

Tema 1: HTML y CSS

Parte 2: CSS

CSS



BLOQUE 1: Desarrollo web en cliente - HTML y CSS

Módulo: Desarrollo de Interfaces Web

Ciclo: Desarrollo de Aplicaciones Web

Curso: 2023 – 2024

Profesora: Rosa Medina

Índice

1. Cómo usar CSS.....	5
1.1. Enlace CSS externo (etiqueta link).....	5
1.2. Enlace CSS interno (etiqueta style).....	6
1.3. Estilos en línea (atributo style).....	6
2. Estructura de CSS.....	7
2.1. Compatibilidad.....	8
2.2. Concepto de herencia.....	8
2.2.1. Valores especiales de herencia.....	10
2.3. Concepto de cascada en CSS.....	10
3. Estilos CSS básicos.....	11
3.1. Tipografía.....	11
3.1.1. Propiedad: color.....	11
3.1.2. Propiedad: font-family.....	12
3.1.3. Propiedad: font-size.....	13
3.1.4. Propiedad: font-style.....	13
3.1.5. Propiedad: font-weight.....	14
3.1.6. Propiedad: font-variant.....	14
fuentes externas.....	14
3.1.7. Textos y alineaciones.....	16
propiedad: letter-spacing.....	16
propiedad: word-spacing.....	16
propiedad: line-height.....	16
propiedad: text-indent.....	16
propiedad: white-space.....	17
propiedad: tab-size.....	17
propiedad: direction.....	17
propiedad: text-align.....	17
propiedad: text-justify.....	17
propiedad: text-overflow.....	17
propiedad: vertical-align.....	18
propiedad: text-decoration.....	18
propiedad: text-transform.....	18
propiedad: text-shadow y box-shadow.....	18
3.2. Fondo (background).....	19
3.2.1. Propiedad: background-color.....	19
3.2.2. Propiedad: background-image.....	19
3.2.3. Propiedad: background-repeat.....	20
3.2.4. Propiedad: background-position.....	20
3.2.5. Propiedad: background-attachment.....	20
4. Los selectores.....	21
4.1. Seleccionar por ID (valor único).....	21
4.2. Seleccionar por clases.....	22
4.3. Prioridad de selectores.....	23
4.4. Selecciones mixtas.....	24
4.5. Selectores CSS avanzados.....	24
4.5.1. Agrupación de selectores.....	25
4.5.2. Selector descendiente.....	26

4.5.3. Selector hijo.....	26
4.5.4. Selector hermano adyacente.....	27
4.5.5. Selector hermano general.....	27
4.5.6. Selector universal.....	28
5. Modelo de cajas.....	29
5.1. Dimensiones (ancho y alto).....	29
5.2. Propiedades de los bordes.....	30
5.2.1. Propiedad border-style.....	30
5.2.2. Propiedad border-color.....	31
5.2.3. Propiedad border-width.....	31
5.2.4. Propiedad border-radius.....	31
5.2.5. Diferentes bordes a un mismo elemento.....	31
5.3. Propiedades de los bordes - outline.....	32
5.4. Propiedades de los márgenes.....	33
5.4.1. Centrar un elemento.....	33
5.5. Propiedades de los márgenes internos.....	34
6. Propiedades de las tablas.....	35
6.1. Propiedad table-layout.....	35
6.2. Propiedad border-collapse.....	35
6.3. Propiedad border-spacing.....	36
6.4. Propiedad empty-cells.....	36
7. Propiedades de visibilidad.....	37
7.1. Propiedad cursor.....	37
7.2. Propiedad display.....	38
7.3. Propiedad visibility.....	38
7.4. Propiedad float.....	38
7.5. Propiedad clear.....	39
8. Posicionamiento de los elementos – CSS2.....	40
8.1. Propiedad position.....	40
8.2. Propiedad overflow.....	41
8.3. Propiedad z-index.....	41
8.4. Propiedad left.....	41
8.5. Propiedad right.....	42
8.6. Propiedad top.....	42
8.7. Propiedad bottom.....	42
9. Posicionamiento con rejilla - grid.....	42
9.1. Grid y flexbox.....	42
9.2. La rejilla – filas y columnas.....	43
9.3. Espacios/márgenes/huecos en grid.....	44
9.4. Posición de los elementos en el grid.....	45
10. Propiedades de las listas.....	46
10.1. Propiedad list-style-type.....	46
10.2. Propiedad list-style-image.....	47
10.3. Propiedad list-style-position.....	47
10.4. Listas y menú.....	49
11. Selectores avanzados.....	50
11.1. Pseudoclases.....	50
11.1.1. Pseudoclases de enlaces.....	50
11.1.2. Pseudoclases de ratón.....	50

11.1.3. Pseudoclases de interacción.....	50
11.1.4. Pseudoclases de activación.....	51
11.1.5. Pseudocalses de validación.....	51
11.1.6. Pseudocalses de negación.....	52
11.1.7. Pseudoclases para elementos según su posición.....	52
11.1.8. Pseudoclases para elementos del mismo tipo.....	53
11.1.9. Otras pseudoclases.....	53
11.2. Pseudoelementos.....	53
11.2.1. Pseudoelemento de generación de contenido.....	54
11.2.2. Pseudoelemento – Primera letra y/o primera línea.....	54
11.3. Atributos.....	54
12. Cascada en CSS.....	56
12.1. !important.....	57
12.2. Especificidad de los selectores.....	57
12.3. Orden.....	58
13. Diseño adaptativo y responsive.....	58
13.1. Diseño adaptativo vs responsive.....	59
13.2. Media queries.....	59
13.3. ¿Cómo pruebo cómo se ve con el navegador diferentes resoluciones?.....	62
14. Diseño responsive - flexbox.....	65
14.1. Configuración de la caja contenedora.....	66
14.2. Configuración de las cajas internas.....	66
15. Prefijos CSS.....	69
16. Otras propiedades avanzadas de CSS.....	70

Hasta el momento hemos visto cómo organizar un documento HTML, a lo largo de este tema vamos a ver qué son las **hojas de estilo en cascada** (*Cascading Style Sheets*) o **CSS**. Debemos tener claro que una página web en realidad es un documento de texto escrito en código HTML y que el código CSS es el encargado de dar forma, color, posición (entre otras características) a una página web.

El término CSS es el lenguaje de estilos utilizado para establecer cómo se van a presentar los documentos HTML o XML. Se denomina hoja de estilo en cascada porque las reglas se aplican de arriba a abajo.

1. Cómo usar CSS

Antes de comenzar a ver las reglas de CSS debemos conocer las diferentes formas que podemos utilizar para incluir estilos en nuestros HTML.

A continuación, vamos a ver las tres formas que hay para incluir nuestros estilos, siendo la más común (y recomendada) la primera, y la última la menos recomendada principalmente en proyectos grandes.

1.1. Enlace CSS externo (etiqueta link)

En la cabecera de nuestro HTML (dentro de la etiqueta `<head>`) debemos incluir una etiqueta `<link>` con la que haremos referencia a el archivo CSS que vamos a utilizar, en el atributo `href` indicaremos la ubicación de esa hoja de estilos. Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi primera hoja de estilo</title>
    <!-- Incluyo mi archivo CSS -->
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Aprendiendo CSS</h1>
    <p>Hola este mensaje estará en rojo</p>
  </body>
</html>
```

Al incluir la etiqueta `link` con el `href` a nuestra hoja de estilos, estamos indicando al navegador que deben aplicar los estilos del archivo `estilo.css`. Antiguamente también se añadía el atributo `type="text/css"`, pero desde HTML5 ya no es necesario, y si se incluye es por el tema de compatibilidad con navegadores muy antiguos.

1.2. Enlace CSS interno (etiqueta style)

Otra forma habitual de incluir estilos en una página web es añadiéndolos directamente en el documento HTML mediante la etiqueta `<style>` dentro de la etiqueta `<head>`:

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi primera hoja de estilo</title>
    <!-- Incluyo mi CSS -->
    <style>
      p {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Aprendiendo CSS</h1>
    <p>Hola este mensaje estará en rojo</p>
  </body>
</html>

```

El ejemplo anterior estamos incluyendo nuestro CSS mediante la etiqueta `<style>`, pero con el objetivo de poder reutilizar nuestros estilos a posteriori es recomendable que se utilice mediante la etiqueta `<link>` como vimos en el punto anterior ([Enlace CSS externo](#)).

1.3. Estilos en línea (atributo style)

La tercera y última forma que tenemos para aplicar estilos en un documento HTML es hacerlo directamente mediante el atributo `style` a la etiqueta a la que le queremos aplicarle el estilo.

Ejemplo:

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Mi primera hoja de estilo</title>
  </head>
  <body>
    <h1>Aprendiendo CSS</h1>
    <p style="color: red;">Hola este mensaje estará en rojo</p>
  </body>
</html>

```

Al igual que con el método anterior (etiqueta `<style>`), no se recomienda utilizar este método.

¡IMPORTANTE! Si modificamos una misma propiedad desde varios sitios (hoja de estilos / etiqueta `style`/ atributo `style`, ¿cuál se aplicará?. Se aplica el estilo de más prioridad, siendo de mayor a menor: atributo `style` → etiqueta `style` → Hoja CSS. Ejemplo:

Archivo index.html	Archivo estilo.css
<pre> <!DOCTYPE html> </pre>	<pre> p { </pre>

```

<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Mi primera hoja de estilo</title>
    <style>
      p {
        color: green;
      }
    </style>
  </head>
  <body>
    <p style="color: red;">Hola este mensaje estará en
color...???</p>
  </body>
</html>

```

2. Estructura de CSS

Para definir un estilo CSS, se utilizan reglas. Una regla debe tener el siguiente formato:

```
selector { propiedad: valor; }
```

El **selector** es el elemento HTML al que vamos a aplicarle un estilo. Por ejemplo, si usamos el selector `p` estamos seleccionando todos los párrafos de nuestra página. Entre llaves, debemos poner las **declaraciones** de los estilos como puede ser el color de la letra, el borde de una tabla, etc. Cada declaración estará formada por una propiedad y un o varios valores asociados a esa propiedad, y cada declaración irá separada por punto y coma.

```

p {
  color: red; /* Con esta declaración todos los párrafos tendrán el color de letra rojo */
}

```

Nota: Un comentario en CSS comienzan con `/*` y finalizan con `*/` como en la mayoría de los lenguajes de programación.

Si quisiéramos aplicar más estilos al párrafo como puede ser que el texto esté en cursiva, sería:

```

p {
  color: red; /* Con esta declaración todos los párrafos tendrán el color de letra rojo */
  font-style: italic; /* Debemos separar cada propiedad por punto y coma */
}

```

Nuestra hoja de estilos tendrá tantas reglas como queramos, si por ejemplo queremos además poner el heading `<h1>` de color azul sería:

```

p {
  color: red; /* Con esta declaración todos los párrafos tendrán el color de letra rojo */
  font-style: italic; /* Debemos separar cada propiedad por punto y coma */
}
h1 {

```

```
color: blue; /* Pongo de azul el contenido de la etiqueta h1*/
}
```

Es importante seguir como en programación una serie de buenas prácticas a la hora de escribir nuestro código para que sea más fácil de leer:

- Una **regla por línea**
- Usa la **indentación**
- Después de cada regla usa el **punto y coma**, aunque el último dentro de un bloque sea opcional.

2.1. Compatibilidad

Cada día hay más funcionalidades de CSS disponibles, pero no todas ellas son soportadas por todos los navegadores. Con el objetivo de poder saber si alguna propiedad CSS, elemento HTML o funcionalidad de JavaScript hay disponible en X navegador, podemos hacer uso de herramientas como [Can I Use](#), dentro de esta página tendremos:

- En **rojo** las características que no están soportadas en dicha versión
- En **amarillo/marrón** aquellas características que no están soportadas por completo en dicha versión.
- En **Verde** las características que sí están soportadas.

Además, si aparece un cuadradito amarillo significa que se puede usar con prefijos y si tiene una bandera es que dicha característica se puede activar mediante flags.



2.2. Concepto de herencia

El término CSS es *Cascading Style Sheets* (Hoja de Estilos en Cascada). El concepto de herencia y de cascada son las dos características más importantes de CSS. En primer lugar, debemos saber que algunas propiedades de CSS se heredan desde los elementos padres a los hijos modificando el valor que tuvieran por defecto.

Por ejemplo, imaginemos que le aplicamos a la etiqueta `<body>` un color, haciendo que todos los textos que estén dentro de la etiqueta `<body>` de nuestro HTML se le aplique dicho color. Por ejemplo:

```
body {
  color: blue;
}
```

En este ejemplo estamos haciendo que todos los textos que estén dentro de la etiqueta `<body>` estén de color azul. Si tenemos una etiqueta `<div>` con texto en su interior y no tenemos ningún estilo de `color` para dicho elemento, el texto también aparecerá de color azul. Esto es debido a que la propiedad `color` en el caso de no darle ningún valor específico pasará a **heredar** el valor que tenga su elemento padre.


```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Aplicando estilos</title>
    <!-- Pongo el CSS aquí para que os resulte más fácil verlo en los apuntes, ->
    pero recordad que deberíamos ponerlo en un archivo externo para separar HTML de CSS -->
    <style>
      body {
        color: blue;
      }
    </style>
  </head>
  <body>
    <h1>El título estará en azul</h1>
    <div id="container">
      <p>Este párrafo también estará en azul</p>
    </div>
  </body>
</html>

```

El título estará en azul

Este párrafo también estará en azul

Debemos tener en cuenta que no todas las propiedades se heredan, hay propiedades como son los bordes que no se heredan. Veamos qué pasa si añadimos al ejemplo anterior un borde:

```

body {
  color: blue;
  border: 2px solid red;
}

```

El título estará en azul

Este párrafo también estará en azul

Si la propiedad `border` se heredara todos los elementos que son hijos de `<h1>` (`<body>`, `<div>`, `<p>`) deberían tener un borde de color rojo, pero si nos fijamos no es así, sólo el `<body>` de nuestra web tiene esa propiedad. La herencia por tanto no ocurre con todas las propiedades CSS sino con algunas como `color` o `font`, a medida que vayamos trabajando con CSS nos iremos familiarizando con esto.

2.2.1. Valores especiales de herencia

Además de los valores de cada propiedad CSS, podemos aplicar unos valores especiales comunes a todas las propiedades existentes de forma que modifiquemos el comportamiento de la herencia para dicha propiedad. Estos valores son:

- **inherit**: hereda el valor que tiene la misma propiedad CSS que su elemento padre
- **initial**: establece el valor inicial que tenía la propiedad CSS inicialmente.
- **unset**: combinación de las dos anteriores, es decir, hereda la propiedad del elemento padre, y en el caso de no existir se le aplica su valor inicial.

Por ejemplo, si queremos que en el ejemplo anterior al `<div>` se le aplique una propiedad que no se le aplica por defecto (propiedad `border` del elemento padre que es el `<body>`):

```

body {

```

El título estará en azul

Este párrafo también estará en azul

```

color: blue;
border: 2px solid red;
}
div {
border: inherit;
}

```

2.3. Concepto de cascada en CSS

Uno de los conceptos más importantes de CSS es el de **cascada**. Vamos a ver con un ejemplo qué significa el término de cascada en CSS. Imaginemos que tenemos dentro de nuestra hoja de estilos dos veces el mismo selector (en este caso el `<div>`) y para cada uno de ellos le hemos dado un valor diferente a la propiedad `color`.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aplicando estilos</title>
    <!-- Pongo el CSS aquí para que os resulte más fácil verlo en los apuntes, ->
        pero recordad que deberíamos ponerlo en un archivo externo para separar HTML de CSS -->
    <style>
      div {
        color: blue;
        border: 2px solid red;
      }

      div {
        color: orange;
        margin: 2px;
      }
    </style>
  </head>
  <body>
    <h1>El título de mi web</h1>
    <div id="container">
      <p>¿De qué color estará el párrafo azul o naranja?</p>
    </div>
  </body>
</html>

```

El título de mi web

¿De qué color estará el párrafo azul o naranja?

