

---

## DISEÑOS BASADOS EN DIMENSIONES

JavaScript es un lenguaje de programación interpretado, basado en el estándar ECMAScript (European Computer Manufacturer's Association Script). Se caracteriza por ser un lenguaje de programación orientado a eventos y basado en prototipos, dinámico y no demasiado tipado.

### 6.1 TABLAS

---

Las tablas no son nada más que una forma de organizar la información a través de filas y columnas. El problema reside cuando estas estructuras contienen mucha información y no pueden representarse de manera correcta en dispositivos con poca resolución o de pequeño tamaño.

En este capítulo vamos a ver cómo definir tablas, cómo hacerlas decorativas, cómo hacerlas adaptativas o Responsive y cómo hacerlas usables y accesibles.

#### 6.1.1 Elementos disponibles en HTML5

##### 6.1.1.1 ELEMENTO CAPTION

El elemento `CAPTION` especifica que el contenido que se va a representar es el título de una tabla. Sólo puede definirse un elemento `CAPTION` por tabla y es importante que el elemento `CAPTION` sea el primer hijo directo del elemento `TABLE`.

##### 6.1.1.2 ELEMENTO TABLE

El elemento `TABLE` especifica que el contenido que se va a representar es una estructura de datos tabulados en forma de filas y columnas, es decir, una tabla.



Entre los atributos que admite en su configuración, se deben destacar **BORDER**, **CELLPADDING**, **CELLSPACING** y **WIDTH**, pero todos ellos es mejor declararlos a través de sus homólogos de CSS.

Las tablas es uno de los elementos de HTML menos accesibles que, a menudo, encontramos en las páginas. Primero porque los desarrolladores no conocen todas las posibilidades de configuración y, segundo, porque si no se ve toda ella en su conjunto puede ser algo muy difícil de entender o contextualizar. Como ejemplo extremo, piénsese que, si un usuario sólo puede ver un dato en una tabla que, además, no presenta una cabecera por la circunstancia que sea, puede no saber a qué se refiere dicho dato.

Por tanto, si se han de utilizar, se deben especificar las dimensiones en términos de porcentaje y establecer todas sus propiedades para que no se pierda semántica y/o accesibilidad.

La declaración de los elementos de cabecera y pie de tabla (**THEAD** y **TFOOT**) deben establecerse antes que el elemento del contenido de la tabla **TBODY** para que el agente de usuario pueda renderizar la información de contexto antes de recibir el detalle con todas las filas de datos, que pueden ser muchas.

Cabe destacar que, los atributos **ID**, **HEADERS** y **SCOPE**, no tienen ningún efecto visual, sin embargo, junto con el elemento **CAPTION**, son muy útiles para las tecnologías asistidas como los lectores de pantalla puesto que aclaran y fortalecen su significado.

#### 6.1.1.3 ELEMENTO COLGROUP

El elemento **COLGROUP** especifica que el contenido que se va a representar es un grupo de una o más columnas de una tabla. Suele ser útil para aplicar estilos de forma agrupada en vez de tener que repetirlos de uno en uno.

Es importante que el elemento **COLGROUP** sea hijo directo del elemento **TABLE**, que esté declarado justo después del elemento **CAPTION** y justo antes de los elementos **THEAD**, **TBODY** o **TFOOT** porque, de no ser así, puede afectar a la usabilidad web y a la accesibilidad web.

Para especificar o definir las propiedades de cada columna dentro de cada elemento **COLGROUP** se debe utilizar el elemento **COL**. Este elemento sólo permite el atributo **SPAN** para definir el número de columnas que debe abarcar.

```
<colgroup>
<col style="background: whitesmoke;"></col>
<col span="2" style="background: lavender;"></col>
</colgroup>
```

#### 6.1.1.4 ELEMENTOS THEAD Y TFOOT

El elemento **THEAD** especifica que el contenido que se va a representar es el encabezado de una tabla. El elemento **TFOOT** es idéntico al elemento **THEAD**, con la diferencia de que el contenido que se va a representar es el pie de página de una tabla.

Cabe destacar que los elementos **THEAD** y **TFOOT** deben declararse justo después del elemento **CAPTION** y **COLGROUP** y justo antes del elemento **TBODY**. También es importante

constatar que el elemento **THEAD** no se debe omitir puesto que su omisión puede perjudicar de forma notable a la usabilidad web y a la accesibilidad web de la página.

#### 6.1.1.5 ELEMENTO **TBODY**

El elemento **TBODY** especifica que el contenido que se va a representar es el cuerpo de una tabla.

Cabe destacar que elemento **TBODY** debe declararse justo después de los elementos **THEAD** y **TFOOT**. Además, no se debe omitir puesto que su omisión puede perjudicar de forma notable a la usabilidad web y a la accesibilidad web de la página.

#### 6.1.1.6 ELEMENTO **TR**

El elemento **TR** especifica que el contenido que se va a representar es una fila perteneciente a un encabezado, cuerpo o pie de página una tabla.

#### 6.1.1.7 ELEMENTO **TH**

El elemento **TH** especifica que el contenido que se va a representar es una celda de encabezado.

