Dando forma al Layout

Viewport units

vw

Unidad relativa a la anchura del *viewport* (normalmente, la ventana del navegador, pero podría ser un **iframe**). **100vw** equivale al 100% de la anchura del *viewport*. Es recomendable evitar aplicar un **100vw** a un elemento (o combinación de elementos) porque la posición lateral de la barra de *scroll* en función del sistema operativo podría provocar un poco de *scroll* horizontal en la página.

vh

Unidad relativa a la altura del *viewport*. **100vh** equivale al 100% de la altura del *viewport*.

vmin y vmax

Unidades equivalentes a **vw** (anchura) o **vh** (altura) del *viewport*, en función de cuál sea la dimensión menor de las dos (en el caso de **vmin**) o cuál sea la dimensión mayor de las dos (en el caso de **vmax**). Son unidades interesantes en muchas ocasiones, por ejemplo, en los que queremos que una imagen de contenido se vea lo más grande posible respecto al *viewport*, pero sin que quede cortada por una de sus dos dimensiones.

Imágenes de contenido con `object-fit`

`object-fit: …;`

La propiedad **object-fit** se aplica a una imagen de contenido para indicar cómo se redimensiona respecto a su contenedor, cuando dicha imagen tiene un tamaño relativo a dicho contenedor (ejemplo: **width: 100%** y **height: 100%**).

object-fit: cover;

La imagen de contenido mantiene sus relación de aspecto (no se deforma) y cubre completamente el contenedor, aunque quede alguna parte cortada.

`object-fit: contain;`

La imagen de contenido mantiene sus relación de aspecto (no se deforma) y se muestra completamente, aunque queden partes del contenedor sin cubrir.

`object-fit: fill;`
La imagen de contenido cubre todo el contenedor y se muestra completa, deformándose si es necesario.

`object-fit: none;`
Cancela la propiedad **`object-fit`**, para los casos en los sea necesario devolver la imagen a su visualización original.

`object-fit: scale-down;`
Funciona como un **`object-fit: none`** o un **`object-fit: contain`**, en función de cuál dé como resultado un tamaño de imagen más pequeño.

`object-position: …;`
Define cómo debe alinearse respecto al contenedor una imagen que tenga **`object-fit`**. Funciona de manera equivalente a **`background-position`**, admitiendo dos valores (alineación horizontal y vertical) que pueden ser palabras (**`left`**/**`right`**/**`top`**/**`bottom`**) o unidades relativas (como `%`) o absolutas (como **`px`**).

CSS columns

`column-count: …;`
Propiedad que podemos aplicar a un elemento que contenga texto para que se distribuya en columnas (tantas como la cifra que señalemos en esta propiedad). Con un valor de **`auto`**, el número de columnas se decide por otras propiedades, como **`column-width`**.

`column-width: …;`
Propiedad que podemos aplicar a un elemento que contenga texto para que se distribuya en columnas, teniendo como referencia ideal para definir estas columnas la anchura que le asignemos (en **`px`**, **`em`** u otras medidas).

`column-gap: …;`
Anchura del espacio entre columnas (**`px`**, `%` y otras unidades), en un elemento que se le ha aplicado alguna de las propiedades que provocan una distribución del contenido en columnas.

`column-rule: … … …;`
Línea separadora entre columnas que sigue las mismas reglas de un **`border`** (anchura, tipo de borde, color del borde).

CSS shapes

clip-path: polygon(... ..., ..., ...);
Área de recorte visual de un elemento (por ejemplo, una imagen) con un polígono definido con la función **polygon()**, que funciona de manera equivalente a un polígono SVG; cada par de valores separados por comas indican la posición horizontal y vertical de cada uno de los puntos que conforman el dibujo. Alternativamente, podemos usar la función **circle()** o **ellipse()** en lugar de **polygon()**.

circle(… at …)
Con la función **circle()** podemos darle forma circular (pasando un valor que indique el radio del círculo) y un segundo valor (tras el **at**) para indicar la posición a partir de la cual se realiza el recorte (por ejemplo **top left**, **bottom center** o **60%**).

ellipse(… … at …)
La función **ellipse()** funciona de manera similar a **circle()**, pero acepta dos valores de radio: uno para el horizontal y otro para el vertical.

shape-outside: polygon(… …, … …, … …) border-box;
Área de recorte, definida por un **polygon()** o equivalente, a partir de la cual el texto puede distribuirse alrededor (siempre y cuando dicho elemento flote). Esta área de recorte no tiene porqué coincidir con el área visual de recorte, **clip-path**. Añadir el valor **border-box** asegura que el área de recorte tome como referencia el borde del elemento, y no se vea afectada/deformada por su **margin**.

shape-margin: …;
Margen en **px** u otra unidad que podemos dejar entre la forma definida por **shape-outside** y el texto que se distribuye alrededor.