¿Que ha ocurrido? ¿Qué propiedad se va a aplicar? Si nos fijamos, tenemos dos propiedades `color` aplicadas al mismo elemento `<div>` y están al mismo nivel. En este caso la respuesta es fácil: siempre se va a aplicar la última regla definida, la cual va a mezclar las dos propiedades (`border` y `margin`) y sobrecribir aquella que esté repetida (`color`). Es como si en nuestra hoja de estilos tuviéramos lo siguiente:

```

div {
  color: orange; /* Se aplica el último valor que se encuentra en el caso de tener 2
                  propiedades con valores distintos */
}

```

```
border: 2px solid red;
margin: 2px;
}
```

Este no es el único caso donde podemos ver el concepto de cascada de CSS, más adelante seguiremos viendo cómo saber qué propiedad se aplicará a un mismo elemento si accedemos a él por medio de la etiqueta, de su atributo `id` y/o de una `class`. Antes debemos saber cómo aplicar estilos mediante selectores.

3. Estilos CSS básicos

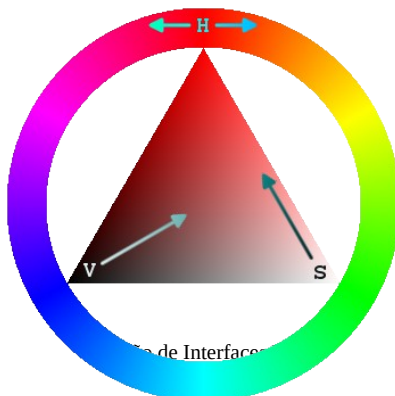
3.1. Tipografía

La elección de una tipografía de nuestro sitio web puede resultar bastante importante a la hora de que un usuario valore nuestro sitio web (color, tamaño letra, espaciado entre letras, interlineado, etc.). En este tipo vamos a ver cómo podemos modificar desde CSS estas propiedades.

3.1.1. Propiedad: color

La propiedad `color` nos permite modificar el color del texto. Los valores que puede tomar esta propiedad son:

- **Palabras clave de color:** Hay más de 140 palabras para indicar los colores como: `red`, `blue`, `orange`, ...
- **Esquema RGB:** Las siglas RGB (rojo, verde y azul) se expresan cada uno de los colores con un valor de 0 a 255. Antiguamente estos valores estaban entre paréntesis separados por comas. Ejemplo: `color: rgb(0 255 0);`, pero hoy en día se pueden separar por espacios. Ejemplo: `color: rgb(0 255 0);`
- **Esquema RGB hexadecimal:** es el más utilizado, consiste en dar el valor de cada color del RGB en hexadecimal comenzando con el carácter `#`. De modo que el ejemplo anterior sería: `color: #00FF00;` Podemos utilizar el formato abreviado cuando **los pares de cifras** son idénticos. En el caso anterior podríamos resumirlo en: `color: #0Fo;`
- **Esquema HSL:** Las siglas HSL significan (color, saturación y brillo). La primera de ellas es el matiz del color (cifra de 0 a 360 grados), los siguientes valores son la saturación y el brillo indicados en porcentajes, estos valores como el esquema RGB puede ir separado por comas o espacios. Ejemplo: `color: hsl(180deg, 15%, 85%);`



Además de estos, podemos especificar también el canal alfa para establecer una transparencia parcial en determinados colores, el valor alfa se da en porcentajes de 0% a 100% o numérico de 0 al 1 con decimales. Este valor alfa se da si utilizamos rgb, hsl, o con el formato hexadecimal, y para ello pasaríamos a utilizar → rgba(...), hsla(...), ó #00FF0080;

Hay dos herramientas muy utilizadas para seleccionar el color [HSL color picker](#) y [Adobe Color](#)

3.1.2. Propiedad: font-family

Con esta propiedad podemos seleccionar el tipo de tipografía que queremos (Verdana, Arial, etc.) La forma de indicar qué familia tipográfica queremos será escribiendo su nombre, en el caso de ser un nombre compuesto por ejemplo 'Times New Roman', deberemos ponerlo entre comillas simples. Si el usuario final no tiene instalada la tipografía verá la letra con la tipografía por defecto. Una buena práctica es poner varias tipografías separadas por coma para que en el caso de no visualizar una que tenga otra opción como alternativa.

Veamos un ejemplo:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Aplicando estilos</title>
    <!-- Pongo el CSS aquí para que os resulte más fácil verlo en los apuntes, ->
      pero recordad que deberíamos ponerlo en un archivo externo para separar HTML de CSS -->
    <style>
      p{
        font-family: Georgia, 'Times New Roman', Times, serif;
      }
    </style>
  </head>
  <body>
    <h1>Título de mi web</h1>
    <p>Mi perra me está dando con el hocico en el brazo</p>
  </body>
</html>
```

Es importante elegir una fuente genérica al final del listado de nuestras fuentes para asegurarnos una fuente aceptable ya que no hay nada que nos asegure que el usuario tenga un tipo de fuente en concreto instalada. Las fuentes seguras son: serif, sans-serif, cursive, fantasy y monospace.

3.1.3. Propiedad: font-size

Con esta propiedad podemos especificar el tamaño que va a tener la fuente, los valores que puede tener son:

- **Medidas absolutas:** `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, por defecto el valor es `medium`.
- **Medidas relativas:** representan un tamaño más grande/pequeño que el actual: `larger`, `smaller`
- **Medida específica:** Indicamos los píxeles/porcentajes/pt/otra con el tamaño en concreto a usar. El punto (pt) es el que se recomienda para fijar el tamaño de la fuente.

Ejemplo:

```
p{
  font-size: 12pt;
}
```

3.1.4. Propiedad: font-style

Con esta propiedad podemos aplicar ciertos estilos muy utilizados para maquetar textos. Los valores que puede tomar son:

- **normal:** valor por defecto
- **italic:** Cursiva, estilo con ligera inclinación a la derecha, es la versión cursiva de la fuente creada por el diseñador
- **oblique:** oblicua, es igual al anterior salvo que esta inclinación se realiza de forma artificial. Muchas veces no se aprecia diferencia entre `italic` y `oblique`

3.1.5. Propiedad: font-weight

Con esta propiedad podemos indicar el grosor de la fuente. Los valores que podemos darle son:

- **Valores absolutos:** `normal` o `bold`. Por defecto el valor es normal.
- **Valores relativos:** `bolder` o `lighter`, dependiendo de si la queremos más gruesa o más fina a la actual.
- **Valor numérico:** un número de 100(menos gruesa) a 900(más gruesa), normalmente el valor es de 100 en 100.



3.1.6. Propiedad: font-variant

Permite convertir las letras minúsculas en mayúsculas dentro de un texto. La diferencia es que el tamaño de las que antes estaban en minúsculas es más pequeño aún siendo mayúsculas, y esto crea un efecto curioso. Valores:

- **normal:** Sin cambios.
- **small-caps:** Aplica el efecto mencionado anteriormente, así el texto "Me llamo Juan", quedaría como "ME LLAMO JUAN " más o menos.

Por ejemplo, a un elemento con la `id="destacado"` le vamos a aplicar una fuente "Arial", de tamaño 24 puntos, en cursiva y negrita. Y además, que las letras minúsculas aparezcan como mayúsculas más pequeñas de lo normal.

```
#destacado{
  font-family: Arial;
  font-size: 24pt;
  font-style: italic;
  font-variant: small-caps;
  font-weight: bold;
}
```

ME LLAMO JUAN

fuentes externas

En lugar de depender de los tipos de letra instalados en el sistema del usuario, podemos cargar el/los archivo/s con nuestra propia fuente y obligar al navegador a utilizarla. Estos archivos deberán estar en formato True Type Font (ttf), Vectorial (svg), o OpenType Font (otf). En internet hay disponibles miles de fuentes así para descargar. Para ello, ponemos la regla `@font-face` para descargar una fuente de una web y cargarla en el navegador para utilizarla aunque no esté instalada en nuestro sistema. Esta regla se suele poner al principio del fichero CSS.

Para declarar un tipo de fuente a partir de un archivo debemos darle un nombre, e indicar el archivo (puede ser local o en una url), a partir del cual se obtiene la fuente:

```
@font-face {
  font-family: SuperFuente;
  src: url('fonts/superfnt.ttf');
}
```

Si disponemos de archivos para negrita o cursiva, por ejemplo, de la misma fuente, se pueden englobar siempre que los declaremos bajo el mismo nombre de fuente, e indiquemos el estilo con `font-weight` y/o `font-style`.

```
@font-face {
  /* Ahora le indicamos que debe buscar la versión en negrita de la fuente anterior*/
  font-family: SuperFuente;
  src: url('fonts/superfnt-bold.ttf');
  font-weight: bold;
}
```

Finalmente, para usar este fuente que hemos declarado simplemente, hay que aplicar la propiedad `font-family` con el nombre personalizado al elemento, por ejemplo:

```
p {
  font-family: SuperFuente;
}
```

Otra posibilidad, es utilizar las fuentes externas de [Google](#). Cuando elegimos la fuente a utilizar nos indica el código html que debemos introducir dentro de nuestra etiqueta `<head>` y el nombre de la fuente a utilizar en la propiedad `font-family`.

Por ejemplo, imaginad que queremos utilizar la fuente [Road Rage](#), debemos pulsar sobre “Select this style” y nos aparecerá qué debemos copiar en el head para poderla utilizar, en este caso es:

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Road+Rage&display=swap"
rel="stylesheet">
```

Además, nos pone el código CSS que debemos utilizar para seleccionar dicha fuente:

```
font-family: 'Road Rage', cursive;
```

Si esto lo aplicamos a un párrafo, el resultado sería:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Fuente externa</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Road+Rage&display=swap" rel="stylesheet">
    <style>
      p {
        font-family: 'Road Rage', cursive;
      }
    </style>
  </head>
  <body>
    <p>Este párrafo tiene una fuente Road Rage</p>
  </body>
</html>
```

Este párrafo tiene una fuente Road Rage

3.1.7. Textos y alineaciones

Con CSS los estilos que podemos dar en relación al espaciado, o interlineado son los siguientes:

propiedad: letter-spacing

Esta propiedad te permite establecer un espaciado **entre cada letra** de un texto (interletraje o tracking). Los valores que puede tener son: *normal* o un tamaño específico. Si el valor es un número negativo tendremos más unidas las letras, si el valor es positivo estarán más separadas.



propiedad: word-spacing

Esta propiedad te permite establecer un espacio que hay **entre una palabra y otra** en un texto. Los valores que puede tener son: `normal` o un tamaño específico.

propiedad: line-height

Esta propiedad te permite establecer el **interlineado** (altura entre líneas). Los valores que puede tener son: `normal` o un tamaño específico.

propiedad: text-indent

Esta propiedad te permite establecer las **sangrías** (indentación del texto). El valor será un número, por defecto es 0. Si el valor es negativo desplazará el texto a la izquierda, en el caso de ser positivo se desplaza a la derecha.

propiedad: white-space

Esta propiedad te permite establecer un cierto comportamiento de los espaciados. Los valores más habituales que suele tener son: `normal` | `nowrap` | `pre` | `pre-line` | `pre-wrap`. Por defecto el valor es `normal` (transforma múltiples espacios en blanco en un solo espacio consecutivo), pero las otras opciones nos permite:

- **normal**: Los espacios se transforman en uno solo, se ajusta al contenedor.
- **nowrap**: Los espacios se transforman en uno solo, ignora los saltos de línea.
- **Pre**: Respeta los espacios, pero ignora los saltos de línea
- **pre-wrap**: Respeta los espacios y se ajusta al contenedor
- **pre-line**: Respeta literalmente los espacios y suprime los espacios del final. Se ajusta al contenedor.

La diferencia entre `pre-wrap` y `pre-line` es que este último sí respeta literalmente los espacios que están antes del texto, mientras que si sobran después del texto los elimina.

propiedad: tab-size

Esta propiedad te permite establecer el ancho de las tabulaciones (espacio o tamaño). Los valores que puede tener son: un número o un tamaño específico.

propiedad: direction

Esta propiedad te permite establecer la dirección del texto, es decir, si el contenido, texto y otros elementos fluyen de izquierda a derecha o viceversa. Los valores que puede tener son: `ltr` o `rtl`, es decir, de izquierda a derecha o de derecha a izquierda respectivamente.

propiedad: text-align

Esta propiedad te permite justificar el texto. Los valores que puede tener son: `left` | `center` | `right` | `justify`, es decir, alineamos el texto a la izquierda, centro, derecha o justificamos el texto respectivamente, de la misma forma que podemos hacer con el Writer/Word.

propiedad: text-justify

Esta propiedad te permite establecer el método con el que el navegador justificará los textos. Los valores que puede tener son: `auto` | `inter-word` | `inter-character` | `none`, es decir, automáticamente el navegador elige cómo justificar, que sea una justificación entre palabras, una justificación entre caracteres, o que no haya justificación.

propiedad: text-overflow

Esta propiedad te permite modificar el comportamiento que tiene por defecto el navegador cuando un texto no cabe y se desborda. Los valores que puede tener son: `clip` | `ellipsis` | `texto`, es decir, desbordar el contenedor (valor por defecto), muestra puntos suspensivos cuando el texto no cabe, o mostrar un texto cuando no cabe y no queremos que muestre los puntos suspensivos.

propiedad: vertical-align

Del mismo modo que la propiedad `text-align`, esta propiedad nos permite alinear un elemento de forma vertical. Los valores que puede tener entre otros valores son:

- **top**: el elemento se posiciona en la parte superior del elemento padre.
- **middle**: el elemento se posiciona en la mitad del elemento padre
- **bottom**: el elemento se posiciona en la parte inferior del elemento padre.
- **text-top**: el elemento se posiciona en la parte superior del texto padre.
- **text-bottom**: el elemento se posiciona en la parte inferior del texto padre.

Cuando queramos alinear bloques de contenido o crear estructuras de diseño utilizaremos **flexbox** como veremos más adelante.

propiedad: text-decoration

Esta propiedad te permite subrayar el texto (`underline`), subrayar por encima del texto(`overline`) y tachar el texto(`line-through`) si se utiliza el valor `none` se elimina cualquiera de

los formatos anteriores. Esta propiedad se utiliza mucho para eliminar por ejemplo el subrayado que por defecto aparece con la etiqueta de enlace/hipervínculo.

propiedad: text-transform

Esta propiedad te permite convertir textos a mayúsculas (*uppercase*) o minúsculas (*lowercase*). También nos permite capitalizar el texto, es decir poner sólo la primera letra en mayúsculas independientemente de cómo esté escrito (*capitalize*).

propiedad: text-shadow y box-shadow

Con estas dos propiedades podemos dar sombras a los textos (*text-shadow*) o a otros elementos como cajas o contenedores (*box-shadow*).

text-shadow aplica una sombra a un contenido de texto, los valores que puede tener son:

- Valores posiciónX y posiciónY: permite indicar las coordenadas X e Y donde pondremos la sombra con respecto al texto o contenedor original.
- Valor size, para el caso del *text-shadow* es el valor del blur, es decir el radio que de desenfoque de la sombra medido en px, em, etc. Cuanto más pequeño sea el valor menos difuminada estará la sombra, cuanto más alto sea más borrosa se verá. En el caso del *box-shadow* se le puede dar dos valores de size, el primero será como el de *text-shadow* y el segundo hace referencia al factor de crecimiento de la sombra. Aumentando este valor podemos hacer que la sombra crezca hacia los lados.
- Valor color: podemos indicarle el valor que queremos que tome la sombra.

Por otro lado, la propiedad *box-shadow* aplica una sombra a una caja o contenedor. Puede tener los mismos valores que *text-shadow* junto con *inset* que sirve para indicarle que la sombra la queremos interna en lugar de externa que es la que aparece por defecto.

Veamos un ejemplo:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aplicando estilos</title>
    <!-- Pongo el CSS aquí para que os resulte más fácil verlo en los apuntes, ->
    pero recordad que deberíamos ponerlo en un archivo externo para separar HTML de CSS -->
    <style>
      p {
        text-shadow: 5px -5px 2px #444;
      }
      div {
        box-shadow: 2px 2px 10px #666;
      }
    </style>
  </head>
  <body>
    <h1>Título de mi web</h1>
    <p>Mi perra me está dando con el hocico en el brazo</p>
```

Título de mi web

Mi perra me está dando con el hocico en el brazo

normal

```
<div>normal</div>
</body>
</html>
```

3.2. Fondo (background)

A lo largo de este punto vamos a ver cómo podemos poner un color de fondo o una imagen.