Entre los atributos que admite en su configuración, se deben destacar **COLSPAN**, que especifica el número de columnas que se deben unificar, **ROWSPAN**, que especifica el número de filas que se deben unificar, **ID**, que especifica el identificador de la columna y que es necesario para utilizarlo con el atributo **HEADERS** del elemento **TD**, **HEADERS**, que especifica la lista de identificadores únicos (separados por espacios en blanco) que se corresponden con los atributos **ID** pertenecientes a los elementos **TH** y **SCOPE**, que especifica un único valor que vincula la información entre las celdas de la cabecera y las celdas de datos para indicar si una celda de encabezado es un encabezado para una columna, una fila o un grupo de columnas o un grupo de filas.

#### 6.1.1.8 ELEMENTO **TD**

El elemento **TD** especifica que el contenido que se va a representar es una celda de datos.

Entre los atributos que admite en su configuración, se deben destacar **COLSPAN**, que especifica el número de columnas que se deben unificar, **ROWSPAN**, que especifica el número de filas que se deben unificar y **HEADERS**, que especifica la lista de identificadores únicos (separados por espacios en blanco) que se corresponden con los atributos **ID** pertenecientes a los elementos **TH**.

### 6.1.2 Elementos disponibles en CSS

A continuación, se muestran las propiedades de CSS que están expresamente dedicadas a tablas.



### 6.1.2.1 PROPIEDADES BORDER-COLLAPSE

Especifica si se deben fusionar o separar los bordes del elemento. Entre sus posibles valores podemos encontrar **COLLAPSE**, que indica que los bordes deben fusionarse cuando sea posible y no es efectivo cuando se encuentra en conjunción con las propiedades **EMPTY-CELLS** y **BORDER-SPACING** y, **SEPARATE**, que es el valor por defecto e indica que los bordes deben mostrarse separados e independientes para cada celda o elemento.

#### **NOTA**

La propiedad **BORDER-COLLAPSE** sólo es válida para los elementos **TABLE**, **TH** y **TD**.

### 6.1.2.2 PROPIEDAD BORDER-SPACING

Especifica la distancia entre los bordes de las celdas adyacentes, siempre y cuando la propiedad **BORDER-COLLAPSE** esté establecida a **SEPARATE**. Sus posibles se deben asignar a través de un valor establecido en una de las medidas permitidas de CSS.

#### **NOTA**

La propiedad **BORDER-COLLAPSE** sólo es válida para los elementos **TABLE**, **TH** y **TD**.

### 6.1.2.3 PROPIEDAD CAPTION-SIDE

Especifica la posición del título de una tabla. Entre sus posibles valores podemos encontrar **BOTTOM**, que indica que el título debe estar debajo de la tabla y **TOP**, que indica que el título debe estar encima de la tabla. Es el valor por defecto.

### 6.1.2.4 PROPIEDAD EMPTY-CELLS

Especifica si se deben mostrar o no los bordes de las celdas vacías. Entre sus posibles valores podemos encontrar **HIDE**, que indica que NO se deben mostrar y **SHOW**, que es el valor por defecto e indica que se deben mostrar.

### 6.1.2.5 PROPIEDAD TABLE-LAYOUT

Especifica el modo en el que se tienen que diseñar celdas, filas y columnas de la tabla. Entre sus posibles valores podemos encontrar **AUTO**, que indica que el ancho de la columna debe establecerse sin romper el texto o contenido de las celdas y **FIXED**, que indica que el ancho de la tabla será gestionado por el usuario y que, las columnas, deberán ser gestionadas por el ancho de las celdas de la primera fila. Si no se estableciesen anchos en la primera fila, los anchos de las columnas se dividirán por partes iguales, independientemente de su contenido.

### 6.1.3 Creación de tablas responsive

Las tablas son, quizás, el componente menos flexible que ofrece HTML. Sin embargo, gracias a CSS y JavaScript es posible hacer que esta característica se vuelva algo menos rígida. Por ejemplo, supongamos una tabla de datos como la siguiente:

EJEMPLO DE TABLA ADAPTATIVA						
ID	Empresa	F. Movimiento	Tipo	Concepto	Importe	Estado
1	Consultores SA	30-12-2019	Ingreso	Nómina	+1268.00 €	Efectuado
2	Carrefour	01-01-2020	Recibo	Supermercado	-128.56 €	Efectuado
3	El Corte Inglés	03-01-2020	Recibo	Chaqueta hombre L-XL	-99.99 €	Pendiente
4	El Corte Inglés	03-01-2020	Recibo	Pantalón hombre M-L	-48.50 €	Pendiente

Si probásemos esta tabla en un dispositivo móvil, lo más probable es que viésemos una barra de desplazamiento horizontal y, dependiendo de la resolución del dispositivo, de los tamaños de fuente y de los estilos agregados, puede que hasta prácticamente nada de información útil.

Para solucionar este supuesto, a continuación, se muestran algunas de las técnicas para hacer que las tablas de HTML puedan verse en cualquier dispositivo sin perder legibilidad e independientemente de sus dimensiones o densidad.

#### 6.1.3.1 MEDIANTE BARRAS DE DESPLAZAMIENTO

Esta técnica consiste en definir normalmente la tabla y aplicarle unas consultas de medios cuando se produce una condición determinada, como pueda ser el ancho del dispositivo.

Es la técnica más sencilla de todas, pero no la que mejor se adapta a las condiciones del dispositivo ya que, si los datos son muy largos, puede no verse casi nada de información.