Variables CSS

--nombreVariable: …;
Las variables CSS se declaran dentro de una declaración CSS normal (como una propiedad más) y se escriben prefijadas de un doble guion y con un sistema de nomenclatura equivalente al de las clases.

propiedad: var(–nombreVariable);
Para asignar como valor una variable previamente declarada, englobaremos esta propiedad en la función **var()**. Adicionalmente, podemos asignar también como argumento un valor alternativo, separado por una coma, para el caso en el que la variable no tuviera un valor asignado.

Ejemplo:

```
font-size: var(--encabezado1, 24px);
```

Alcance de las variables

Las variables que se declaran dentro de `:root` o la etiqueta `html` tienen un alcance global. Las variables que se declaran en otros nodos o selectores están acotadas a ese nodo y sus descendientes únicamente. Las variables se pueden sobreescribir en nodos inferiores, haciendo que tengan un valor a nivel global (etiqueta `html`) y un valor diferente a nivel local (por ejemplo, en la clase `.destacado`).

`:root{ … }`

El selector `:root` equivale a la etiqueta `html`, y se utiliza habitualmente para establecer variables globales. La única diferencia entre `:root {}` y `html {}` es que el primero tiene mayor especificidad.

Flexbox

`display: flex;`
Propiedad que permite crear un espacio *flex*. Esta propiedad se aplica a un elemento contenedor, y los hijos directos de dicho contenedor se alinearán siguiendo las propiedades de *flexbox*.

`flex-direction: …;`
Propiedad de contenedor que indica la dirección del eje principal a partir del cual se alinearán los elementos. Los valores pueden ser **`row`** (fila, por defecto), **`column`** (columna), **`row-reverse`** (fila inversa) o **`column-reverse`** (columna inversa).

`flex-wrap: …;`
Propiedad de contenedor que permite indicar a un espacio *flex* si tiene que realizar un salto de línea cuando el contenido no tiene espacio suficiente para mostrarse. El valor por defecto es **`nowrap`** (no realizar el salto), y podemos indicar **`wrap`** (salto de línea) y **`wrap-reverse`** (salto invertido).

`flex-grow: …;`
Propiedad de elemento que permite indicarle si debe expandirse para ocupar el espacio sobrante. Por defecto es `0` (no expandirse), pero cualquier valor positivo asignará una proporción de reparto del espacio sobrante (si todos los elementos tienen `1`, lo reparten proporcionalmente, y si uno tiene un `2`, cogerá el doble de espacio sobrante).

`flex-shrink: …;`
Propiedad de elemento, equivalente a **`flex-grow`**, que permite indicar una proporción de reducción de un elemento cuando no tiene suficiente espacio. Por defecto es `1` (se reduce proporcionalmente), pero podemos dar un valor mayor para que reduzca más o un valor de `0` para que no reduzca.

`flex-basis: …;`
Propiedad de elemento que le permite establecer una anchura ideal (similar a un **`min-width`**) si no hay otros elementos condicionantes.

flex: …;
Forma reducida de escribir las propiedades **flex-grow**, **flex-shrink** y **flex-basis**, en ese orden y separados por espacios.

Ejemplo:

```
flex: 1 0 200px;
```

order: …;
Propiedad de elemento que permite cambiar el orden en el que se muestra. Por defecto, todos los elementos tienen un **order** de 0, por lo que cualquier valor negativo hará que un elemento aparezca antes que los demás, y si es positivo, después.

justify-content: …;
Propiedad de contenedor que permite alinear el contenido en el eje principal, con el valor por defecto **flex-start** (al inicio del eje principal) y los valores **flex-end** (al final del eje principal), **center** (centrado), **space-between** (repartiendo el espacio sobrante de manera proporcional entre elementos) o **space-around** (similar a **space-between**, pero incluyendo el principio y el final de la caja). Esta propiedad se amplió después con el valor **space-evenly**, que distribuye el contenido de una manera similar a **space-around**, pero asegurándose de que queda el mismo espacio entre elementos que al principio y al final del grupo.