3.2.1. Propiedad: background-color

La propiedad `color` establecía el color del texto, la propiedad `background-color` establece el color de fondo del elemento. Todas las propiedades CSS donde existen valores color, pueden tomar los mismos valores que vimos en el punto de la [propiedad color](#).

```
body {
  background-color: red;
}
```

3.2.2. Propiedad: background-image

Si queremos utilizar una imagen de fondo utilizaremos la propiedad `background-image` y en el valor será `url()` y dentro de los paréntesis la ubicación de la imagen.

```
body {
  background-image: url(imagen.jpg);
}
```

Una vez que hemos establecido la imagen que vamos a utilizar podremos personalizar cómo hacer que se repita (`background-repeat`), dónde ponerla (`background-position`), o cómo queremos que se comporte al hacer scroll (`background-attachment`).

3.2.3. Propiedad: background-repeat

Indica si la imagen se debe repetir o no hasta ocupar todo el ancho y alto del elemento (por defecto no se repite) sus valores pueden ser:

- **no-repeat**: No repetirá la imagen, la mostrará sólo 1 vez, aunque no ocupe todo el alto o el ancho del elemento.
- **repeat**: Repetirá la imagen, tanto horizontal como verticalmente, hasta ocupar todo el ancho y alto del elemento (valor por defecto).
- **repeat-x**: Repetirá la imagen, pero sólo en el eje x, hasta ocupar todo el ancho del elemento.
- **repeat-y**: Repetirá la imagen, pero sólo en el eje y, hasta ocupar todo el alto del elemento.

3.2.4. Propiedad: background-position

Indica la posición vertical y horizontal donde se debe situar la imagen dentro del elemento. Sus valores se pueden definir como:

- **posX posY**: Posición que ocupan en el eje vertical (y) y el horizontal (x) del elemento.
- **posX** puede tomar los valores left (izquierda), center (centrado), right (derecha), y % (siendo 0% el extremo derecho, y 100% el extremo izquierdo), coordX (un número de 0 hasta el ancho en píxeles del elemento).
- **posY** puede tomar los valores top (arriba), center (centrado), bottom (abajo), y % (siendo 0% la parte más alta, y 100% la más baja del elemento), coordY (un número de 0 hasta el alto en píxeles del elemento).

3.2.5. Propiedad: background-attachment

Indica si la imagen se debe mover al hacer scroll en la página o quedarse fija donde está. Sus valores son:

- **scroll**: La imagen se comportaría como esperamos, es decir, al hacer scroll se quedaría en su sitio junto con el resto de contenidos (opción por defecto si no ponemos esta propiedad).
- **fixed**: La imagen bajaría y subiría con nosotros al hacer el scroll (hasta que ya no fuera visible el elemento que la contiene).

Ejemplo de celda a la que se le asigna una imagen de fondo que no debe repetirse y que además se situará en la parte superior de la misma y centrada en el eje X:

```
td{
  background-image: url(imagen.jpg);
  background-repeat: no-repeat;
  background-position: top center;
}
```

Es posible establecer todas estas propiedades anteriores en una sola regla de CSS a modo de atajo, donde aunque no es estricto se recomienda utilizar el siguiente orden:

```
background: <color> <imagen> <repeat> <attachment> <position>;
```

4. Los selectores

Hasta el momento hemos visto cómo podemos aplicar un estilo a una etiqueta de nuestro código HTML. sin embargo, lo que estamos haciendo haciendo es seleccionando todos los elementos del documento que sean dicha etiqueta y aplicando el estilo a todos.

Por ejemplo, en el primer estilo que aplicamos al párrafo dándole el valor rojo (**color: red**):

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Aplicando estilos</title>
    <!-- Pongo el CSS aquí para que os resulte más fácil verlo en los apuntes, ->
      pero recordad que deberíamos ponerlo en un archivo externo para separar HTML de CSS -->
    <style>
      p {
```

```

        color: red;
    }
</style>
</head>
<body>
    <h1>El título de mi web</h1>
    <div id="container">
        <p>Primer párrafo de mi web</p>
        <p>Este es otro párrafo</p>
    </div>
</body>
</html>

```

En este ejemplo por tanto estamos diciéndole al navegador que todas las etiquetas `<p>` que encuentre en nuestra página le aplique el color rojo.

4.1. Seleccionar por ID (valor único)

Hemos visto la forma más básica de seleccionar elementos en CSS utilizando como selector la etiqueta a la que le queremos aplicar el estilo. En el tema pasado, vimos que todas las etiquetas HTML pueden tener un atributo `id`, ese valor va a ser un **valor único** y por tanto no podremos encontrar dentro de nuestro HTML dos etiquetas con el mismo valor `id` (el `id` actúa como nuestro DNI, no podemos encontrar dos personas con el mismo DNI). Por ejemplo:

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Aplicando estilos</title>
  </head>
  <body>
    <div id="header">
      <h1>El título de mi web</h1>
    </div>
    <div id="container">
      <p>Primer párrafo de mi web</p>
      <p>Este es otro párrafo</p>
    </div>
  </body>
</html>

```

En el ejemplo anterior, tenemos dos elementos `<div>` cada uno con un `id` diferente, el primero de ellos tiene el valor `header`, y el segundo `container`. No podríamos poner otro `<div>` dentro de nuestro HTML con uno de esos dos valores ya que ya hay una etiqueta con ese valor.

A día de hoy no se suelen utilizar muchos ids (desde HTML5), pero son atributos válidos que se emplean para designar una zona que sabemos que no se va a repetir e identificarla como única. Si al código anterior HTML, le aplicamos el siguiente estilo:

```

#header {
  color: red;
}

```

El título de mi web

Primer párrafo de mi web

Este es otro párrafo

En el anterior ejemplo, estamos dando un estilo accediendo con el selector `id`. Para ello utilizamos el símbolo `#`, es decir, estamos dándole el color rojo al texto que contenga el atributo: `id="header"`.

4.2. Seleccionar por clases

Cuando tenemos propiedades CSS que se repiten y no podemos asignarle a varias etiquetas distintas el mismo identificador utilizamos las clases de CSS. Toda etiqueta HTML puede tener el atributo `class`, la diferencia con respecto a los ids es que estos no tienen porqué ser únicas y por tanto podemos encontrarnos varias clases iguales en un mismo documento HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Aplicando estilos</title>
  <style>
    .btn-page {
      background-color: aqua;
      color: white;
    }
    .btn-cancel {
      background-color: maroon;
      color: white;
    }
  </style>
</head>
<body>
  <form action="#" method="post">
    <button class="btn-page">Atrás</button>
    <button class="btn-page">Siguiente</button>
    <button class="btn-cancel">Cancelar</button>
  </form>
</body>
</html>
```

En el ejemplo anterior tenemos 3 botones, los dos primeros son botones para navegar en una página, y para establecerle el mismo estilo le hemos puesto una clase llamada `btn-page`. Por otro lado tenemos el botón Cancelar que le hemos puesto un estilo diferente (clase `btn-cancel`). Al usar `class` nos ha permitido con un sólo selector utilizar un mismo estilo en dos botones y podremos reutilizarlos si queremos aplicar ese estilo en más elementos de nuestro HTML.

En CSS si queremos acceder a una determinada clase, se accede mediante el carácter punto(.): Ejemplo: `.btn-page`, de esa forma estamos accediendo a todos los elementos de nuestro HTML que tengan el atributo: `class="btn-page"`

Además, en las clases podemos indicar en nuestro CSS qué tipo de elemento se trata, pudiendo utilizar ese mismo nombre de la clase en otros elementos HTML y darle un estilo diferente. Por ejemplo:

```
button.btn-page { /* Asignamos estilo sólo a los elementos button que tengan la clase btn-page */
  background-color: aqua;
  color: white;
```

```

}
p.btn-page{ /* Asignamos estilo sólo a los elementos p que tengan la clase btn-page */
  background-color: maroon;
  color: white;
}

```

4.3. Prioridad de selectores

¡IMPORTANTE! ¿Qué pasa si aplicamos varias veces la misma propiedad a un mismo elemento?. Hay un orden de prioridad establecido de mayor a menor: id, clase, elemento), ejemplo:

```
<p id="par1" class="cont">Soy un texto de color</p>
```

Si en el documento CSS pusiéramos:

```

p {
  color: red;
}
#par1 {
  color: green;
}
.cont {
  color: blue;
}

```

Los 3 afectan al elemento `<p>` de arriba, pero el color de su texto sería verde (green), ya que el identificador por id es el que mayor prioridad tiene. Si este se eliminase, se le aplicaría el color azul (blue), ya que el selector de clase es el siguiente que más prioridad tiene.

Si se modifica más de 1 vez la misma propiedad para un mismo elemento, pero ambas modificaciones tienen la misma prioridad, se le aplicará la última especificada (de ahí el nombre de Hojas de Estilo en Cascada). Por ejemplo:

```

.cont {
  color: red;
  color: blue;
}

```

Se aplicaría un color azul porque es la última modificación y ambas tienen la misma preferencia (clase).

4.4. Selecciones mixtas

Es posible utilizar varias clases en un mismo elemento HTML, para ello cada clase irá separada por espacios dentro del atributo `class`. De esta forma, a dicho elemento se le aplicará los estilos de cada una de las clases que tenga asignada. Por ejemplo:

```
<button class="btn blue btn-page">Atrás</button>
```

Por último, indicar que si queremos ser más específicos y aplicar un cierto estilo sólo a los elementos HTML que tengan todas las clases indicadas se realiza poniendo en nuestro CSS cada clase de forma consecutiva:

```
button.btn.blue.btn-page {
  ... /* Este estilo se aplicará sólo a aquellos elementos HTML button que tengan una clase btn,
      una clase blue y una clase btn-page. Si falla alguna de ellas no se aplicará */
}
```

4.5. Selectores CSS avanzados

Además de los selectores visto en los puntos hay una serie de métodos para seleccionar elementos dependiendo de la estructura del documento HTML, son los denominados combinadores CSS:

Nombre	Símbolo	Ejemplo	Significado
Agrupación de selectores	,	p, a, div { ... }	Se aplican estilos a varios elementos
Selector descendiente		#page div { ... }	Se aplican estilos a elementos dentro de otros
Selector hijo	>	#page > div { ... }	Se aplican estilos a elementos hijos directos
Selector hermano adyacente	+	div + div { ... }	Se aplican estilos a elementos que siguen a otros
Selector hermano general	~	div ~ div { ... }	Se aplican estilos a elementos al mismo nivel
Selector universal	*	#page * { ... }	Se aplican estilos a todos los elementos

4.5.1. Agrupación de selectores

Si queremos asignar el mismo estilo CSS a varios selectores distintos, en lugar de crear el mismo bloque para cada selector, una buena práctica es simplificar nuestro CSS agrupandolo mediante una coma (,).

```
button.btn-page {
  background-color: aqua;
  color: white;
}
p {
  background-color: aqua;
  color: white;
}
```

De esta manera, el código anterior lo podríamos simplificar e indicarle al navegador que estas propiedades CSS se las aplique al conjunto de selectores indicado.

```
button.btn-page, p {
  background-color: aqua;
  color: white;
}
```



```
}
```

Una buena práctica para mejorar la legibilidad del código CSS es poner cada selector en lugar de uno detrás de otro, ponerlo cada uno en una línea separada. Esto hacemos que en grandes documentos CSS sea más fácil de leer y por tanto de revisar/modificar/mantener nuestro CSS.

```
button.btn-page,
p {
  background-color: aqua;
  color: white;
}
```

4.5.2.Selector descendiente

El selector descendiente es una forma de denominar a ciertos elementos que están dentro de otros. La forma de reflejar esto en CSS es mediante un espacio, por ejemplo:

```
div.container div{
  background-color: gray;
}
```

En el ejemplo anterior, estamos aplicando un fondo a todos los elementos `<div>` que estén dentro de otro `<div>` con `class container` (sean hijos, nietos, etc.) De esta forma, si hay un `<div>` fuera con la clase `container` no se le aplicarán los estilos. Ejemplo:

```
...
<main>
  <div class="container">
    <div class="articulo"> <!-- Se aplica -->
      <h2>Economía</h2>
      <p>...</p>
    </div>
    <div class="articulo"> <!-- Se aplica -->
      <h2>Deporte</h2>
      <p>...</p>
    </div>
    <article>
      <div> <!-- Se aplica aun siendo "nieto" u no hijo directo -->
        <p>Otro ejemplo</p>
      </div>
    </article>
  </div> <!-- Cerramos el div con class container -->
  <div> <!-- No se aplica -->
    <p>Aquí no se aplicará</p>
  </div>
</main>
<footer>
  <div> <!-- No aplica -->
    <p>Página realizada por...</p>
  </div>
</footer>
```

4.5.3.Selector hijo

El selector hijo lo utilizaremos cuando en lugar de querer seleccionar todos los elementos descendientes como hacíamos antes, seleccionar sólo los descendientes directos (sólo hijos, no habrá nietos ni sucesivos) del primer elemento especificado. Para ello utilizaremos el símbolo `>`.

```
div.container > div{
  background-color: gray;
}
```

Utilizando este selector, en el HTML anterior estaremos seleccionado sólo los `<div>` que en este caso tienen la clase `articulo`.

4.5.4.Selector hermano adyacente

Si queremos acceder únicamente a aquellos elementos que están directamente a continuación del elemento especificado (elemento hermano), utilizaremos el símbolo `+`, con esto seleccionaremos los hermanos que están seguidos el uno del otro, es decir, al mismo nivel.

```
div.container div+div {
  background-color: gray;
}
```

Con el ejemplo anterior de HTML y este CSS estamos seleccionado el `div` con la clase `articulo` que contiene información de deporte, ya que estamos seleccionando todos los `divs` que están a continuación de `<div class="container">` `<div>`. Si tuviéramos a continuación otro `div` de tecnología también se seleccionaría.

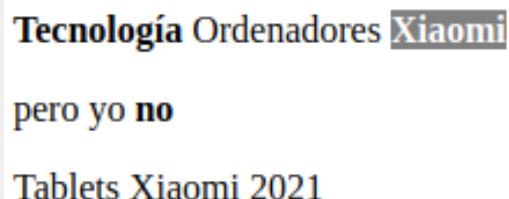
```
<div class="container">
  <div class="articulo"> <!-- Se aplica -->
    <h2>Economía</h2>
    <p>...</p>
  </div>
  <div class="articulo"> <!-- Se aplica -->
    <h2>Deporte</h2>
    <p>...</p>
  </div>
  <div class="articulo"> <!-- Se aplica -->
    <h2>Tecnología</h2>
    <p>...</p>
  </div>
</div>
<article>
  <div> <!-- No se aplica al no ser hermano -->
    <p>Otro ejemplo</p>
  </div>
</article>
</div>
```

Es importante tener en cuenta que el primer `div` (el de economía) no se selecciona al estar usándolo de base.

4.5.5.Selector hermano general

Si por lo contrario lo que queremos es seleccionar todos los hermanos en general sin tener en cuenta si son adyacentes o no, tendremos que utilizar el símbolo ~. Veamos un ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Aplicando estilos</title>
  <style>
    div.container strong~strong {
      background-color: gray;
      color: white;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="articulo">
      <strong>Tecnología</strong>
      <span>Ordenadores</span>
      <strong>Xiaomi</strong> <!-- Yo soy su hermano -->
      <p>pero yo <strong>no</strong></p>
    </div>
    <div class="articulo">
      <span>Tablets</span>
      <span>Xiaomi</span>
      <span>2021</span>
    </div>
  </div>
</body>
</html>
```



Tecnología Ordenadores **Xiaomi**
pero yo no
Tablets Xiaomi 2021

Si nos fijamos, no es necesario que el elemento `` se encuentre adyacente al primero, sino que basta con que sean hermanos en el mismo nivel.

4.5.6.Selector universal

El selector universal se representa con un `*` y es la forma de aplicar ciertos estilos en todos y cada uno de los elementos HTML correspondientes. Por ejemplo, si utilizando el HTML anterior tenemos el siguiente código CSS:

```
div.container * {
  background-color: gray;
}
```

Estaríamos seleccionando todos los elementos que se encuentran dentro de la clase `.container`, que en este caso son los dos articulo.

El selector universal se suele utilizar para resetear propiedades de un documento como son los distintos márgenes que algunos navegadores ponen.

