# UNIDAD 2 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.