La técnica consiste en ajustar el tamaño de la tabla al 100% del ancho del dispositivo, cambiar el modo de representación de la tabla a BLOCK, en vez de TABLE, y habilitar el desplazamiento horizontal en la misma.

---

#### Código CSS

```
@media screen and (max-width: 620px) {  
  table { display: block; overflow-x: auto; width: 100%; }  
}
```

## Posible resultado

EJEMPLO DE TABLA ADAPTATIVA						
ID	Empresa	F. Movimiento	Tipo	Concepto	Importe	Estado
1	Consultores SA	30-12-2019	Ingreso	Nómina	+1268.00 €	Efectu
2	Carrefour	01-01-2020	Recibo	Supermercado	-128.56 €	Efectu
3	El Corte Inglés	03-01-2020	Recibo	Chaqueta hombre L-XL	-99.99 €	Pendi
4	El Corte Inglés	03-01-2020	Recibo	Pantalón hombre M-L	-48.50 €	Pendi

### 6.1.3.2 MEDIANTE CONSULTAS DE MEDIOS

Al igual que sucede con la técnica de la barra de desplazamiento horizontal, esta técnica consiste en definir normalmente la tabla y aplicarle unas consultas de medios cuando se produce una condición determinada, como pueda ser el ancho del dispositivo.

Aunque esta técnica resulta ser algo más tediosa y laboriosa, es la que mejor se adapta a las condiciones del dispositivo si no se desea recurrir a JavaScript. Por ello, es una de las más utilizadas en situaciones reales.

La técnica consiste en ajustar el tamaño de la tabla al 100% del ancho del dispositivo, ocultar los campos de cabecera, cambiar el modo de representación de las celdas y, agregar unos atributos personalizados con los nombres de las columnas para utilizarlos como identificadores de campo.

Estos identificadores de campo serán mostrados a través de pseudo-elemento BEFORE en la parte izquierda de cada celda cuando las condiciones de la consulta de medios se cumplan y, para que los valores de estos campos no pierdan legibilidad, el valor de las celdas se alineará a la derecha, todo ello, además, con la intención de aprovechar, al máximo, el espacio disponible.

### Código CSS

```
html,
body { margin: 0; padding: 0; font-family: 'Roboto', sans-serif;
display: block; font-size: 14px;
}

table { border: 1px solid rgba(0,0,0,0.2); border-spacing: 2px;
margin: 10px 0; }

table caption { background: #000000; color: #fff; font-size: 1.0rem;
font-weight: bold; line-height: 1.5; padding: 0;
text-transform: uppercase; }
```

```

table td,
table th { border: 1px solid rgba(0,0,0,0.2); border-spacing: 0;
font-size: 1rem; padding: 5px; text-align: left;
margin: 2px 0; white-space: nowrap; }

table thead th:nth-child(6),
table tbody td:nth-child(6){ text-align: right; }

@media screen and (max-width: 620px) {
table { width: 100%; }

thead { display: none; }

tr td:first-child { background: #f0f0f0; font-weight: bold; }

tbody td { display: block; text-align: right; }

tbody td:before { content: attr(data-field); display: block;
float: left; font-weight: bold; padding: 0 10px 0 0;
text-align: left; width: auto; }
}

```

Cabe destacar que este código CSS se alimenta de un atributo personalizado DATA-FIELD que debe estar definido en cada elemento TD de la tabla y que debe ser idéntico al texto contenido dentro del elemento TH.

En lo referente a este último código de CSS, y al igual que pasaba con la técnica anterior, el cambio de comportamiento se realiza cuando la resolución llega al valor de 620 píxeles.

### Posible resultado

EJEMPLO DE TABLA ADAPTATIVA	
ID	1
Empresa	Consultores SA
F. Movimiento	30-12-2019
Tipo	Ingreso
Concepto	Nómina
Importe	+1268.00 €
Pago	Efectuado
ID	2



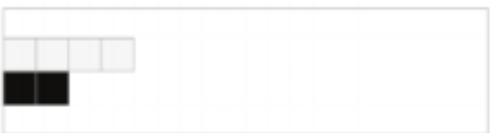
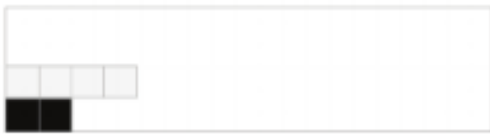
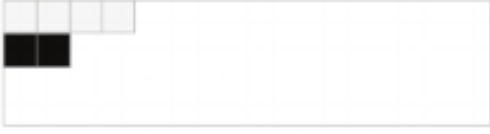
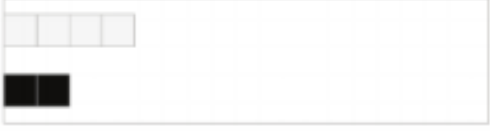


## 6.2 DISEÑO BASADO EN CAJAS FLEXIBLES (FLEXBOX)

Las cajas flexibles no son nada más que otra de las formas de organizar la información a través de filas y columnas, pero, al contrario que las tablas, sus elementos pueden manipularse, ensancharse o encogerse para rellenar el espacio adicional y, con ello, representarse de manera correcta en dispositivos con poca resolución o de pequeño tamaño.