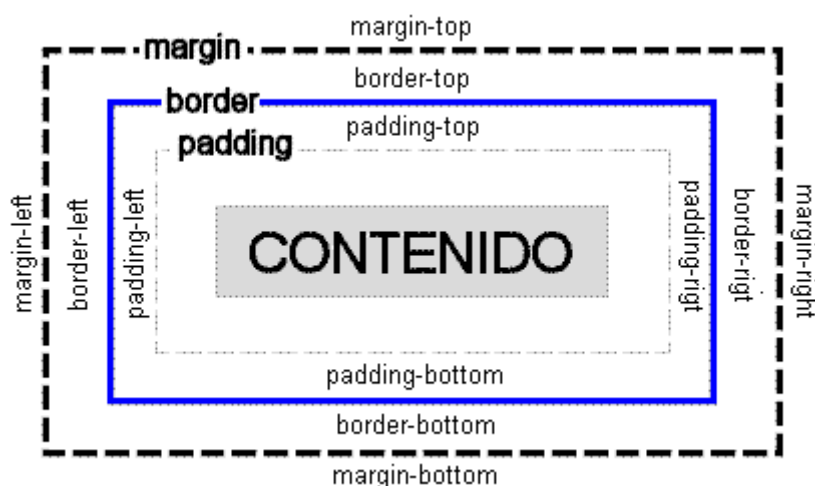
```
* { /* Eliminamos los márgenes internos y externos de todos los elementos del HTML */  
  margin: 0;  
  padding: 0;  
}
```

5. Modelo de cajas

El modelo de cajas en CSS es uno de los puntos más complejos de dominar, hace referencia a las distintas partes de un elemento HTML con sus dimensiones.

El modelo de cajas se compone de:

- **Contenido** HTML.
- **Dimensiones** de la caja: alto y ancho
- **Borde** (border) es el límite que separa el interior del exterior del elemento, el borde es un recuadro que encierra por tanto el contenido
- **Margen interior** (padding): espacio o margen interno entre el borde y el contenido
- **Margen** (margin): espacio que hay entre el borde de la caja y otras cajas adyacentes, es decir, la parte exterior del elemento (fuera del borde).



Cada una de las partes del modelo de cajas de CSS se puede modificar cambiando sus dimensiones, colores, etc. consiguiendo que cada elemento HTML tenga su propio borde, margen, relleno y contenido.

5.1. Dimensiones (ancho y alto)

Si queremos darle un tamaño específico a los distintos elementos de un documento HTML utilizaremos las propiedades ancho(**width**) y alto(**height**). Los valores serán a elección del programador con las medidas normalmente de porcentajes o píxeles. En el caso de darle el valor auto, estamos indicándole al navegador que sea él quien le de el tamaño apropiado dependiendo de su contenido (valor por defecto).

```
div {
  width: 100%;
  height: 200px;
  background-color: pink;
}
```

¡OJO! Cuando estamos asignando un alto/ancho a un elemento, ese tamaño no incluye el tamaño que pueda tener su borde o padding, el tamaño total del elemento será la suma de todos ellos.

Además de estos valores, tenemos la posibilidad de indicarle un máximo/mínimo con las propiedades: `min-width`, `max-width`, `min-height`, `max-height`. De modo que en lugar de darle un tamaño fijo establecemos unos rangos donde el ancho/alto podría variar entre esos valores. Es importante tener en cuenta que los mínimos(`min-width`, `min-height`) por defecto tienen un valor de 0, mientras que los máximos(`max-width`, `max-height`) tienen el valor `none`.

Si por error le damos al ancho(`width`) o alto(`height`) un valor superior que su máximo, el tamaño será entre el mínimo y el máximo. Ejemplo:

```
div {
  width: 100px;
  height: 200px;
  background-color: pink;
  max-width: 80px;
}
```

El ancho será entre 0 (valor por defecto) y el 80px.

Ejercicio 1: Define 2 párrafos con un ancho mínimo de 150px y máximo de 300px, una altura mínima de 60px y máxima de 100px. Ponles a ambos un borde también.

- El primer párrafo añádele poco contenido (4 letras valen) de tal forma que no llegue al ancho ni alto mínimo.
- El segundo párrafo añádele mucho contenido, de tal forma que sobrepase el ancho y alto máximos y se salga por los bordes.

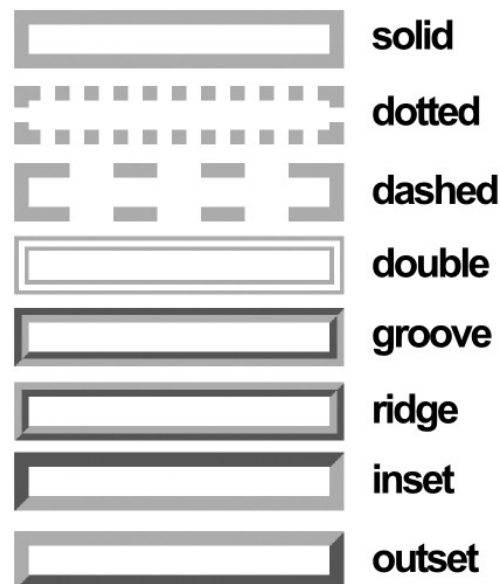
5.2. Propiedades de los bordes

Con estas [propiedades](#) podemos definir las características que tendrá el borde de un elemento HTML, como por ejemplo, el grosor, el color, el estilo, etc.

5.2.1. Propiedad border-style

Define el estilo del borde que rodea a un elemento. Se pueden definir los estilos de cada lado del borde por separado en lugar de usar `border-style`, utilizando `border-bottom-style` (abajo), `border-top-style` (arriba), `border-left-style` (izquierda) y `border-right-style` (derecha). Los valores a asignar son:

- `none`: No hay borde.
- `hidden`: El borde está pero no será visible, aunque se tendrá en cuenta para posicionar elemento en la página.
- `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`: Son los diferentes estilos de borde. Podéis ver (y modificar) los efectos en directo, en [esta página](#).



5.2.2. Propiedad border-color

Define el color del borde que rodea al elemento. Se pueden definir los colores de cada lado del borde por separado en lugar de usar `border-color`, utilizando `border-bottom-`

`color`(abajo), `border-top-color` (arriba), `border-left-color` (izquierda) y `border-right-color` (derecha). Los valores a asignar son:

- **transparent**: como su nombre indica, el color sería invisible.
- Un **valor** para el color como el visto en la propiedad `background-color` por ejemplo.

5.2.3.Propiedad border-width

Define el grosor del borde. Como hemos explicado anteriormente, si solo queremos influir sobre uno de los lados podemos utilizar `border-bottom-width`(abajo), `border-top-width` (arriba), `border-left-width` (izquierda) y `border-right-width` (derecha). Los valores a asignar son:

- **thin**: borde fino.
- **medium**: borde normal.
- **thick**: borde grueso.
- **longitud**: Tamaño en píxeles del borde. Un valor de 5px establecería un borde de 5 píxeles de grosor.

5.2.4.Propiedad border-radius

Indica el radio de curvatura que tendrá la esquina del borde. Se mide en píxeles. Cuanta más cantidad, más redondeado se verá. Se puede indicar la curvatura de cada borde de forma individual usando los valores:

- **border-top-left-radius**: esquina superior izquierda.
- **border-top-right-radius**: esquina superior derecha.
- **border-bottom-left-radius**: esquina inferior izquierda.
- **border-bottom-right-radius**: esquina inferior derecha.

Antes de que fuera estándar las versiones de los navegadores que usaban motor Gecko (Mozilla), o Webkit (Safari, Chrome,), ponían el prefijo `-moz-` y `-webkit-` respectivamente, delante de esta propiedad.

5.2.5.Diferentes bordes a un mismo elemento

Hasta el momento hemos visto cómo aplicar el mismo borde a un elemento, o cómo aplicar a cada lado un borde diferente. Tenemos otra posibilidad más y es especificar uno, dos, tres o cuatro parámetros dependiendo de lo que queremos hacer.

Vamos a ver un ejemplo con el color. Si a la propiedad `border-color` le asignamos solo un color, por ejemplo: `border-color: red;`. Estamos dando el mismo color a los cuatro bordes de dicho elemento. Si en lugar de dar un valor damos dos, por ejemplo: `border-color: red blue;`, estamos diciendo que el primer valor(`red`) será el valor para arriba/abajo y el segundo(`blue`) para la izquierda/derecha. En el caso de darle tres valores: `border-color: red blue yellow;`, estamos dando el primer valor (`red`) arriba, el segundo valor(`blue`) a la izquierda y derecha y el tercer valor (`yellow`) abajo. Para el caso en el que le demos los cuatro valores: `border-color: red blue yellow pink;` será: arriba(`red`), derecha(`blue`), abajo(`yellow`), izquierda(`pink`). Si nos fijamos va en sentido de las agujas del reloj. Lo mismo ocurriría si en lugar de estar modificando la propiedad `border-color`, fuese `border-width` o `border-style`.

Al igual que con otras propiedades CSS, podemos utilizar la propiedad `border`, con la que podemos hacer un resumen e indicarle las múltiples propiedades individuales de la siguiente forma: `border: <width> <style> <color>;`

Ejemplo:

```
div {
  border: 2px solid pink;
}
```

Nota: Lo mismo sería para: `border-top: <width> <style> <color>;` `border-right: <width> <style> <color>;` `border-bottom: <width> <style> <color>;` y `border-left: <width> <style> <color>;`

Ejercicio 2: Define 2 párrafos:

- El primero tendrá el borde uniforme (los 4 lados iguales). Debe tener un grosor normal, un color oscuro y el estilo `ridge`.
- El segundo tendrá cada uno de sus bordes diferentes (y además un color de fondo amarillo). Los lados tendrán los colores verde, azul, rojo y marrón (da igual a que lado le asignéis cada color, mientras sean diferentes). Cada uno tendrá un estilo diferente a vuestra elección. El grosor de los bordes de arriba y abajo será de 5 píxeles, mientras que el grosor de los laterales será de 3 píxeles.

5.3. Propiedades de los bordes - outline

Esta propiedad es como un doble borde, de forma que podemos añadir otro borde al elemento por fuera del que asignábamos con la propiedad `border`. **¡Importante!** A diferencia que los bordes, esta línea divisoria (`outline`), por defecto no ocupa espacio y no tiene porqué tener una forma rectangular.

Como se puede ver en este [ejemplo](#) donde hay cuatro párrafos, el primero con un `outline` negro a rayas, el segundo con puntos rojos, el tercero liso de color amarillo y el último `ridge` de color rosa.

Las propiedades son las mismas que para definir el borde, la única diferencia es que no diferenciaremos entre arriba, abajo, izquierda y derecha, sino que las propiedades del "outline" se aplicarán a los 4 lados.

- **outline-style:** igual que border-style.
- **outline-width:** igual que border-width.
- **outline-color:** igual que border-color.

Podríamos definir un elemento con un borde de 4px, continuo, de color verde y rojo:

```
#doble_borde {
  border: 2px solid green;
  outline: 2px solid red;
}
```

En CSS3, también está la propiedad `outline-offset` que nos permite ampliar el desplazamiento/espacio interior dentro del perfil del elemento.

Ejercicio 3: Define un párrafo con un borde normal en el que cada lado tenga un color, y que además tenga un `outline` de color negro, fino, y con puntos.

5.4. Propiedades de los márgenes

El [margen](#) es el espacio exterior de un elemento, es decir, el espacio entre el borde de un elemento y su exterior. Las propiedades son:

- [margin-top](#), [margin-bottom](#), [margin-left](#), [margin-right](#): Distancia del elemento respecto al que se encuentra en su parte de (arriba, abajo, izquierda y derecha en este orden). Se especifica de 3 formas:
 - [auto](#): El navegador será el que decida cual será el margen a aplicar.
 - porcentaje (%): El margen será un porcentaje del espacio disponible para el elemento (si es margen de arriba o abajo, será el espacio del eje vertical, y si es el margen de derecha o izquierda, el espacio del eje horizontal).
 - distancia (píxeles->px, centímetros->cm,...): distancia fija con respecto al margen superior, inferior, derecho o izquierdo.

5.4.1. Centrar un elemento

Para centrar un elemento en el eje horizontal, basta con poner el margen derecho e izquierdo automáticos, de esta forma, la mayoría de los navegadores pondrán el margen derecho e izquierdo del mismo tamaño centrando el elemento. Esto no funciona para centrarlo verticalmente.

Elemento centrado en el eje horizontal.

```
.centro_h {
  margin-left: auto;
  margin-right: auto;
}
```

Si queremos poner el valor auto a todos los márgenes, no hace falta que pongamos los valores a cada 1 de los 4 ejes. Esta solución le aplica el valor auto a los 4 márgenes.

```
.centro_h {
  margin: auto;
}
```

Ejercicio 4: Crea un documento HTML con:

1. Crea un párrafo con un margen superior de 50 píxeles, y un margen izquierdo de 100 píxeles.
2. Crea un párrafo con un ancho de 200 píxeles, color de fondo verde, y céntralo
3. Centra una imagen (comprobarás que una imagen no se considera un bloque, sino que es igual que si fuera texto a nivel de alineamiento).

5.5. Propiedades de los márgenes internos

El [padding](#) es un concepto parecido al margen que hemos estado viendo, pero con una diferencia fundamental. En este caso, hablamos de la distancia entre el borde del elemento y su contenido (en el caso del margen hablábamos de distancia entre el borde del elemento el elemento que lo contenía, o los elementos de alrededor).

Los valores que se pueden dar son similares a los que vimos al asignar márgenes:

- **padding-top, padding-bottom, padding-left, padding-right:** Distancia del contenido respecto a los bordes (arriba, abajo, izquierda y derecha en este orden). Se especifica de 2 formas:
 - **porcentaje (%)**: El **padding** será un porcentaje del tamaño del elemento (si es **padding** de arriba o abajo, será el tamaño en altura, y si es el **padding** de derecha o izquierda, el tamaño en anchura).
 - **distancia** (píxeles->px, centímetros->cm,...): distancia fija del contenido con respecto al borde superior, inferior, derecho o izquierdo.

Si queremos poner un **padding** de 20 píxeles a cada lado, y de 25 píxeles en la parte superior e inferior, se puede hacer como en el ejemplo:

```
#padding1 {
  padding-top: 25px;
  padding-bottom: 25px;
  padding-left: 20px;
  padding-right: 20px;
}
```

Otra forma de hacer lo mismo (se puede aplicar a **margin** y a muchas propiedades de CSS, pero tenemos que saber lo que hacemos) es aplicar todos los valores al atributo **padding** separados por espacios (estos cuatro valores se aplicarían en orden a los lados arriba, derecha, abajo e izquierda), quedando así:

```
#padding1 {
  padding: 25px 20px 25px 20px;
}
```

Si sólo le ponemos un valor a **padding**, aplica ese valor a los 4 lados, y si le ponemos 2 valores, el primero lo aplica a los lados de arriba y abajo, y el segundo, a derecha e izquierda. Con lo cual lo podemos simplificar al máximo así:

```
#padding1 {
  padding: 25px 20px;
}
```

Ejercicio 5: Crea una tabla con un ancho de 800 píxeles (**width: 800px** en CSS) que tenga 4 celdas, 2 arriba y 2 abajo. Todas las celdas deben tener un borde (puede ser el mismo para todas), y la misma anchura y altura (un 50% en cada propiedad, **width** y **height**).

- En la primera celda debe haber un **padding** de 25 píxeles en los 4 lados.
- En la segunda un **padding** de 40 píxeles en el lado izquierdo, y 20 píxeles en el resto.
- En la tercera, un **padding** del 10% a derecha e izquierda.

- En la cuarta un `padding` de 50 píxeles con el borde superior.
- Introduce texto suficiente en todas las celdas para que se vea el efecto.