align-items: …;
Propiedad de contenedor que permite alinear el contenido en el eje cruzado, con el valor por defecto **stretch** (estirar) y los valores **flex-start** (inicio del eje cruzado), **flex-end** (final de eje cruzado), **center** (centrado) y **baseline** (alineado con la línea base del texto).

align-content: …;
Propiedad de contenedor que permite alinear el contenido en el eje cruzado cuando hay más de una línea, con el valor por defecto **stretch** (estirar) y los valores **flex-start** (inicio del eje cruzado), **flex-end** (final de eje cruzado), **center** (centrado) y **space-between** (espacio entre medio) y **space-around** (espacio a los lados).

align-self: …;
Propiedad de elemento que permite cambiar la alineación de un elemento en el eje cruzado, con el valor por defecto **auto** (el que venga dado por el contenedor) y los valores **flex-start** (inicio del eje cruzado), **flex-end** (final de eje cruzado), **center** (centrado), **baseline** (alineado con la línea base del texto) y **stretch** (estirado).

CSS grid

`display: grid;`

Propiedad que permite crear un espacio *grid*. Los espacios *grid* se dividen en filas (**`rows`**) y columnas (**`columns`**). Estas filas y columnas reciben el nombre común de *tracks*. El espacio *grid*, igual que el espacio *flexbox*, es la relación entre un elemento contenedor con **`display: grid`** y sus elementos "hijos" (descendientes directos). Los descendientes de los descendientes directos no se ven afectados por este tipo de alineación.

Retícula implícita y retícula explícita

Cuando el contenido supera el número de filas o columnas que hemos planteado en nuestro código (retícula explícita), el espacio *grid* hace una prolongación de las columnas o filas respectivamente, creando lo que se conoce como retícula implícita.

`grid-template: … … … / … … …;`

La propiedad **`grid-template`** permite declarar el número de filas y columnas que tendrá un *grid* (separados por el carácter **`/`**). Cada medida separada por un espacio supone una nueva fila o columna respectivamente.

Ejemplo:

```
grid-template: 300px 1fr 200px / 200px 3fr 100px 2fr;
```

`grid-template-columns: …;`

Propiedad que permite declarar específicamente el número de columnas de un *grid*, de manera equivalente a la que se aplica para **`grid-template`**.

`grid-template-rows: …;`

Propiedad que permite declarar específicamente el número de filas de un *grid*, de manera equivalente a la que se aplica para **`grid-template`**.

`1fr`

Unidad que indica 1 fracción del espacio libre disponible. De esta forma, el espacio libre se dividirá entre las columnas o filas (respectivamente) que tengan esta unidad de medida, y proporcionalmente a la cifra que se les haya dado (**`1fr`**, **`2fr`**, etc).

`repeat(…, …)`

Función de repetición que se emplea dentro de las propiedades de **`grid-template`** para indicar que un número de veces (primer argumento de la función) se repetirá una fila o columna (según dónde la apliquemos) con un tamaño concreto (especificado en el segundo argumento de la función).

Ejemplo:
```
repeat(4, 1fr)
```

`grid-gap: … …;`
Espaciado entre tracks, en un espacio *grid*. Podemos especificar un valor para las filas y otro para las columnas (separadas por un espacio) o especificar un solo valor para ambos.

Ejemplo:
```
grid-gap: 20px;
```

`grid-auto-columns: …;`
Propiedad que especifica el tamaño que deberían tener las columnas adicionales que se creen en la retícula implícita.

`grid-auto-rows: …;`
Propiedad que especifica el tamaño que deberían tener las filas adicionales que se creen en la retícula implícita.

`grid-auto-flow: …;`
Propiedad de retícula que indica cómo deben posicionarse los elementos que no tienen instrucciones específicas de posicionamiento en el *grid* (y por tanto siguen un posicionamiento natural respecto a su orden en el **`HTML`**). El valor **`row`** (que tiene por defecto) indica que intente rellenar las filas, y añada nuevas filas si es necesario. El valor **`column`** rellenará por columnas, y añadirá nuevas columnas si es necesario. Y el valor **`dense`** intenta rellenar todos los huecos disponibles para que quede una retícula compacta.

`grid-column: … / …;`
Propiedad que permite indicar a un elemento dónde empieza y acaba respecto a las líneas que marcan las columnas de un *grid*. Si se indican valores negativos, se empieza a contar desde el final de las líneas de la retícula explícita. Opcionalmente, uno de los dos valores (principio o final) puede ser sustituido por la palabra **`span`** seguida de un valor, que indica la extensión del elemento a lo largo de dichas líneas de retícula.

