaje, em/rem y medidas basadas en viewport (vw/vh) permiten lograr un dinamismo con respecto a la referencia a la que están asociadas.

Porcentaje: siempre un elemento hijo podrá referirse a sus dimensiones en términos de porcentaje con respecto a las del elemento padre que lo contiene.	
Em: servirá como medida relativa que permitirá escalar un elemento concreto en base al font-size declarado en el mismo elemento y sus dimensiones tales como alto, ancho y padding referidas en esta unidad.	
Rem: tomando como referencia el font-size declarado en la raíz, permite aplicar medidas a todos los elementos del documento. Puede por tanto producir un escalado múltiple de elementos si a posteriori se varía el font-size de la raíz.	<pre>:root{ font-size: 16px; }</pre>
vh/vw: las medidas viewport permitirán que ciertos elementos que basen su tamaño en ellas escalen cuando varíe el tamaño del visor del navegador.	

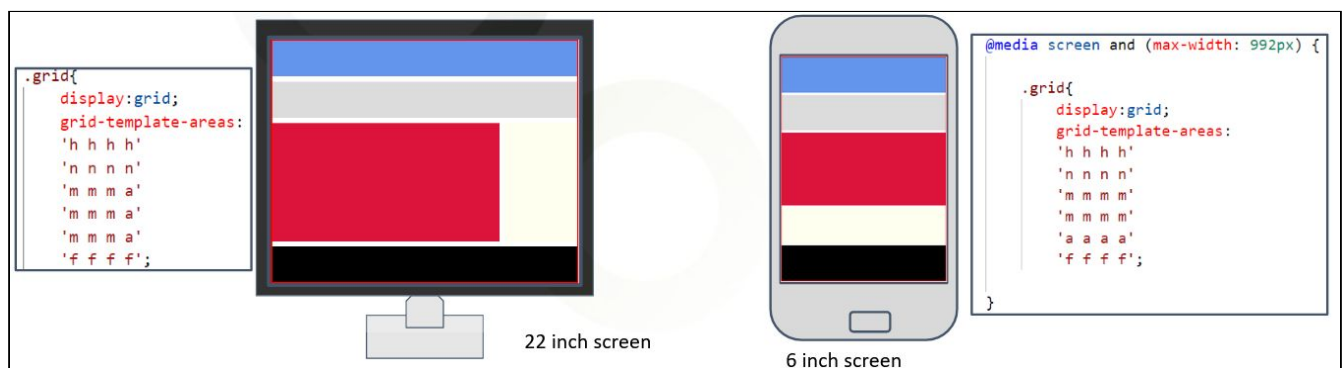
Seguramente las decisiones tomadas en un layout perfectamente diseñado para una pantalla de escritorio full hd no sean buenas para el visor de un dispositivo móvil.



Ejemplo de layout no directamente aplicable a terminales móviles.

Una solución que plantea CSS es el uso de **media queries**, las cuales pueden ser vistas como condicionales definidas que al satisfacerse una determinada condición permiten aplicar conjuntos de reglas CSS para un caso concreto (por ejemplo: una medida concreta de una pantalla).

Esto será estudiado con calma en segundo curso en el módulo profesional Diseño de Interfaces web.



Ejemplo de media query definida para lograr un diseño adaptativo móvil.

Formalmente debiéramos tener presentes unos tamaños de pantalla estandarizados y asegurar así la compatibilidad responsive de nuestros diseños con el mayor número de dispositivos.

Lo estudiaréis con calma en DIW y algún framework puede que hasta os eche un cable con ello!

10. Bibliografía

[1] <http://librosweb.es/libro/css/>

[2] <https://www.w3schools.com/>