6. Propiedades de las tablas

Estas propiedades son específicas para el manejo de las tablas, de hecho, están pensadas para usarse sólo con el elemento `<table>` de HTML. Gran parte de ellas tienen que ver con los bordes de tabla y celdas como veremos:

6.1. Propiedad `table-layout`

Indica si el tamaño horizontal de la tabla y las celdas debe adaptarse al tamaño del contenido, o por el contrario, debe mantenerse fijo aunque los contenidos no quepan dentro del tamaño establecido. Sus valores son:

- **automatic:** Es el valor por defecto si no pusiéramos esta propiedad. Las celdas y la tabla pueden tener un tamaño definido por el usuario. Pero si el contenido no cabe (bien sea por insertar una imagen o un elemento que ocupe más que la celda, o porque introducimos mucho texto sin espacios en blanco para que pueda hacer saltos de línea), entonces aumentará el tamaño de la celda (y con ello el de la tabla y hablamos de tamaños fijos en las celdas y no porcentajes) para adaptarse al nuevo contenido.
- **fixed:** El tamaño de las celdas y la tabla siempre será el mismo. Si insertamos un contenido que no quepa, se saldrá por los bordes, quedando un efecto un poco feo como se puede ver [aquí](#).

6.2. Propiedad `border-collapse`

Esta propiedad indica si los bordes de las celdas deben estar pegados entre sí o separados como en HTML por defecto. Sus valores son:

- **separate:** Opción por defecto. Cada celda tendrá su propio borde separado del resto (se debe definir un borde para la tabla y las celdas en este caso), y a su vez el borde que rodea la tabla también estará separado y será independiente. del resto
- **collapse:** Con esta opción se crea un efecto muy parecido al atributo `cellspacing="0"` de HTML, ya que los bordes de la celda quedan "pegados" entre sí, y a su vez con el borde de la tabla. En este caso no hace falta poner un borde a la tabla, pero si le ponemos un borde diferente al de las celdas, se dibujará en el contorno el de más grosor (el de las celdas o el de la tabla) . A igual grosor, se dibujará encima el borde de las celdas.

6.3. Propiedad `border-spacing`

Esta propiedad sólo tiene sentido cuando tenemos separados los bordes de las celdas, es decir, `border-collapse: separate;`. Esta propiedad indica la distancia a la que estarán unos bordes de otros (unas celdas de otras).

- **Tamaño** en píxeles u otra medida soportada por CSS del espacio entre celdas. Si ponemos sólo un valor, por ejemplo: `border-spacing: 5px;`, indicamos tanto el espacio

horizontal como el vertical. Si queremos poner un espacio diferente para el eje vertical y para el horizontal, se deben poner los 2 valores seguidos y separados por un espacio. Ejemplo: `border-spacing: 5px 2px;` (5 píxeles horizontalmente y 2 verticalmente).

6.4. Propiedad empty-cells

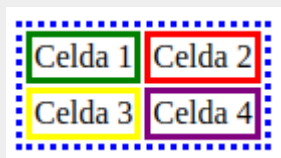
Esta propiedad (sólo para bordes separados como la anterior) indica que cuando alguna celda no tenga contenido, si deberá mostrarla (dibujar los bordes de la misma) o no.

- **show:** Aunque la celda esté vacía dibujará sus bordes.
- **hide:** Este es el valor por defecto. Si una celda de la tabla no tiene ningún tipo de contenido, no dibujará sus bordes. Puede crear un efecto bastante bueno, si no dibujamos ningún borde para tabla, sino que únicamente las celdas tienen bordes, en caso contrario, quedará un efecto mayor de "vacío".

Podemos asignar un borde que rodee la tabla con un diseño, y un diseño independiente para cada borde de celda, siempre que tengamos los bordes separados.

Código HTML

```
...
<table class="tabla1">
  <tr>
    <td class="td1">Celda 1</td>
    <td class="td2">Celda 2</td>
  </tr>
  <tr>
    <td class="td3">Celda 3</td>
    <td class="td4">Celda 4</td>
  </tr>
</table>
...
```



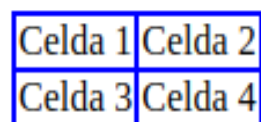
Código CSS

```
.tabla1 {
  border-collapse: separate;
  border-style: dotted;
  border-color: blue;
}
.td1 {
  border-style: solid;
  border-color: green;
}
.td2 {
  border-style: solid;
  border-color: red;
}
.td3 {
  border-style: solid;
  border-color: yellow;
}
.td4 {
  border-style: solid;
  border-color: purple;
}
```

Otra opción es definir un estilo único para los bordes de las celdas, apropiado para tener los bordes juntos:

Código HTML

```
...
<table class="tabla2">
  <tr>
    <td class="celda">Celda 1</td>
    <td class="celda">Celda 2</td>
  </tr>
  <tr>
    <td class="celda">Celda 3</td>
    <td class="celda">Celda 4</td>
  </tr>
</table>
...
```



Código CSS

```
.tabla2 {
  border-collapse: collapse;
}
.celda {
  border-style: solid;
  border-width: 2px;
}
```

```

<td class="celda">Celda 3</td>
<td class="celda">Celda 4</td>
</tr>
</table>
...
border-color: blue;
}

```

Ejercicio 6: Crea un documento HTML con la siguiente tabla:

- Crea una tabla con 2 filas y 3 celdas por fila como mínimo. Esta tabla tendrá los bordes separados, y un mismo estilo de borde (al gusto de cada uno) para todas sus celdas (la tabla no tendrá borde). Indica también en las propiedades de la tabla que no muestre las celdas vacías. Deja alguna de las celdas sin contenido para ver el resultado.

Ejercicio 7: Crea un documento HTML con la siguiente tabla:

- Crea una tabla de 500 píxeles de ancho y con 2 celdas en 1 fila. La primera celda ocupará el 20% del ancho de la tabla (`width: 20%;` en CSS). En las propiedades de la tabla debe aparecer la propiedad que indica que el tamaño horizontal de las celdas sea fijo. Ponle un borde a las celdas (los bordes deben aparecer juntos). Finalmente inserta una imagen(no muy grande) y texto (sin espacios en blanco) que no quepan en la celda, para ver el efecto.

7. Propiedades de visibilidad

Estas propiedades especifican como se visualiza un elemento y el posicionamiento de este en la página (aunque de esta última parte hablaremos más en profundidad en la siguiente sección).

7.1. Propiedad cursor

Establece el tipo de cursor del ratón que aparecerá cuando pasemos el ratón por encima del elemento. La lista es la siguiente y para ver el efecto lo mejor es probarlos, o ir al siguiente [enlace](#)

- auto, crosshair, default, pointer, move, e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize, text, wait, help.

7.2. Propiedad display

Indica como se representa un elemento en la página (o incluso si se ocultará). Podemos cambiar la propiedad de un elemento con esto de tal forma que un tipo de elemento como `` se pudiese comportar como si fuera otro (`<div>` por ejemplo):

- **none:** El elemento no se dibujará.
- **block:** El elemento será de tipo "bloque" (como `<div>` o `<p>` por ejemplo), teniendo entre otras cosas, un salto de línea al principio y al final.
- **inline:** Lo contrario que lo anterior. El elemento se alinearán con el resto del texto o contenido eliminándose los saltos de línea al principio y final.
- **inline-block:** Es una mezcla de los dos anteriores, Permite que un elemento de bloque se comporte como si fuera un elemento en línea pero conservando algunas propiedades de bloque como la posibilidad de definir su anchura/altura.

- **hidden**: similar a `none`, pero el elemento sigue conservando su espacio (no se ve).
- **grid**: para definir rejillas CSS con su filas y columnas. Lo veremos en el punto de [Posicionamiento de elementos con grid](#)
- **list-item**: El elemento se comportará como si fuera un elemento tipo `` (elemento de una lista).
- **table**: El elemento se comportará como si fuera un elemento `<table>`
- **table-row**: El elemento se comportará como si fuera un elemento `<tr>`
- **table-cell**: El elemento se comportará como si fuera un elemento `<td>`
- Y muchos otros....

7.3. Propiedad visibility

Indica si un elemento debe estar oculto (invisible) o permanecer visible.

- **visible**: Valor por defecto. El elemento se mostrará.
- **hidden**: El elemento se ocultará (útil si queremos hacer un mapa oculto de la web para que los buscadores puedan navegar mejor por nuestra página y que los usuarios no lo vean).

7.4. Propiedad float

Establece un elemento flotante e indica en la posición donde se situará dentro de la página (Este elemento se situará en algunas ocasiones encima de otros elemento fijos que hubieran dentro de su elemento padre, así que cuidado).

- **left**: Posicionará el elemento a la izquierda dentro del elemento padre (en cuanto encuentra un hueco lo suficientemente grande para que quepa, lo coloca ahí)
- **right**: Posicionará el elemento a la derecha dentro del elemento padre (en cuanto encuentra un hueco lo suficientemente grande para que quepa, lo coloca ahí)
- **none**: Por defecto, dibujará el contenido donde aparezca en el código HTML.

Ejemplos de como alinear una imagen a la derecha [dentro de un párrafo](#), o [fuera de el](#).

Ejemplo de como alinear la primera letra de un [párrafo a la izquierda](#).

Ejemplo de como hacer un [menú a partir de una lista](#), quitándole la propiedad de salto de línea a los elementos `` con (`display: inline;`), y poniendo los enlaces `<a>` como elementos flotantes alineados a la izquierda (cambia `left` por `right` para ver el efecto)

7.5. Propiedad clear

Establece si a los lados de un elemento (normalmente hablamos de un elemento flotante) podrán aparecer otros elementos flotantes.

- **left**: Con esta propiedad, aunque haya espacio a la izquierda, no se podrá situar ningún elemento flotante a la izquierda de este.
- **right**: Lo mismo pero para el lado derecho.
- **both**: Impide que otro elemento flotante se sitúe en ninguno de los lados del actual.

- **none:** Permite que los elementos se puedan situar a izquierda y derecha del actual.

Estructura ejemplo con elementos flotantes (copiarla, primero sólo el HTML y luego añadirle el estilo para ver el efecto):

Código HTML

```
...
<div id="pagina">
  <div id="cabecera">Soy una cabecera</div>
  <div id="contenedor">
    <div id="parteiz">
      <div class="menu">Soy el primer menú</div>
      <div class="menu">Soy el segundo menú</div>
    </div>
    <div id="parteder">
      <div class="menu2">Soy un menú a la derecha</div>
    </div>
    <div class="contenido">Soy el contenido de la página</div>
  </div>
</div>
...
```

Código CSS

```
#pagina {
  width: 900px;
  margin: auto;
}
#cabecera {
  float: left;
  width: 100%;
  background-color: red;
  text-align: center;
  min-height: 50px;
}
#contenedor {
  float: left;
  width: 100%;
  background-color: #FFF7A2;
  text-align: center;
  min-height: 50px;
}
#parteiz {
  float: left;
  width: 150px;
  margin-right: 25px;
}
#parteder {
  float: right;
  width: 150px;
}
.menu {
  width: 150px;
  background-color: #B1FFEC;
  margin-top: 25px;
}
.contenido {
  margin-left: 175px;
  width: 550px;
  background-color: #BDC9FF;
  margin-top: 25px;
  min-height: 400px;
}
.menu2 {
  width: 150px;
  background-color: #B1FFEC;
  margin-top: 25px;
}
```

Ejercicio 8: Realiza una estructura similar a la del ejemplo de arriba, pero más sencilla.

Pon una cabecera, un menú a la izquierda, un cuerpo donde meterás el contenido a la derecha de ese menú, y abajo del todo un pie de página. Ponles un color de fondo a cada uno para diferenciarlos, por ejemplo.

8. Posicionamiento de los elementos – CSS2

8.1. Propiedad position

Indica si un elemento estará posicionado en un lugar estático, relativo, absoluto o fijo dentro de la página

- **static:** Valor por defecto. El elemento se posiciona donde la página lo dibuje de forma natural. En este caso no sirven las propiedades `top`, `bottom`, `right` y `left` que veremos más adelante.
- **relative:** Parecido al anterior pero si que deja que desplacemos al elemento con las propiedades `top`, `bottom`, `right` y `left` que veremos más adelante.
- **absolute:** El elemento se posiciona en una capa superior, por encima del resto de elementos. Al contrario que en la posición `relative`, el hueco que ocupaba pasa a estar libre (como si se fuera a otra dimensión). Para moverlo tenemos 2 opciones:
 - **top, left, right, bottom:** El elemento se mueve a partir de las coordenadas 0,0 de la página web. Es muy parecido al efecto que se da con la posición `fixed` pero el elemento no bajaría con el scroll.
 - **margin:** Si lo movemos con `margin`, el elemento se moverá a partir de la posición que ocuparía originalmente. Esto lo hace mucho más parecido al comportamiento de `relative`, pero sin dejar ocupado el hueco original.
- **fixed:** El elemento se posiciona en las coordenadas indicadas con las propiedades `top`, `bottom`, `right` y `left`, dentro de la ventana del navegador. Aunque hagamos scroll, el elemento se mantiene en las coordenadas indicadas.

8.2. Propiedad overflow

Indica lo que pasa cuando el contenido de un elemento se sale de sus bordes (Lo hemos experimentado antes, pero ahora podemos controlar lo que pasa).

- **visible:** El contenido se saldrá de los bordes, pero por encima de estos (se verá la par del contenido que quede fuera del elemento).
- **hidden:** El contenido se saldrá pero por debajo del borde y no será visible la parte que se salga.
- **scroll:** Se insertan 2 barras de scroll para cuando el contenido sea demasiado grande (desaconsejado porque aparecerán las barras aunque el contenido se ajuste bien).
- **auto:** Más aconsejable que la de arriba. Aunque esta opción hace que dependa del navegador, normalmente este añadirá una o las 2 barras de scroll al elemento conforme se necesiten (dependiendo de por donde se salga el contenido).

8.3. Propiedad z-index

Indica la profundidad del elemento en la página o posición [en el eje Z](#). Por defecto el valor es 0. Se puede emplear para cuando un elemento esté encima de otro por ejemplo. Sus valores son:

- **auto**: el navegador decide.
- **Numero** negativo, 0, o positivo: Cuanto menor sea el valor más profundo estará el elemento, y cuanto mayor más arriba. Lo que define este valor es que si dos elementos tienen partes que están en el mismo sitio, cual es el que estará más arriba y por lo tanto, será completamente visible.

8.4. Propiedad left

Con esta propiedad se especifica que posición debe ocupar el elemento hacia la izquierda. Sus valores son auto, porcentaje o una medida en píxeles u otro sistema métrico. Dependiendo del valor de la propiedad [position](#), esta posición puede ser:

- [position](#): [static](#) → No tiene ningún efecto esta propiedad.
- [position](#): [relative](#) → Distancia del margen izquierdo con respecto a su posición normal (relativa). Esto movería el elemento hacia la derecha. Este valor puede ser negativo consiguiendo el efecto contrario.
- [position](#): [absolute](#) → Definiría la distancia del elemento en su parte izquierda con respecto al padre, o elemento que lo contiene (parecido a [margin-left](#)). Este valor puede ser negativo consiguiendo el efecto contrario.
- [position](#): [fixed](#) → Definiría la distancia del elemento en su parte izquierda con respecto a la ventana de la página web. Este valor puede ser negativo consiguiendo el efecto contrario.

8.5. Propiedad right

Igual que [left](#), pero tomando como referencia el lado derecho.

8.6. Propiedad top

Igual que [left](#), pero tomando como referencia la parte de abajo.

8.7. Propiedad bottom

Igual que [left](#), pero tomando como referencia la parte de arriba.

Ejercicio 9: Crea lo siguientes elementos. Ponles a todos un borde (puede ser el mismo para todos), y color de fondo (este diferente para cada uno) para diferenciarlos:

- Un `<div>` principal de `500px` de ancho y alto.
- Tres `<div>` dentro de este, que contengan texto, una altura de `100px` y anchura de `200px` cada uno. Con el segundo prueba las siguientes cosas:
 - Cámbiale el valor de la propiedad [position](#). Cada vez que la cambies mueve su posición con valores en las propiedades [top](#), [bottom](#), [right](#) o [left](#) para ver su efecto. Si al cambiar de posición se queda encima o debajo de otro elemento,

prueba a cambiar la propiedad `z-index` para traerlo atrás o al frente.

- Introduce texto que no quepa dentro del elemento y juega con la propiedad `overflow` para ver lo que pasa.

9. Posicionamiento con rejilla - grid

Tradicionalmente, se ha utilizado el posicionamiento visto anteriormente. Este sistema daba bastante dolores de cabeza sobre todo cuando queríamos adaptar una página a las diferentes resoluciones: escritorio, móviles, tablet, etc.