Ejemplo:
```
grid-column: 3 / span 2;
```

`grid-row: … / …;`

Propiedad equivalente a **`grid-column`**, pero en este caso, se indica el principio y final de un elemento respecto a las líneas que marcan las filas del *grid*.

`grid-row-start, grid-row-end, grid-column-start` y `grid-column-end`

Propiedades que, en cada caso y de manera concreta, permiten indicar el inicio o el final de un elemento, respecto a las filas o las columnas.

`minmax(..., ...)`

Función de medida empleada en una propiedad **`grid-template`** que permite indicar un valor mínimo (primer argumento) y un valor máximo (segundo argumento) para una fila o una columna.

Ejemplo:

```
minmax(300px, 1fr)
```

`auto-fit, auto-fill`

Valores que permiten indicar el comportamiento de las filas o columnas de una retícula en el espacio sobrante. Se suele emplear dentro de la función **`repeat()`**.
En el caso de **`auto-fill`**, grid rellena tantas filas o columnas como quepan, incluso si quedan vacías. En el caso de **`auto-fit`**, se hace una distribución similar, pero las filas/columnas vacías se esconden y el espacio sobrante se reparte entre las filas/columnas que sí tienen contenido.

Ejemplo:

```
grid-template-columns: repeat( auto-fit, minmax(250px, 1fr) );
```

`auto, min-content, max-content`

Valores que podemos dar dentro de una propiedad **`grid-template`** para dimensionar una fila o columna. El valor **`min-content`** hace referencia al tamaño mínimo que puede tener el elemento en dicha fila/columna; el valor **`max-content`**, el tamaño máximo que puede ocupar dicho elemento. El valor **`auto`** dependerá del resto de contenidos, tomando como punto de partida el valor máximo.

Ejemplo:

```
grid-template-columns: repeat(auto-fill, minmax(0, max-content));
```

`grid-template-areas: … ;`

De manera complementaria a las propiedades de **`grid-template`**, podemos declarar **`grid-template-areas`** para dotar a áreas concretas de un *grid* con nombres clave. Podemos indicar cuántas columnas o filas ocupa una área repitiendo su nombre varias veces (una por cada columna o fila), y podemos separar cada fila agrupándolas entre comillas. El resultado es bastante visual, lo que ayuda a entender mejor la disposición de las áreas. Además, si dentro de nuestro *grid* no queremos nombrar un espacio concreto como área, podemos usar un punto (".") para indicar esto y se mostrará como una celda más.

Ejemplo:

```
grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
```

`grid-area: … ;`

Propiedad que permite identificar un elemento con una área, y haciendo que el mismo se distribuya conforme a lo designado en la propiedad **`grid-template-areas`**.

`order: …;`

Propiedad equivalente al **`order`** de *flexbox*, y que permite cambiar el orden en el que aparece un elemento en un espacio *grid*. Por defecto todos los elementos tienen un **`order: 0`**. Los valores negativos harán que aparezca antes que los elementos cuyo **`order`** no se haya alterado, y los valores positivos, después.

Reglas `justify-…`

Las reglas **`justify`** se aplican el eje *x*, y alinean el contenido dentro de un espacio *grid*.

`justify-content: …;`
Esta propiedad alinea las columnas de un *grid* cuando sobra espacio en dicho *grid*. Los valores que se pueden dar son: **`center`** (centrado), **`start`** (al inicio del espacio en el eje x), **`end`** (al final del espacio), **`stretch`** (las columnas crecen hasta ocupar todo el espacio), **`space-around`** (el espacio sobrante se reparte al principio y al final de cada columna), **`space-between`** (el espacio se reparte proporcionalmente entre las columnas) y **`space-evenly`** (el espacio se reparte como **`space-around`**, pero manteniendo la misma proporción entre columnas que en los exteriores).

`justify-items: …;`
Propiedad que alinea el contenido en el eje *x* en en interior de cada celda de la retícula. Los valores que se pueden dar son: **`center`** (centrado), **`stretch`** (ocupa el 100% de la celda), **`start`** (inicio de la celda), **`end`** (final de la celda).

`justify-self: …;`
Propiedad que alinea un elemento concreto dentro de su celda en el eje *x*, sin afectar al resto de elementos de la fila. Los valores que se pueden dar son: **`center`** (centrado), **`stretch`** (ocupa el 100% de la celda), **`start`** (inicio de la celda), **`end`** (final de la celda).

Reglas `align-…`
Las reglas align se aplican el eje *y*, y alinean el contenido dentro de un espacio *grid*.