### 6.2.1 Principales elementos disponibles en CSS

#### 6.2.1.1 PROPIEDAD ALIGN-CONTENT

Especifica cómo se deben distribuir los elementos verticalmente. Es una propiedad similar a `ALIGN-ITEMS`, pero en lugar de alinear elementos flexibles, alinea líneas flexibles. Entre sus posibles valores podemos encontrar:






<b>CENTER:</b> indica que las líneas de elementos deben distribuirse verticalmente por la zona media del contenedor flexible.	
<b>FLEX-END:</b> indica que las líneas de elementos deben distribuirse verticalmente por la zona final del contenedor flexible.	
<b>FLEX-START:</b> indica que las líneas de elementos deben distribuirse verticalmente por la zona inicial del contenedor flexible.	
<b>SPACE-AROUND:</b> indica que las líneas de elementos deben distribuirse verticalmente de forma uniforme por el contenedor flexible con espacios perceptibles en cada extremo.	
<b>SPACE-BETWEEN:</b> indica que las líneas de elementos deben distribuirse verticalmente de forma uniforme por los extremos del contenedor flexible.	
<b>STRECTCH:</b> indica que las líneas de elementos deben ajustarse verticalmente para ocupar o rellenar el espacio restante. Es el valor por defecto.	

## NOTA

La propiedad `ALIGN-CONTENT` sólo tendrá algún efecto cuando el modo de visualización (`DISPLAY`) sea `FLEX` y la propiedad `FLEX-WRAP` esté establecida a `WRAP` o a `WRAP-REVERSE`.

### 6.2.1.2 PROPIEDAD `ALIGN-ITEMS`

Especifica la alineación predeterminada para los elementos que están dentro de un contenedor flexible. Entre sus posibles valores podemos encontrar:






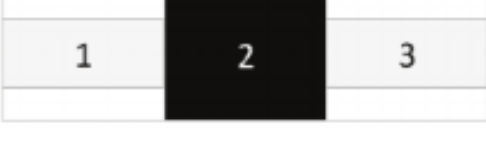
<b>BASELINE:</b> indica que los elementos deben estar posicionados en la línea base del contenedor flexible.	
<b>CENTER:</b> indica que los elementos deben estar posicionados en la parte central del contenedor flexible.	
<b>FLEX-END:</b> indica que los elementos deben estar posicionados al final del contenedor flexible.	
<b>FLEX-START:</b> indica que los elementos deben estar posicionados al principio del contenedor flexible.	
<b>STRETCH:</b> indica que los elementos deben ajustarse al alto del contenedor para rellenarlo. Es el valor por defecto.	

## NOTA

La propiedad `ALIGN-ITEMS` sólo tendrá algún efecto cuando el modo de visualización (`DISPLAY`) sea `FLEX` y puede anularse a través de la propiedad `ALIGN-SELF`.

### 6.2.1.3 PROPIEDAD ALLIGN-SELF

Especifica la alineación determinada para un elemento que está dentro de un contenedor flexible. Entre sus posibles valores podemos encontrar:

<b>AUTO:</b> indica que la alineación es inherente y que debe heredarse de la propiedad <b>ALIGN-ITEMS</b> definida en su contenedor. Es el valor por defecto.	
<b>BASELINE:</b> indica que el elemento debe estar posicionado en la línea base del contenedor flexible.	
<b>CENTER:</b> indica que el elemento debe estar posicionado en la parte central del contenedor flexible.	
<b>FLEX-END:</b> indica que el elemento debe estar posicionado al final del contenedor flexible.	
<b>FLEX-START:</b> indica que el elemento debe estar posicionado al principio del contenedor flexible.	
<b>STRECTCH:</b> indica que el elemento se debe ajustar al alto del contenedor para rellenarlo.	

#### **NOTA**

La propiedad **ALIGN-ITEMS** sólo tendrá algún efecto cuando el modo de visualización (**DISPLAY**) sea **FLEX**.

#### 6.2.1.4 PROPIEDAD FLEX

Es una propiedad compuesta que especifica, de forma conjunta, las propiedades de crecimiento flexible, decrecimiento flexible y el ancho del elemento.

El crecimiento viene determinado por la propiedad `FLEX-GROW` y se establece a través de un número que indica cómo irá creciendo el elemento con respecto al resto de elementos flexibles.

El decrecimiento viene determinado por la propiedad `FLEX-SHRINK` y se establece a través de un número que indica cómo irá decreciendo el elemento con respecto al resto de elementos flexibles.

El ancho viene determinado por la propiedad `FLEX-BASIS` y se establece a través de alguna de las unidades de medida estándar de CSS.