Hoy en día uno de los conceptos más utilizados es flexbox. Es un sistema de elementos flexibles donde los elementos HTML se adaptan y posicionan automáticamente siendo más fácil personalizar los diseños, el problema es que está orientado a estructuras de una sola dimensión. A día de hoy, muchos frameworks y librerías utilizan el sistema grid, donde definen una cuadrícula determinada y modificando los nombres de las clases de los elementos podemos darle tamaño y/o posición.

9.1. Grid y flexbox

La diferencia básica entre estos dos sistemas es que flexbox se creó para diseños de una única dimensión ya sea en vertical (columna) o en horizontal (fila). En cambio grid se pensó para el diseño bidimensional, es decir, en varias filas y columnas al mismo tiempo.

9.2. La rejilla – filas y columnas

Los elementos básicos de este nuevo esquema son los siguientes:

1. **Contenedor:** es el elemento padre, en su interior estarán los elementos (ítems)
2. **Ítem:** Cada uno de los hijos(elementos) que va a tener el contenedor en su interior.

Código HTML

```
...
<div class="container">
  <div class="celda1">Celda 1</div>
  <div class="celda2">Celda 2</div>
  <div class="celda3">Celda 3</div>
  <div class="celda4">Celda 4</div>
</div>
...
```

Una vez que hemos definido los elementos que va a tener nuestro contenedor, debemos pasar a indicar desde CSS que le activamos la cuadrícula `grid`, es decir:

Código CSS

```
...
.container{
  display: grid;
}
...
```

El siguiente paso será configurar nuestra cuadrícula, es decir, utilizando las propiedades `grid-template-columns` y `grid-template-rows`, indicaremos las dimensiones de cada una de las celdas, diferenciando entre columnas y filas.

- **grid-template-columns:** Nos permite establecer el tamaño de cada una de las columnas. Para ello, debemos separar por espacios el tamaño de cada columna pudiendo utilizar porcentajes, píxeles, `auto` (que obtiene el tamaño restante), etc. o la unidad propia de grid llamada fr (fraction) que se refiere a fracciones del espacio disponible (restante) del grid. Ejemplo, una columna 2fr será el doble de ancha que una 1fr.
- **grid-template-rows:** Nos permite establecer el tamaño de cada una de las filas. En este caso no se usa fr, utilizamos píxeles, `auto`.

Por ejemplo, si queremos crearnos una rejilla de 3x3 donde la columna del medio es el doble de ancha que las otras dos, y la primera fila y última tienen una altura de 40px dejando la altura de la fila del centro que se ajuste automáticamente al contenido (`auto`) sería:

Código CSS

```
...
.container{
  display: grid;
  grid-template-columns: 1fr 2fr 1fr; /* Otra opción para definir las columnas sería 25% 50% 25%*/
  grid-template-rows: 40px auto 40px;
}
...
```

Nota: Podemos combinar varias unidades diferentes, pudiendo utilizar píxeles (px), fracciones restantes (fr), porcentajes (%) de forma simultánea.

Si queremos repetir los valores de las columnas/filas, podemos utilizar el método `repeat([num_veces], [valor o valores])`, donde indicaríamos las veces que repetimos el valor que le indicamos. Si ponemos varios valores estaremos repitiendo los valores tantas veces como `num_veces` indiquemos. Veamos un ejemplo:

Código CSS

```
...
.container{
  display: grid;
  grid-template-columns: 100px repeat(2, 50px) 100px;
  grid-template-rows: repeat(2, 50px 60px);
}
...
```

Con el ejemplo anterior nos estamos creando una rejilla de 8 ítems, (4x4), las columnas serán: 100px, 50px 50px 100px, y las filas serán 50px 60px 50px 60px.

9.3. Espacios/márgenes/huecos en grid

Por defecto, la cuadrícula tiene sus celdas contiguas. Podríamos darle un margin, pero en grid la forma más apropiada es con gutters, es decir, utilizando las propiedades `column-gap` y/o `row-gap`.

- **column-gap:** establecemos el tamaño de los huecos entre columnas (verticales)
- **row-gap:** establecemos el tamaño de los huecos entre filas (horizontales)

Código CSS

```
...
```

```

.container{
  column-gap: 10px;
  row-gap: 20px;
}
...

```

Con esto estaríamos dando un espacio de 20px entre filas, y un espacio de 10px entre columnas. Como atajo a estas dos propiedades podríamos utilizar la propiedad **gap**.

Código CSS

```

...
.container{
  gap: 20px 10px; /* Estaríamos haciendo como en el ejemplo anterior, «row-gap»«column-gap» */

  /* Si queremos darle el mismo tamaño de separación a la fila y columna con poner el valor
  1 vez nos vale */
}
...

```

9.4. Posición de los elementos en el grid

Por defecto, los elementos se colocan de izquierda a derecha y de arriba a abajo ocupando cada uno una celda de la rejilla.

1	2	3
4	5	6
7	8	9

Si queremos cambiar la ubicación, podemos utilizar las siguientes propiedades:

- **grid-column-start**: Indica en qué columna empezará el ítem de la cuadrícula
- **grid-column-end**: Indica en qué columna terminará el ítem de la cuadrícula
- **grid-row-start**: Indica en qué fila empezará el ítem de la cuadrícula
- **grid-row-end**: Indica en qué fila terminará el ítem de la cuadrícula

Por ejemplo, si queremos que la posición de los elementos sea la siguiente:

1	Ítem 2		Ítem 3
2		Ítem 4	
3			Ítem 1
4			

El código será el siguiente:

Código CSS

```

...
.item1{
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row-start: 3;
  grid-row-end: 4;
}
.item2{
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row-start: 1;
  grid-row-end: 2;
}
.item3{
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 2;
}
.item4{
  grid-column-start: 2;
  grid-column-end: 3;
  grid-row-start: 2;
  grid-row-end: 3;
}
...

```

Como atajo al posicionamiento anterior, podemos utilizar la notación `grid-row/grid-column`, su sintaxis es: `grid-row: <grid-row-start> / <grid-row-end>;` y lo mismo sería con las columnas: `grid-column: <grid-column-start> / <grid-column-end>;` El ejemplo anterior, quedaría:

Código CSS

```

...
.item1{
  grid-column: 3 / 4;
  grid-row: 3 / 4;
}
.item2{
  grid-column-start: 1/2;
  grid-row-start: 1/2;
}
.item3{
  grid-column-start: 3/4;
  grid-row-start: 1/2;
}
.item4{
  grid-column-start: 2/3;
  grid-row-start: 2/3;
}
...

```

Nota: Cuando ocupa sólo una fila o una columna el `/ <grid-XXX-end>;` puede omitirse. El ejemplo anterior quedaría:

Código CSS

```

...
.item1{
  grid-column: 3;
  grid-row: 3;
}
.item2{
  grid-column-start: 1;
  grid-row-start: 1;
}
.item3{
  grid-column-start: 3;
  grid-row-start: 1;
}
.item4{
  grid-column-start: 2;
  grid-row-start: 2;
}
...

```

¡IMPORTANTE! Un contenido puede ser contenedor a su vez, es decir, si tratamos grid como una tabla, dentro de una celda podemos crear otra tabla a su vez.

Juego para aprender grid: <https://cssgridgarden.com/#es>

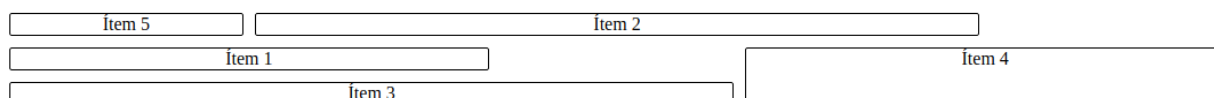
Ejercicio 10: Utilizando **grid** y el siguiente código html (disponible en AULES), crea una página dónde el resultado sea el siguiente:

Código HTML

```

...
<div class="container">
  <div class="item" id="item1">Ítem 1</div>
  <div class="item" id="item2">Ítem 2</div>
  <div class="item" id="item3">Ítem 3</div>
  <div class="item" id="item4">Ítem 4</div>
  <div class="item" id="item5">Ítem 5</div>
</div>
...

```



10. Propiedades de las listas

En esta sección vamos a ver propiedades de CSS específicas para las listas, es decir, se aplicarían solamente al elemento que define la lista. En este caso hablamos de

``, ``, o también `` en muchos casos. Si vamos a modificar el tipo de marcador usado para la lista con CSS, no importa que hayamos definido la lista como ordenada o desordenada, funcionará de la misma manera.

10.1. Propiedad `list-style-type`

Esta propiedad define el tipo de marcador que vamos a usar en los elementos de la lista. En HTML vimos que podíamos usar círculos, cuadrados, letras del alfabeto, números romanos, etc... Con CSS estas posibilidades se amplían bastante. Podemos tener desde los estilos clásicos que teníamos en HTML (`circle`, `square`, `decimal`, `lower-roman`, ...), hasta numerarlos por el alfabeto hebreo (`hebrew`), armenio (`armenian`), e incluso los silabarios del japonés (`hiragana`, `katakana`), o los ideogramas usados para representar los números tanto en japonés como en chino, llamados kanji en japonés (`cjk-ideographic`).

Un ejemplo de todos estos estilos los podemos encontrar [aquí](#).

10.2. Propiedad `list-style-image`

Si se define esta propiedad y la anterior, se aplicará esta última, ya que tiene más peso. Lo que define es una imagen que se utilizará para marcar los elementos de la lista en lugar de usar alguno de los estilos de arriba.

El valor a usar es el clásico para definir una imagen, es decir, `url(ruta_imagen)` por ejemplo. No hay que olvidar usar el `url()`, para englobar la imagen. En la siguiente [página](#) podréis ver un ejemplo utilizando un cuadrado rosa como imagen para los elementos. Debemos tener en cuenta que el tamaño de la imagen será el que se aplique, no se puede modificar dicho tamaño desde código.

10.3. Propiedad `list-style-position`

Esta propiedad indica si el espacio de tabulador que se le aplica a los elementos de la lista, se le aplicará también al elemento marcador. El efecto final es que la lista aparecerá un poco más desplazada a la derecha, o como siempre. Los valores son:

- `outside`: Este es el valor por defecto que se aplica siempre si no se indica nada. El tabulado sólo se aplica al texto de cada elemento, quedando el marcador un poco más a la izquierda.
- `inside`: Incluye el marcador en el desplazamiento por tabulado, así que toda la lista estará un poco desplazada hacia la derecha. Se pueden ver ambos efectos en la siguiente [página](#).

Si quisiéramos que una lista tuviera como marcador una imagen, en lugar de los símbolos de siempre, y si además queremos que la lista esté desplazada un poco más a la derecha porque el tabulado afecte también al marcador, lo pondríamos así:

Código HTML

```
...
<ul id="lista_imagen">
  <li>Elemento 1</li>
  <li>Elemento 2</li>
</ul>
...
```

Código CSS

```
#lista_imagen{
  list-style-image: url(flecha1.jpg);
  list-style-position: inside;
}
```

Podemos hacer que algunos elementos de la lista tenga un estilo diferente al resto de la lista definiendo su propio estilo:

Código HTML

```
...
<ul class="numeros">
  <li>Elemento 1</li>
  <li class="romanos">Elemento 2</li>
  <li>Elemento 3</li>
</ul>
...
```

Código CSS

```
.numeros{
  list-style-type: decimal;
  list-style-type: upper-roman;
}
```

Si queremos indicar varias propiedades de las listas de una vez usamos la propiedad `list-style`, el orden aconsejado es: `list-style: <type> <position> <image>;`


Ejercicio 11: Crea un documento HTML con dos listas:

- Crea una lista con 3 elementos mínimo, en la que utilices una imagen pequeña como marcador.
- Crea una lista con 6 elementos. Los elementos impares (1º, 3º, 5º) tendrán un estilo de marcador tipo `hiragana`, mientras que los elementos pares (2º, 4º y 6º) tendrán un estilo `katakana`.

10.4. Listas y menú

Una práctica muy habitual es la de usar listas para definir los elementos de un menú (como sabemos en HTML5 el menú va en el `<nav>`). Es habitual ver los elementos uno al lado del otro, pero como estamos viendo las listas por defecto aparecen un ítem debajo del otro al ser etiquetas de bloque. Para modificar su comportamiento deberemos utilizar la propiedad `display` para los ítem y forzarles a que se comporten como un elemento en línea para que estén uno tras otro.

Por ejemplo, en la web del instituto tenemos el siguiente menú:



Sería algo así:

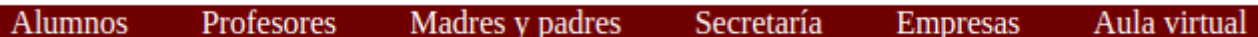
Código HTML

```
...
<nav>
  <ul class="menuPrincipal">
    <li><a title="Alumnos" href="alumnos.html">Alumnos</a></li>
    <li><a title="Profesores" href="profesores.html">Profesores</a></li>
    <li><a title="Madres y padres" href="ampa.html">Madres y padres</a></li>
    <li><a title="Secretaría" href="secretaria.html">Secretaría</a></li>
    <li><a title="Empresas" href="empresas.html">Empresas</a></li>
    <li><a title="Aula virtual" href="aulavirtual.html">Aula virtual</a></li>
  </ul>
</nav>
...
```

Código CSS

```
nav {
  background-color: #700000;
}
nav ul.menuPrincipal {
  list-style-type: none;
  text-align: center;
}
nav ul.menuPrincipal li {
  display: inline;
  padding: 1%;
}
ul.menuPrincipal li a {
  color: white;
  text-decoration: none;
}
```

Consiguiéndose el siguiente resultado:



11. Selectores avanzados

Durante la primera semana vimos cómo acceder a los distintos elementos de nuestro HTML utilizando selectores. Vimos cómo seleccionar un determinado elemento con su id, a varios elementos por la clase que utilizaba y otros selectores más avanzados como era un descendiente (selectorPadre selectorDescendiente), a los hijos directos (selectorPadre > selectorHijo), al hermano adyacente (selector + hermanoAdyacente), selector al hermano general (selector ~ hermano), y el selector universal (*).

11.1.Pseudoclasses

Las pseudoclasses se utilizan para referenciar a ciertos comportamientos de los elementos HTML. Las pseudoclasses se definen añadiendo dos puntos (:) antes de la pseudoclase.

11.1.1. Pseudoclasses de enlaces

Son pseudoclasses orientadas a hipervínculos o hiperenlaces (etiqueta `<a>`).

- **:link** → Aquellos que no han sido visitado todavía
- **:visited** → Aquellos que han sido visitados anteriormente

Veamos un ejemplo. Imaginad que queremos poner de color rojo y en negrita aquellos enlaces que aún no han sido visitados:

```
a:link {
  color: red;
  font-weight: bold;
}
```

11.1.2. Pseudoclasses de ratón

Las siguientes pseudoclasses se utilizan para cambiar el comportamiento cuando nos posicionamos o pulsamos un elemento con el ratón.