`align-content: …;`
Reglas de alineación equivalentes a las de **`justify-content`**, pero en el eje *y*. Los valores que se pueden dar son los mismos.

`align-items: …;`
Reglas de alineación equivalentes a las de **`justify-items`**, pero en el eje *y*. Los valores que se pueden dar son los mismos.

`align-self: …;`
Reglas de alineación equivalentes a las de **`justify-self`**, pero en el eje *y*. Los valores que se pueden dar son los mismos.

`minmax(0, 1fr)`
En retículas de anchura flexible, un valor mínimo de `0` asegura que el contenido no modifique las dimensiones del *track* (y si es el caso, lo sobrepase).

Sistema de retícula anidada

La propiedad **`subgrid`**, que permitiría pasar unas reglas de alineación de *grid* a elementos anidados, no está implementada aún en los navegadores. Eso significa que para mantener una retícula general consistente necesitamos replicarla en elementos anidados. Una forma de hacerlo con cierta consistencia es guardar los valores de **`grid-template`** en una variable para poder replicarlos fácilmente. Otra forma podría ser usar el valor **`inherit`**, es decir, hereda lo que tenga el elemento padre. Por ejemplo, en el caso de que el elemento anidado tenga la misma anchura que el padre, podríamos asignarle al elemento hijo estas propiedades:

```
display: grid;
grid-template-columns: inherit;
```

Profundidad y Scrolls

3D transforms

Transformaciones 3D

Las transformaciones 3D no están pensadas para realizar un modelado 3D complejo (para esto ya existen otras herramientas en la web), sino para añadir pequeñas mejoras a nuestras interfaces. Las funciones de transformación 3D son: **`rotateX()`**, **`rotateY()`**, **`rotateZ()`**, **`translateZ()`** y **`scaleZ()`**. Estas funciones utilizan el mismo tipo de unidades que sus equivalentes 2D. Además de estas, existen las versiones abreviadas **`rotate3d()`**, **`translate3d()`** y **`rotate3d()`**, que permiten dar los 3 valores de la función (uno por eje), separados por comas.

`perspective: …;` o `perspective(…)`

La perspectiva es necesaria para activar el espacio 3D. Esta perspectiva puede aplicarse a un contenedor para que los elementos en su interior que sean transformados tengan una perspectiva común (emplearíamos entonces la propiedad **`perspective: … ;`**) o bien se puede aplicar a un elemento concreto que estemos transformando, a través de la función **`perspective(…)`**, como una función más en la propiedad **`transform`**. La perspectiva se puede marcar en **`px`**, y representa la distancia del espectador respecto al elemento transformado. A menor valor, más cerca está el espectador y más dramático es el efecto.

`perspective-origin: … …;`

Podemos controlar el punto de fuga de la perspectiva con esta propiedad, dando valores absolutos o relativos para el eje *x* e *y*.

Por ejemplo:

```
perspective-origin: 25% 75%;
```

`backface-visibility: hidden;`

Cuando damos la vuelta a un elemento para mostrar su cara trasera u oculta (como en el caso de **`rotateY()`**), al ser la web un entorno no preparado por defecto para el 3D, vemos el mismo elemento al revés. Podemos anular este efecto utilizando la propiedad **`backface-visibility: hidden`**, a partir de la cual el elemento queda oculto al ser volteado. Sin embargo, es conveniente que haya otro elemento (previamente volteado y con su propio **`backface-visibility: hidden`**) que aparezca a continuación, si lo que queremos es conseguir un efecto de tarjeta con dos caras.

`transform-style: preserve-3d;`

Esta propiedad permite que los descendientes de un elemento que reciba una transformación 3D la hereden también, y se comporten de una manera coherente a esta transformación, en vez de quedar "planos" respecto al elemento transformado.

Cabeceras fijas

`position: fixed;`

La propiedad **`position`** con el valor **`fixed`** funciona de una manera similar a **`position: absolute`**, pero toma como referencia la ventana del navegador, no la página, por lo que no se desplaza al hacer *scroll*. Para posicionarla se usan las propiedades **`top`**, **`bottom`**, **`left`** y/o **`right`**.

`position: sticky;`

La propiedad **`position`** con el valor **`sticky`** muestra un elemento inicialmente en su posición natural, pero a medida que hagamos *scroll*, cuando dicho elemento esté a una distancia del borde superior de la ventana del navegador equivalente a la que hayamos dado con la propiedad **`top`**, pasará a comportarse como si fuera **`position: fixed`**.