Si se asignan los tres valores, se aplicarán en el orden anteriormente indicado, es decir, es como si se estableciese de forma independiente las variables `FLEX-GROW`, `FLEX-SHRINK` y `FLEX-BASIS`, en este orden.

```
li { flex: 1 1 auto; } /* FLEX-GROW FLEX-SHRINK FLEX BASIS */
```

Si se asignan dos valores, se podrán establecer o el crecimiento y el ancho, o el crecimiento y el decrecimiento. Es decir, es como si se estableciese de forma independiente las variables `FLEX-GROW` y `FLEX-BASIS` o `FLEX-GROW` y `FLEX-SHRINK`, en este orden.

```
p { flex: 1 100%; } /* FLEX-GROW FLEX-BASIS */  
p { flex: 1 1; } /* FLEX-GROW FLEX-SHRINK */
```

Si se asigna un único valor, podrá aplicarse o un crecimiento o un ancho, es decir, es como si se estableciese de forma independiente la variable `FLEX-GROW` o la variable `FLEX-BASIS`.

```
p { flex: 1; } /* FLEX-GROW */  
p { flex: 100%; } /* FLEX-BASIS */
```

#### 6.2.1.5 PROPIEDAD FLEX-BASIS

Especifica el ancho inicial de un elemento flexible. Entre sus posibles valores podemos encontrar **AUTO**, que indica que el ancho es igual a la anchura predefinida del elemento flexible o, en ausencia de valor, en función de su contenido y, **[VALOR]**, que indica un valor establecido en una de las medidas permitidas de CSS.

Por ejemplo, imaginemos que tenemos un contenedor flexible con un ancho de 100 píxeles con tres elementos, en donde cada uno de ellos, tiene establecidas las propiedades `FLEX-GROW` y `FLEX-SHRINK` a 0 y la propiedad `FLEX-BASIS` a 33px. Esto debería producir un resultado similar al siguiente:





Ahora, si establecemos la propiedad `FLEX-BASIS` a 0 al segundo elemento, el resultado debería ser similar al siguiente:



Pero, si estableciésemos la propiedad `FLEX-BASIS` a 50px para el segundo elemento el resultado debería ser similar al siguiente:



Como se puede apreciar en la ilustración, el elemento 3 no entra en el contenedor de forma completa y se ve desbordado.

#### 6.2.1.6 PROPIEDAD `FLEX-DIRECTION`

Especifica la dirección de los elementos flexibles. Entre sus posibles valores podemos encontrar:

- **COLUMN:** indica que los elementos deben mostrarse verticalmente empezando por arriba. Un ejemplo podría ser que todos los elementos se sitúen, unos debajo de otros, desde arriba del contenedor en formación de A-B-C-D.
- **COLUMN-REVERSE:** indica que los elementos deben mostrarse verticalmente, empezando por abajo y con los elementos invertidos de orden. Un ejemplo podría ser que todos los elementos se sitúen, unos encima de otros, desde abajo del contenedor en formación de D-C-B-A.
- **ROW:** indica que los elementos deben mostrarse horizontalmente, empezando por la izquierda. Un ejemplo podría ser que los elementos se situasen todos seguidos y alineados a la izquierda en la parte superior del contenedor en formación de A-B-C-D. Es el valor por defecto.
- **ROW-REVERSE:** indica que los elementos deben mostrarse horizontalmente, empezando por la derecha y con los elementos invertidos de orden. Un ejemplo podría ser que los elementos se situasen todos seguidos y alineados a la derecha en la parte superior del contenedor en formación de D-C-B-A.

### 6.2.1.7 PROPIEDAD FLEX-FLOW

Es una propiedad compuesta que especifica la dirección de los elementos flexibles y si deben ajustarse o no al ancho del contenedor.

El ajuste de los elementos viene determinado por la propiedad `FLEX-WRAP`, mientras que la dirección viene determinada por la propiedad `FLEX-DIRECTION`. El orden de asignación es arbitrario, es decir, se puede realizar la asignación de la propiedad a través de la dirección y el ajuste, o a la inversa.

En general, se recomienda utilizar esta, y las demás formas abreviadas, debido a que su interpretación y renderizado se realiza algo más rápido.

### 6.2.1.8 PROPIEDAD FLEX-GROW

Especifica la relación de crecimiento del elemento con respecto a los demás. Entre sus posibles valores podemos encontrar un valor `[NÚMERO]` y que es un valor entero que indica, por decirlo así, el factor de multiplicación con respecto a los demás. Esto es, si todos los elementos de un contenedor flexible tienen un valor asignado de 1, menos uno que tiene un valor de 3, eso querrá decir que ese elemento será tres veces mayor que el resto.

### 6.2.1.9 PROPIEDAD FLEX-SHRINK

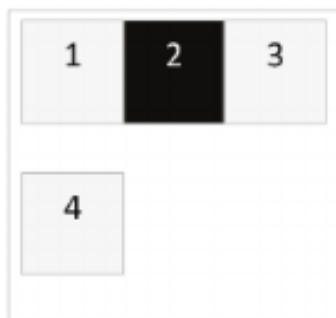
Especifica la relación de decrecimiento del elemento con respecto a los demás. Entre sus posibles valores podemos encontrar un valor `[NÚMERO]` y que es un valor entero que indica, por decirlo así, el factor de división con respecto a los demás. Esto es, si todos los elementos de un contenedor flexible tienen un valor asignado de 1, menos uno que tiene un valor de 3, eso querrá decir que ese elemento será tres veces menor que el resto.

### 6.2.1.10 PROPIEDAD FLEX-WRAP

Especifica si el elemento debe ajustarse o no al ancho del contenedor. Entre sus posibles valores podemos encontrar:

- **NOWRAP**: indica que el elemento no debe ajustarse. Es el valor por defecto.
- **WRAP**: indica que el elemento debe ajustarse si fuese necesario.
- **WRAP-REVERSE**: indica que el elemento debe ajustarse si fuese necesario, pero en orden inverso.