- **:hover** → cuando pasamos el ratón por encima del elemento
- **:active** → cuando estamos pulsando el elemento con el ratón

El ejemplo típico de `:hover` es cuando pasamos el ratón por encima de un enlace (`<a>`), vamos a cambiarle el color a la letra y a añadirle un `padding` a dicho elemento:

```
a:hover {
  color: steelblue;
  padding: 2px;
}
```

11.1.3. Pseudoclasses de interacción

Están orientadas a formularios y cómo interaccionan los usuarios con ellos:

- **:focus** → cuando el elemento tiene el foco (ha pulsado sobre un input para introducir datos)

- **:checked** → cuando la casilla ha sido marcada

Por ejemplo, le vamos a dar un borde verde al input con el foco:

```
input:focus {
  border: 2px dotted #4C4;
}
```

11.1.4. Pseudoclases de activación

De forma habitual, los elementos de un formulario HTML están activados, pero podemos querer desactivar un cierto campo en un momento dado añadiendo el atributo **disabled** en el HTML. Las distintas pseudoclases que nos permite detectar si un campo está activado/desactivado son:

- **:enabled** → cuando el campo está activado
- **:disabled** → cuando el campo está desactivado
- **:read-only** → cuando el campo es de sólo lectura
- **:read-write** → cuando el campo es editable por el usuario

Por ejemplo, vamos a darle un color de fondo blanco a los input que estén activos, y a los que no están activos los ponemos de color gris.

```
input:enabled {
  background-color: white;
}
input:disabled {
  background-color: grey;
}
```

11.1.5. Pseudoclasses de validación

Como sabemos, con HTML5 podemos hacer una primera validación a los campos de un formulario, asegurándonos que el usuario escribe un campo de forma correcta. Existen algunas pseudoclases para las validaciones:

- **:required** → Cuando el campo es obligatorio (atributo html **required**)
- **:optional** → Cuando el campo es opcional (por defecto todos son opcionales)
- **:invalid** → Campo que no cumple la validación HTML5
- **:valid** → Campo que cumple la validación HTML5
- **:out-of-range** → Campos numéricos fuera de rango
- **:in-range** → Campos numéricos dentro del rango

Por ejemplo, vamos a poner un borde azul a aquellos input que son de carácter obligatorio de rellenar:

```
input:required {
  border: 2px solid blue;
}
```

11.1.6. Pseudoclasses de negación

La pseudoclase de negación, nos permite seleccionar todos los elementos que no cumplan los selectores indicados entre paréntesis. Por ejemplo, imaginad que queremos seleccionar todos los elementos `<div>` que no sean la clase `container`:

```
div:not(.container) {
  border: 2px solid blue;
}
```

11.1.7. Pseudoclasses para elementos según su posición

Podemos acceder a ciertos elementos del documento HTML dependiendo de su posición y estructura de los elementos hijos:

- **:first-child** → Primer elemento hijo
- **:last-child** → Último elemento hijo
- **:nth-child(n)** → N-elemento hijo
- **:nth-last-child(n)** → N-elemento hijo partiendo desde el final

Por ejemplo, vamos a cambiar el fondo de un artículo para el primer y último `strong`.

Archivo HTML	Archivo CSS
<pre>... <div id="contenido"> <div class="articulo"> Primer mensaje Segundo mensaje Tercer mensaje Último mensaje </div> <div class="articulo"> Primer mensaje Segundo mensaje Tercer mensaje Último mensaje </div> </div> ...</pre>	<pre>strong:first-child{ background-color: blue; } strong:last-child{ background-color: green; }</pre>

Si lo que queremos es seleccionar un elemento hijo en concreto pero que no es el primero ni el último, como hemos dicho usaríamos `:nth-child(n)`. Por ejemplo, con `strong:nth-child(1)` estaríamos seleccionando el primer elemento hijo que además es un ``. Con esta pseudoclase por ejemplo, podríamos seleccionar todos los hijos pares con: `:nth-child(2n)`, es decir, no es necesario poner un número.

11.1.8. Pseudoclasses para elementos del mismo tipo

En el punto anterior hemos visto cómo seleccionar elementos independientemente del tipo, en este punto veremos cómo seleccionar solo a elementos del mismo tipo:

- **:first-of-type** → Primer elemento hijo (de su mismo tipo)
- **:last-of-type** → Último elemento hijo (de su mismo tipo)
- **:nth-of-type(n)** → N-elemento hijo (de su mismo tipo)
- **:nth-last-of-type(n)** → N-elemento hijo (de su mismo tipo) partiendo desde el final

Las pseudoclases `:first-of-type` y `:last-of-type` son equivalentes a `:first-type` y `:last-type` pero sólo teniendo en cuenta elementos del mismo tipo. En cambio las pseudoclases `:nth-of-type(n)` es la equivalente a `:nth-child(n)` y `:nth-last-of-type(n)` es la equivalente a `:nth-last-child(n)`. Veamos un ejemplo:

Archivo HTML

```
...
<div id="contenido">
  <div class="articulo">
    <strong>Primer mensaje</strong>
    <span>Segundo mensaje</span>
    <strong>Tercer mensaje</strong>
    <strong>Último mensaje</strong>
  </div>
  <div class="articulo">
    <strong>Primer mensaje</strong>
    <strong>Segundo mensaje</strong>
    <span>Tercer mensaje</span>
    <strong>Cuarto mensaje</strong>
    <strong>Último mensaje</strong>
  </div>
</div>
...
```

Archivo CSS

```
strong:first-of-type{
  background-color: yellow;
}
strong:nth-of-type(2){
  background-color: green;
}
strong:nth-last-of-type(1){
  background-color: blue;
}
```

11.1.9. Otras pseudoclases

Podemos encontrar otras pseudoclases como `:only-child`, `:only-of-type` o `:empty` entre otras en [aquí](#)

11.2. Pseudoelementos

Al igual que las pseudoclases, los pseudoelementos son otra característica CSS que nos permite dar estilo a elementos que no existen realmente en el HTML y que podemos generar desde CSS. Con las pseudoclases utilizábamos los dos puntos (:), ahora con pseudoelementos utilizaremos dos dobles puntos (::), aunque antiguamente se utilizaba de la misma forma, con sólo dos puntos.

11.2.1. Pseudoelemento de generación de contenido

Con la propiedad `content` se utilizan los selectores `::before` o `::after`, para indicar que vamos a crear contenido antes o después del elemento respectivamente. Los parámetros que admite esta propiedad es variado, permitiendo concatenar información mediante espacios. Los tipos de contenido que podemos utilizar son:

- **"string"**: Añade el contenido de texto indicado. Ejemplo: `content: "Hola mundo";`
- **attr(atributo)**: Añade el valor del atributo HTML indicado.

- **url(URL):** Añade la imagen indicada en la URL `content: url(fondo.png);`

Podéis encontrar más ejemplos [aquí](#).

11.2.2. Pseudoelemento – Primera letra y/o primera línea

También existen pseudoelementos con los que podemos referenciar a la primera letra de un texto (`::first-letter`) o a la primera línea (`::first-line`) y aplicarle un estilo en concreto.

```
p{
  color: #333;
  font-size: 14px;
}
p::first-letter{
  color: black;
  font-size: 40px;
}
p::first-line{
  color: #999;
}
```

Podemos encontrar más pseudoelementos [aquí](#).

11.3. Atributos

Por último, una característica de CSS muy interesante es la de aplicar estilo dependiendo del atributo y/o contenido que tenga el elemento HTML. En CSS, estos atributos van entre corchetes ([atributo]) y hay varias formas de utilizarlos:

- **[href]:** Si el elemento tiene el atributo href.
- **[href="#"]:** Si el elemento tiene el atributo href y su valor es #.
- **[href*="btn"]:** Si el elemento tiene el atributo href y su valor contiene btn.
- **[href^="https://"]:** Si el elemento tiene el atributo href y su valor comienza por https://.
- **[href\$=".pdf"]:** Si el elemento tiene el atributo href y su valor termina por .pdf.
- **[class~="btn"]:** Si el elemento tiene el atributo class con una lista de valores y uno de esos es btn.
- **[lang]="es"]:** Si el elemento tiene el atributo lang y uno de sus valores empieza por es. (es-ES, es-AR, etc.)

Veamos unos ejemplos:

Atributo existente: Vamos a hacer que todos los elementos HTML que tienen el atributo href no tengan un subrayado.

```
[href]{
  text-decoration: none;
}
```

Atributo con valor exacto: Si por ejemplo queremos acceder a un elemento HTML que tiene la clase `active` sería:

```
[class="active"]{
  background-color: aquamarine;
}
```

Atributo contiene texto: Por ejemplo, vamos a modificar el fondo de aquellos elementos HTML cuyo atributo `name` **contenga** la cadena: `mi`

```
[name*="mi"]{
  background-color: aquamarine;
}
```

Con el ejemplo anterior estamos seleccionando aquellos que en el `name` contiene la cadena `mi`: por ejemplo: `mitexto`, `xiaomi`, etc. es decir, no importa dónde esté la cadena `"mi"`

Comienzo atributo: Por ejemplo, vamos a modificar el fondo de aquellos elementos HTML cuyo atributo `href` **comienza por** la cadena: `"https://"`

```
[href^="https://"]{
  background-color: aquamarine;
}
```

Final de atributo: Por ejemplo, vamos a modificar el fondo de aquellos elementos HTML cuyo atributo `href` **acaba** en `".pdf"`

```
[href$=".pdf"]{
  background-color: aquamarine;
}
```

Ojo! Se distinguen entre mayúsculas y minúsculas, es decir, en el ejemplo anterior se seleccionarían los enlaces que tengan un fichero acabado en `.pdf`, pero NO aquellos que acaben en `.PDF` o `.PdF` o cualquier otra combinación que implique alguna mayúscula. Si queremos que no distinga entre mayúsculas y minúsculas debemos poner una `i` (case insensitive) antes del cierre del atributo:

```
[href$=".pdf" i]{
  background-color: aquamarine;
}
```

Por tanto, a la hora de seleccionar para dar estilo los elementos la estructura que sería la siguiente en CSS:

```
selector #id .clase :pseudoclase ::pseudoelemento [atributo]{
  propiedad: valor;
}
```

12. Cascada en CSS

Como dijimos, uno de los conceptos más importantes en CSS es el de cascada.

<p>Archivo HTML</p> <pre>... <div id="contenido">Hola mundo </div></pre>	<p>Archivo CSS</p> <pre>div{ color: red; padding: 8px; } div{ color: blue; background-color: grey; }</pre>
--	--

Si nos fijamos, en el anterior código estamos aplicando estilos exactamente al mismo selector (div) y donde coincide la propiedad color con diferente valor, ¿Cuál de estas dos reglas prevalece? La respuesta es fácil, siempre prevalece la última regla definida, la cual mezcla y sobrescribe las propiedades definidas anteriormente. Es decir, es como si hubiéramos puesto:

```
div{
    padding: 8px;
    color: blue;
    background-color: grey;
}
```

Sin embargo, puede ocurrir que en determinados casos no esté tan claro cuál es el estilo que sobrescribirá a los anteriores. Veamos un ejemplo:

<p>Archivo HTML</p> <pre>... <div id="mensaje" class="error">Hola mundo </div></pre>	<p>Archivo CSS</p> <pre>div{ color: blue; } #mensaje{ color: green; } .error{ color: red; }</pre>
--	---

En el ejemplo anterior tenemos un div con el atributo id con valor nombre, y el atributo class con valor error. ¿Qué estilo se va a aplicar dado el CSS? ¿Cuál tiene prioridad aquí? En este punto entra en juego el concepto de cascada.

El navegador, para saber qué bloque de estilos tiene prioridad lo que hace es analizar: importancia(!important), especificidad y orden.

12.1. !important

Cuando a una propiedad CSS le añadimos la cadena: `!important` al final, estamos diciéndole al navegador que esa regla tiene prioridad frente a las demás. Si hay varias reglas con `!important`, prevalecerá la última de todas.

Archivo HTML
...
<div>Hola mundo </div>

Archivo CSS
div{
 color: red !important;
 padding: 8px;
}
div{
 color: blue;
 background-color: grey;
}

El resultado final sería:

```
div{
  padding: 8px;
  color: red;
  background-color: grey;
}
```

12.2. Especificidad de los selectores

La [especificidad](#) es uno de los criterios más importantes de CSS. Es un peso que se le da a una declaración CSS determinada por el tipo de selector usado. Cuando varias declaraciones tienen igual especificidad entonces se aplica al elemento la última declaración encontrada. La especificidad sólo se aplica cuando el mismo elemento es objetivo de múltiples declaraciones.

La siguiente lista determina la especificidad:

- estilos aplicados mediante un **atributo** style
- Número de veces que aparece un **id** en el selector
- Número de veces que aparece una **clase**, **pseudoclase** o **atributo** en el selector.
- Número de veces que aparece un **elemento** o **pseudoelementos** en el selector

Para saber por tanto la prioridad, debemos ordenarlos teniendo en cuenta los valores de izquierda a derecha:

```
div{...} /* Especificidad: 0, 0, 0, 1 */
div div{...} /* Especificidad: 0, 0, 0, 2 */
#page div{...} /* Especificidad: 0, 1, 0, 1 */
#page div:hover{...} /* Especificidad: 0, 1, 1, 1 */
#page div:hover a{...} /* Especificidad: 0, 1, 1, 2 */
```



```
#page .sel:hover>a{...}
```

```
/* Especificidad: 0, 1, 2, 1 */
```

12.3. Orden

Dependiendo de en dónde declaremos la regla, se aplicará un valor u otro. DE mayor a menor prioridad:

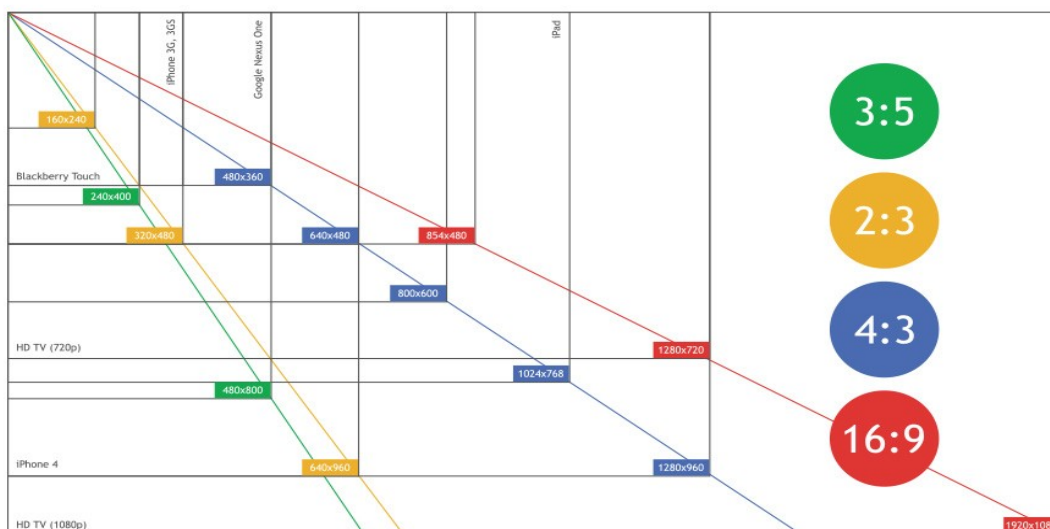
- CSS en el **atributo style**
- CSS en el **elemento style** (en el head head)
- CSS en **archivo** / hoja de estilos separada del código HTML (**.css**)

13. Diseño adaptativo y responsive

El diseño web responsive (RWD Responsive Web Design) es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas. Hoy día las páginas web se ven en multitud de dispositivos como tablets, móviles, portátiles, PC, etc. Además, aún dentro de cada tipo, cada dispositivo tiene sus características concretas: tamaño de pantalla, resolución, potencia de CPU, sistema operativo o capacidad de memoria entre otras. Esta tecnología pretende que con un único diseño web, todo se vea correctamente en cualquier dispositivo.

Variabilidad en las resoluciones de pantalla

Antiguamente el desarrollo web se basaba en la resolución estándar de 1024 x 768, pero hoy existe una amplia variedad de resoluciones, no solo en ordenadores de sobremesa sino también para tablets y móviles. Cuando vamos a crear una web, debemos tener en cuenta que los usuarios finales no van a tener toda la misma resolución de pantalla.



Estos di

- Pantalla de teléfono **móvil** de baja resolución. Es la resolución extra-pequeña, extra-small, abreviada como **xs**. La resolución máxima típica que tiene es inferior a **576px**
- Dispositivos **móviles** pero con una mayor resolución, es la resolución pequeña, small, abreviada en **sm** cuya resolución máxima es de 768px

- Las pantallas **tablets** son el siguiente escalón, son pantallas pequeñas de una resolución media, abreviada como **md**, cuyos valores máximos son **992px**
- Las pantallas más grandes, abreviadas como **lg** que tienen los monitores de media gana alcanzan resoluciones de **1200px**
- Le siguen las pantallas **xl** (aun más grandes) con resoluciones de hasta **1400px**
- Dejando para el último escalón las **xxl** que son todavía más grandes.

A la hora de crear una web, debemos tener en cuenta a qué usuarios va destinados para poder analizar su usabilidad, ya que ya no podemos centrarnos en desarrollar una web pensando que los usuarios van a tener probablemente una única resolución de pantalla.

Es fundamental tener en cuenta que en el diseño responsive **deberemos mostrar en primer lugar los contenidos más importantes e imprescindibles**, y luego el resto del contenido (u ocultar aquello que no queramos mostrar en determinadas resoluciones).