Por ejemplo, imaginemos que tenemos un contenedor flexible que tiene un ancho de 150 píxeles y, dentro, tiene definidos cuatro elementos de 40 por 40 píxeles cada uno. Dependiendo de cómo se establezca la propiedad `FLEX-WRAP`, debería producirse algo similar a uno de los siguientes resultados:



WRAP



WRAP-REVERSE



NOWRAP

#### 6.2.1.11 PROPIEDAD JUSTIFY-CONTENT

Especifica la alineación horizontal para los elementos flexibles cuándo éstos no utilizan, o no cubren, todo el espacio disponible. Entre sus posibles valores podemos encontrar:

<b>CENTER:</b> indica que los elementos deben estar posicionados en la parte central del contenedor flexible.	
<b>FLEX-END:</b> indica que los elementos deben estar posicionados a la derecha del contenedor flexible.	
<b>FLEX-START:</b> indica que los elementos deben estar posicionados a la izquierda del contenedor flexible.	
<b>SPACE-BETWEEN:</b> indica que los elementos deben ajustarse de forma que los espacios adyacentes sean iguales.	
<b>SPACE-AROUND:</b> indica que los elementos deben ajustarse de forma que los espacios entre ellos sean iguales, a excepción del primer y último elemento, en donde los espacios, anterior al primer elemento, y posterior al último elemento, deben ser la mitad que el espacio que hay entre el resto de los elementos.	

#### 6.2.1.12 PROPIEDAD ORDER

Especifica el orden de un elemento flexible con respecto al resto de elementos que tiene a su mismo nivel. Entre sus posibles valores podemos encontrar un valor [NÚMERO] que es un valor entero el cual indica el orden de aparición en la horizontal de izquierda a derecha. Por defecto, su valor es 0.



Para verlo claro, si, por ejemplo, tuviésemos un contenedor flexible con tres elementos y no estableciésemos la propiedad ORDER, los elementos aparecerían colocados según orden de aparición, es decir, **1, 2, 3**. Sin embargo, si estableciésemos al primer elemento un ORDER: 2 y al segundo un ORDER: 1, lo que veríamos es que el orden de aparición en pantalla sería **2, 1, 3**.

## 6.2.2 Creación de flexbox responsive

La creación de una estructura tipo tabla a través de cajas flexibles puede llegar a ser una tarea bastante tediosa y con comportamientos algo indeseables, como que el ancho de las celdas no se suele ajustar al ancho del contenido. Sin embargo, responden muy bien a todo tipo de resoluciones.

Dicho esto, y para ayudar a comprender mejor todo esto de las cajas flexibles, vamos a intentar implementar el mismo conjunto de datos que usamos con las tablas.

Si nos fijamos en el resultado podremos ver que, siendo los mismos datos, el modo de presentarlos en pantalla es muy diferente. Esto es, básicamente, porque las cajas flexibles están pensadas para establecer contenidos adaptables en función del ancho y no para presentar datos como si fuesen tablas.

EJEMPLO DE TABLA CON FLEXBOX CSS						
ID	Empresa	F. Movimiento	Tipo	Concepto	Importe	Estado
1	Consultores SA	30-12-2019	Ingreso	Nómina	+1268.00 €	Efectuado
2	Carrefour	01-01-2020	Recibo	Supermercado	-128.56 €	Efectuado
3	El Corte Inglés	03-01-2020	Recibo	Chaqueta hombre L-XL	-99.99 €	Efectuado
4	El Corte Inglés	03-01-2020	Recibo	Pantalón hombre M-L	-48.50 €	Efectuado

Para hacer esto basta con crear una estructura de datos a modo de un DIV contenedor que posea tantos DIV como filas tenga (incluyendo la cabecera y el título de la tabla) y, dentro de cada uno de estos, tantos DIV como columnas tenga cada fila. Algo como:

```
<div class="flexbox">
  <div class="caption">EJEMPLO DE TABLA CON FLEXBOX CSS</div>

  <div class="row header">
    <div class="col">ID</div>
    <div class="col">Empresa</div>
    <div class="col">F. Movimiento</div>
    <div class="col">Tipo</div>
    <div class="col">Concepto</div>
    <div class="col">Importe</div>
    <div class="col">Estado</div>
```



```

</div>

<div class="row">
<div class="col">1</div>
<div class="col">Consultores SA</div>
<div class="col">30-12-2019</div>
<div class="col">Ingreso</div>
<div class="col">Nómina</div>
<div class="col">+1268.00 €</div>
<div class="col">Efectuado</div>
</div>

<div class="row">...</div>
...
</div>

```

Después, sólo necesitaremos definir el sistema de cajas flexibles a las clases `.ROW` y `.COL`, y unos cuantos estilos adicionales:

---

## Código CSS

```

.flexbox .caption { display: block; text-align: center; font-weight: 600;
background: #000; color: #fff; }
.flexbox { border: 1px solid #ccc; }
.flexbox .row { display: flex; width: 100%; max-width: 100%;margin: 0;}
.flexbox .col { border: 1px solid #ccc; display: flex;
flex-flow: column nowrap; justify-content: flex-start;
align-items: flex-start; flex: 1 1 100%; margin: 1px;
padding: 0 5px; max-width: calc(100% / 7); }
.flexbox .header .col { font-weight: 600; }

```

## 6.3 DISEÑO BASADO EN CUADRÍCULAS (GRID LAYOUT)

---

El diseño basado en cuadrículas no es más que un sistema más actual para realizar diseños de estructuras bidimensionales. Sin embargo, tiene una gran diferencia y es que no requiere de contenedores diferenciables para filas y columnas ya que nos permite alinear los elementos a través de CSS, lo que ahorra en HTML y disminuye la carga del DOM (Document Object Model y representa la interfaz de programación para documentos HTML y XML), el cual se verá más adelante.

El diseño en Grid Layout se puede utilizar para obtener muy diversos resultados, pero desde una perspectiva diferente a las vistas hasta ahora. Dado que puede ser algo muy complicado y largo de explicar, aquí presentaremos lo más básico para empezar a trabajar. Si se desea más información se recomienda visitar la página de MDN Web Docs en [https://developer.mozilla.org/es/docs/Web/CSS/CSS\\_Grid\\_Layout](https://developer.mozilla.org/es/docs/Web/CSS/CSS_Grid_Layout) o la página de CSS Tricks <https://css-tricks.com/snippets/css/complete-guide-grid/>, la cual está en inglés.

## 6.3.1 Principales elementos disponibles en CSS

### 6.3.1.1 PROPIEDAD DISPLAY

Especifica que vamos a definir un contenedor de cuadrículas. Entre sus posibles valores podemos encontrar **GRID**, que indica que se va a definir un grid a nivel de bloque y **INLINE-GRID**, que indica que se va a definir un grid a nivel de línea.

### 6.3.1.2 PROPIEDADES GRID-TEMPLATE-ROWS Y GRID-TEMPLATE-COLUMNS

Especifican las filas y columnas de la cuadrícula mediante una lista de valores que definen el tamaño y espacio entre sus elementos separados por espacios. El tamaño puede ser descrito a través de una de las unidades de medida de CSS o por la palabra clave **FR**, que es lo más frecuente y representa una fracción del espacio libre en la cuadrícula.

El siguiente ejemplo describiría un grid de 4 filas por 7 columnas:

```
.grid {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr;  
  grid-template-rows: 1fr 1fr 1fr 1fr;  
  gap: 5px;  
}
```

Suponiendo que tengamos un contenedor grid con 28 elementos de caja (p.e. DIV) como hijos directos. El resultado debería ser similar a:



### 6.3.1.3 PROPIEDADES GRID-ROW-START Y GRID-ROW-END, GRID-COLUMN-START, GRID-COLUMN-END

Especifican la ubicación de los elementos dentro de la cuadrícula haciendo referencia a posiciones específicas.

Mientras que las propiedades **GRID-COLUMN-START** y **GRID-ROW-START** son para asignar la posición donde comienzan, **GRID-COLUMN-END** y **GRID-ROW-END** son para posición donde terminan. Para que lo veamos un poco más claro, veamos el siguiente ejemplo:

```
.large-item {  
  grid-column-start: 2;  
  grid-column-end: five;  
  grid-row-start: row1-start;  
  grid-row-end: 3;  
}
```

Suponiendo que tengamos un contenedor grid con 19 elementos de caja (p.e. DIV) como hijos directos. Si a uno de estos DIV le asignamos esta clase, el resultado debería ser similar a:



#### 6.3.1.4 PROPIEDAD ALIGN-ITEMS Y JUSTIFY-ITEMS

Especifican cómo se deben distribuir los elementos horizontal y/o verticalmente. Estas propiedades son similares a sus homólogas de Flexbox **ALIGN-ITEMS** y **JUSTIFY-CONTENT**, pero en lugar de alinear elementos flexibles, alinea cuadrículas.

Entre sus posibles valores podemos encontrar **STRETCH**, que es el valor por defecto e indica que las cuadrículas se ajusten al alto disponible de la celda, **START**, que indica que los elementos se coloquen en la parte inicial de su celda, **END**, que indica que los elementos se coloquen en la parte final de su celda, **CENTER**, que indica que los elementos se coloquen en la parte central de su celda y **BASELINE**, que indica que los elementos se alineen a lo largo de la línea de base del texto.

Para que veamos un poco el comportamiento de estas propiedades lo mejor es que lo pongamos en práctica, sin embargo, a continuación, mostraremos un caso de uso particular que es cuando, ambas propiedades, están declaradas como **CENTER**.

```
.grid {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr;  
  grid-template-rows: 1fr 1fr 1fr 1fr;  
  gap: 5px;  
  align-items: center;  
  justify-items: center;  
}
```

El resultado debería ser similar a:



### 6.3.1.5 FUNCIÓN REPEAT Y LAS PALABRAS CLAVE

La función **REPEAT** es un método elegante que nos permite ahorrar tiempo a la hora de definir el tamaño de las cuadrículas o celdas. Por ejemplo, en vez de usar la definición anterior que se mostró en las propiedades **GRID-TEMPLATE-ROWS** y **GRID-TEMPLATE-COLUMNS**, podemos escribir:

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(7, 1fr);  
  grid-template-rows: repeat(4, 1fr);  
  gap: 5px;  
}
```

No obstante, la potencia de esta función reside en las palabras clave que puede utilizar. Entre sus posibles palabras clave hay que destacar **AUTO-FILL**, que indica que se ajusten tantas columnas como sea posible en una fila, incluso si, éstas, están vacías, **AUTO-FIT**, que indica que las columnas se coloquen según el espacio disponible y **MINMAX**, que indica o establece el ancho mínimo y máximo para cada cuadrícula o celda.

Por ejemplo, en el caso anterior que teníamos un grid con 28 celdas, y que respondería perfectamente con el código mostrado en la parte superior de este mismo apartado, podríamos haber definido un ajuste automático con unos valores de máximo y mínimo predefinidos:

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));  
  grid-template-rows: minmax(max-content, 1fr);  
  gap: 5px;  
}
```

Sin embargo, esta regla CSS tiene un gran problema y es que sólo nos resultará válida cuando el ancho del contenedor o elemento padre sea múltiplo entero del número de columnas, en este caso 7. La razón de por qué no sería válido es porque, en cuanto el ancho del contenedor o padre sea 8 o más, las celdas que deberían formar una columna se mostrarán en diagonal.

En realidad, este es un problema que hemos causado a propósito para ver una casuística específica, pero, al definir **GRID-TEMPLATE-COLUMNS** como un **REPEAT** sencillo y poner el **GRID-TEMPLATE-ROWS** como se indica en este último ejemplo, se consigue un comportamiento bastante similar al de las tablas de HTML, todo ello, con considerable menos código CSS.

### 6.3.2 Creación de grid responsive

La creación de una estructura tipo tabla a través de grids puede llegar a ser una tarea algo confusa si no se sabe muy bien lo que hacer, pero, al final resulta un método más



que sencillo para formatear datos en dos dimensiones y responden muy bien a todo tipo de resoluciones.

Dicho esto, si nos fijamos en el resultado podremos ver que, siendo los mismos datos que hemos ido mostrando a lo largo de este capítulo, el modo de presentarlos en pantalla puede llegar a ser muy diferente a uno u otro modelo. Esto es, básicamente, porque las cuadrículas están pensadas para establecer contenidos adaptables en función del ancho y presentarlos como si fuesen tablas.

EJEMPLO DE TABLA CON GRID CSS						
ID	Empresa	F. Movimiento	Tipo	Concepto	Importe	Estado
1	Consultores SA	30-12-2019	Ingreso	Nómina	+1268.00 €	Efectuado
2	Carrefour	01-01-2020	Recibo	Supermercado	-128.56 €	Efectuado
3	El Corte Inglés	03-01-2020	Recibo	Chaqueta hombre L-XL	-99.99 €	Efectuado
4	El Corte Inglés	03-01-2020	Recibo	Pantalón hombre M-L	-48.50 €	Efectuado

Para hacer esto basta con crear una estructura de datos a modo de un DIV contenedor que dentro posea dos DIV, uno para el título y otro para los datos. Dentro de este último DIV , deberemos establecer tantos DIV como filas y columnas se dispongan, es decir, deberemos establecer tantos DIV como celdas tenga el grid. Algo como:

```
<div class="grid">
  <div class="caption">EJEMPLO DE TABLA CON FLEXBOX CSS</div>

  <div class="row">
    <div class="col">ID</div>
    <div class="col">Empresa</div>
    <div class="col">F. Movimiento</div>
    <div class="col">Tipo</div>
    <div class="col">Concepto</div>
    <div class="col">Importe</div>
    <div class="col">Estado</div>

    <div class="col">1</div>
    <div class="col">Consultores SA</div>
    <div class="col">30-12-2019</div>
    <div class="col">Ingreso</div>
    <div class="col">Nómina</div>
    <div class="col">+1268.00 €</div>
    <div class="col">Efectuado</div>

    <div class="col">2</div>
    ...
  </div>
</div>
```

Después, sólo necesitaremos definir el sistema de grid a la clase `.ROW` y unos cuantos estilos adicionales:

## Código CSS

```
.grid .row { display: grid; grid-template-columns: repeat(7, 1fr);
grid-template-rows: minmax(max-content, 1fr);
padding: 1px 1px; border: 1px solid #ccc; }
.grid .caption { display: block; text-align: center; font-weight: 600;
background: #000; color: #fff; }
.grid .col { border: 1px solid #ccc; margin: 1px; padding: 0 5px; }
.grid .row .col:nth-child(-n+7) { font-weight: 600; }
```

## 6.4 PRACTICA Y JUEGA

### Juego: Flexbox Froggy



Se trata de aprender a manejar diseños basados en cajas flexibles mediante el uso de sus propiedades y dispone de 24 niveles explicados detalladamente en español. Se puede acceder desde la dirección <https://flexboxfroggy.com/#es>.

### Juego: CSS Grid Garden



Se trata de aprender a manejar diseños basados en cuadrículas mediante el uso de sus propiedades y dispone de 28 niveles explicados detalladamente en español. Se puede acceder desde la dirección <https://cssgridgarden.com/#es>.

### Maquetación CSS – Parte 4

### Código QR

Corrige el CSS solicitado y juega a cambiar el HTML para realizar tu propia personalización.

<https://codepen.io/pefc/pen/XWPMQPa>