13.1. Diseño adaptativo vs responsive

Con el diseño **responsive** conseguimos que con un único diseño nuestra web se adapte a las distintas resoluciones, de modo que dependiendo de la resolución los elementos se reposicionan en función de las mismas. Para conseguir esto tenemos multitud de frameworks como Bootstrap, y elementos como estudiaremos a continuación, **flexbox**.

Por otro lado, tenemos el diseño **adaptativo** que nos permitirá definir un estilo/diseño para cada una de las posibles resoluciones, de modo que en función de la resolución que tenga el dispositivo se cargará un diseño u otro. Esta estrategia veremos cómo se lleva a cabo mediante el uso de **media queries**.

13.2. Media queries

Las medias queries son un tipo de reglas de CSS que nos van a permitir crear bloques de código. Estos bloques serán procesados en aquellos dispositivos que cumplan la siguiente regla:

```
@media [only | not] <media_type> <expresion1,expresion2, expresion3...>{
/* Estilos a aplicar para cuando se cumpla la regla */
}
```

donde,

- **media_type**: corresponde al tipo de medio al que se van a aplicar esos estilos como screen(sólo en pantallas), print (cuando seleccionamos imprimir una web se aplicaría ese estilo. De forma habitual, el número de imágenes que se muestran al imprimir se reducen), speech (lectores de texto para invidentes) o all(todos los dispositivos, por defecto está este valor).
- **<expresiones>**: es una o varias condiciones que debe cumplir el dispositivo. Normalmente corresponde a su anchura máxima o mínima.

Veamos un ejemplo:

```
@media screen and (max-width: 650px){
```

```

.menu{
  /* Estilos a aplicar a los elementos que tengan la clase menu y su ancho de pantalla
    del dispositivo sea como máximo 650px (móviles)*/
}
}
@media screen and (min-width: 640px) and (max-width: 1280px){
  .menu{
    /* Estilos a aplicar a los elementos que tengan la clase menu y su ancho de pantalla
      del dispositivo sea estén entre 1280px y 640px (tablets)*/
  }
}
@media screen and (min-width: 1280px){
  .menu{
    /* Estilos a aplicar a los elementos que tengan la clase menu y su ancho de pantalla
      del dispositivo sea como máximo 650px (pantallas pc)*/
  }
}

```

Para nuestras prácticas, nos será suficiente con especificar en el apartado de expresión la anchura máxima/mínima, por ejemplo:

```

@media (min-width: 768px){
  /* Estilos a aplicar para cuando se cumpla la regla */
}

```

No suele ser muy habitual el encontrarte reglas `@media` con la palabra clave `not`, que hace que se aplique la hoja de estilos cuando no se cumpla dicha regla.

Es importante recordar el meta que ponemos en el `<head>` de nuestra web, donde estamos indicando que el ancho de la pantalla es el ancho del dispositivo, por lo que el aspecto del `viewport` se va a adaptar:

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">

```

Además de esta forma de indicar los media queries, tenemos la posibilidad de indicarlos en el link y que cada link apunte a un archivo css diferente, por ejemplo para el caso anterior:

```

<link rel="stylesheet" href="movil.css" media="screen and (max-width: 640px)">
<link rel="stylesheet" href="tablet.css" media="screen and (min-width: 640px) and (max-width: 1280px)">
<link rel="stylesheet" href="screen.css" media="screen and (min-width: 1280px)">

```

Una buena práctica es definir los estilos que serán comunes a cualquier resolución junto con los estilos para la resolución más baja o más alta, y luego poner en los media queries las características que para el resto de resoluciones.

```

...
<section id="container">
  <section id="section1">section 1</section>
  <section id="section2">section 2</section>
</section>

```

Por ejemplo, imaginad que en el section de nuestro HTML tenemos dos sections y queremos que para los móviles se muestren una debajo de otra pero para pantallas

mayores que se muestren en paralelo. El primer paso será definir el estilo general y el estilo para los dispositivos móviles:

```
*{
  padding: 0px;
  margin: 0px;
}
#container{
  margin: 10px 5%;
}
#section1{
  background-color: red;
  padding: 10px;
  border: 1px solid black;
}
#section2{
  background-color: blue;
  padding: 10px;
  border: 1px solid black;
}
```

El siguiente paso será definir el estilo para el resto de pantallas que no son móviles (tablets y pc), que como en este caso serán las dos sections una al lado de otra en ambos casos sería:

```
@media (min-width: 768px){
  #container{
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-rows: auto;
    gap: 10px;
  }
}
```

Por último, dentro de este punto indicar que tenemos la posibilidad para detectar ciertas preferencias de usuario dependiendo del dispositivo, sistema operativo o navegador que utiliza. Estas preferencias son: [prefers-color-scheme](#), [prefers-contrast](#), [prefers-reduced-motion](#) y [prefers-reduced-data](#).

13.3. ¿Cómo pruebo cómo se ve con el navegador diferentes resoluciones?

Más herramientas → Herramientas para desarrolladores, de modo que se nos abre lo siguiente:

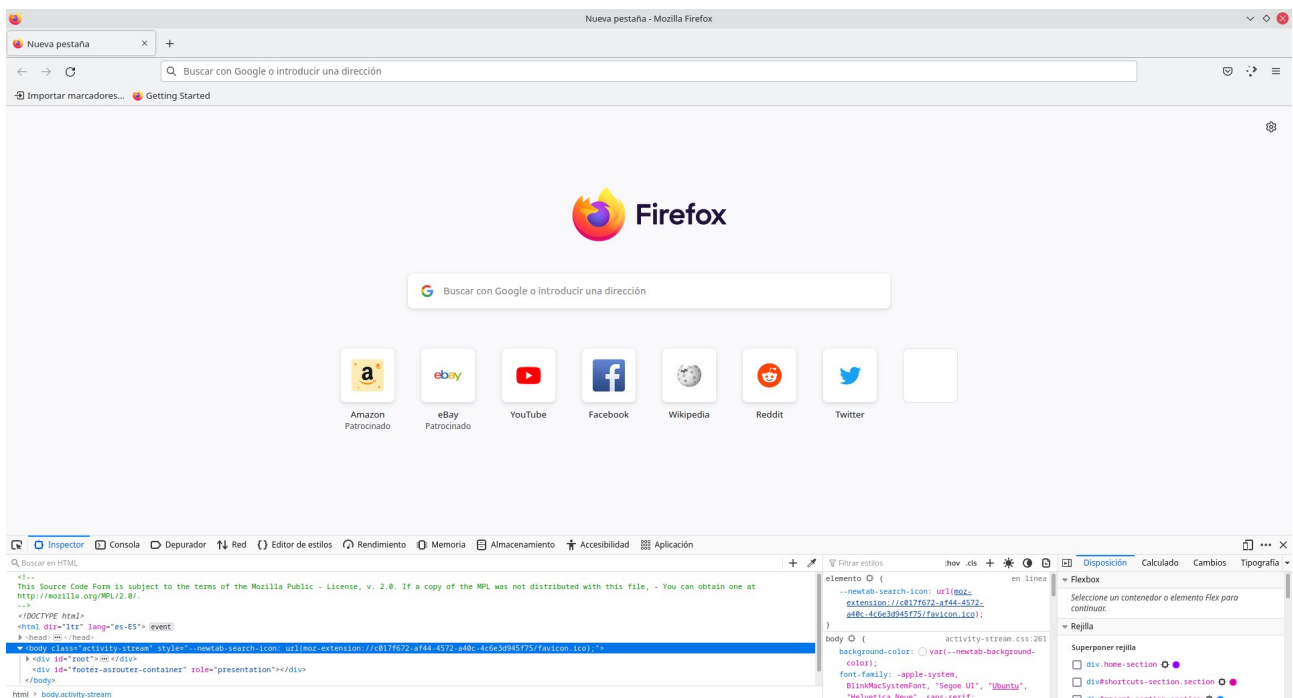


Figura 1: Firefox

Aquí ↓

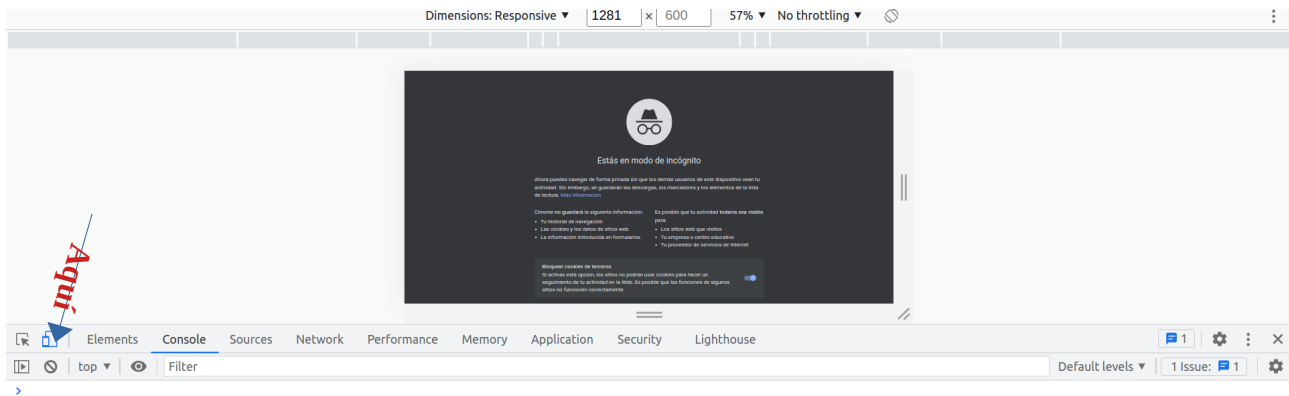


Figura 2: Chrome

Pudiendo desde arriba modificar el ancho/alto de la resolución.

Ejercicio 1: Dado el siguiente HTML disponible en AULES, crea una hoja de estilos donde utilizando media queries defines el estilo para estas tres resoluciones:

- Resoluciones pequeñas (hasta 768px), mostramos una caja debajo de otra

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

- Para resoluciones intermedias (hasta 992px) mostramos 1 caja arriba dos abajo y otra abajo, además la primera y última caja deberá tener diferente color

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

- Para resoluciones mayores mostramos las 4 cajas una al lado de la otra

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

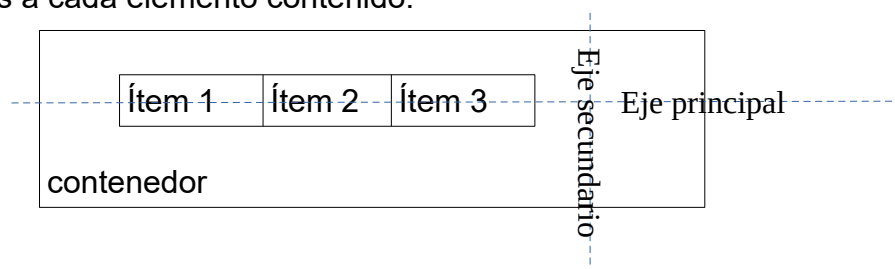
Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

14. Diseño responsive - flexbox

[Flexbox](#) es un sistema de elementos flexibles que nos permite que los elementos HTML se adapten y coloquen automáticamente siendo mucho más fácil personalizar diseños que con el posicionamiento, float, etc. que había antiguamente. Está especialmente diseñado para estructuras de una sola dimensión.

Al igual que con grid, es necesario que las cajas involucradas en el proceso estén contenidas, de modo que le daremos propiedades generales al contenedor y luego las específicas a cada elemento contenido.



14.1. Configuración de la caja contenedora

Entre las propiedades que podemos definir dentro de la caja contenedora(elemento padre), tenemos:

- **display**: sus valores serán `flex` o `inline-flex` dependiendo de si queremos establecer un contenedor en bloque(ocupar todo el ancho del padre) o un contenedor en línea (ocupar sólo lo que ocupe el contenido)
- **flex-direction**: nos permite cambiar la orientación del eje principal, es decir, cómo van a estar las cajas dentro del contenedor en el eje principal. Por defecto, el valor es `row`(establece la dirección del eje principal en horizontal), pero puede ser `row-reverse`(establece la dirección del eje principal en horizontal pero invertido), `column`(establece la dirección del eje principal en vertical), o `column-reverse`(establece la dirección del eje principal en vertical pero invertido)
- **justify-content**: se utiliza para alinear los ítems del eje principal, por defecto el valor es `flex-start`(agrupa los ítems al principio del eje principal), `flex-end`(agrupa los ítems al final del eje principal), `center`(agrupa los ítems al centro del eje principal), `space-between`(distribuye los ítems dejando el máximo espacio para separarlos), `space-around`(distribuye los ítems dejando el mismo espacio a la izquierda y derecha alrededor de ellos).
- **flex-wrap**: Nos permite indicarle cómo se deben comportar los ítems en el caso de que no quepan en la misma fila (o columna). Por defecto, el valor es `nowrap`(establece los ítems en una sola línea, no permite que se desborde el contenedor), pero puede ser `wrap`(establece los ítems en modo multilínea, permite que se desborde el contenedor), o `wrap-reverse`(establece los ítems en modo multilínea pero en dirección inversa).

Podemos probar todas estas propiedades/comportamientos y ver cómo funcionan [aquí](#).

14.2. Configuración de las cajas internas

Una vez que hemos configurado el contenedor, pasaremos a configurar los ítems que tendrá. Como es la ubicación de cada ítem, y el espacio a ocupar:

- **order**: indica el orden de aparición de los ítems. Por defecto, si no se indica nada, cada caja se coloca una a continuación de la otra, pero podemos intercalar los ítems. El valor es un **número**, que nos va a permitir colocar el ítem cuanto mayor sea más a al final de la secuencia del contenedor.
- **flex**: indica el **tamaño** de la caja respecto al resto. Al igual que con Grid usábamos el valor relativo (fr), aquí también se le da un valor relativo. Si todas las cajas tienen un valor de 1, todas van a ocupar lo mismo, si alguna tiene un 2 es porque esa será el doble que las otras. Del mismo modo que Grid, podemos darle un valor fijo en px o porcentajes(%).
- **min-width**: indica la **anchura mínima** que debe tener la caja del ítem para mostrar su contenido. Si esta anchura no se cumple y tenemos la opción `flex-wrap: wrap`; en el contenedor, hará que ocupe también la siguiente fila/columna disponible para mostrarse.

```
...
<section id="contenedor">
  <section id="section1">Ítem 1</section>
  <section id="section2">Ítem 2</section>
  <section id="section3">Ítem 3</section>
</section>
...
```

Dado el anterior HTML, vamos a darle estilo para que las tres cajas estén en horizontal ocupando la central el doble que las otras. Además, todas van a tener una anchura mínima de 300px. En el caso de no poderse cumplir, vamos a hacer que las cajas se distribuyan para que la anchura mínima se respete y por tanto (normalmente) que se sitúe un ítem debajo de otro.

Primero definimos las propiedades del contenedor.

```
#contenedor{
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  flex-wrap: wrap;
}
```

Una vez que hemos definido las propiedades del contenedor, pasamos a definir los ítems:

```
#section1, #section3{
  background-color: red;
  padding: 10px;
  margin: 10px;
  flex: 1;
  min-width: 300px;
}
#section2{
  background-color: blue;
  padding: 10px;
  margin: 10px;
  flex: 2;
  min-width: 300px;
}
```

Nota: Probad qué pasa si quitamos la propiedad: `flex-wrap: wrap;`. Como vemos las cajas ya no se redistribuyen de modo que si tenemos una pantalla pequeña tenemos que hacer scroll para verlas.

Para saber más de flexbox, podéis usar [esta página](#).

Ejercicio 2: Dado el código HTML del ejercicio anterior, crea una hoja de estilos donde utilizando flex-box defines el estilo para:

- Las dos cajas centrales ocupen el doble de tamaño que las laterales. Las laterales tendrán un color de fondo diferente.
- Las cajas centrales tendrán una anchura mínima de 200px y las laterales de 300px. Cuando esto no sea posible, se redistribuirán.

A continuación, os dejo unas imágenes para que veáis cómo se quedará en

determinadas resoluciones:

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

Lorem ipsum

dolor sit amet consectetur adipisicing elit. Fugit mollitia quod architecto quam molestias, blanditiis animi itaque explicabo aliquam enim sed ab? Adipisci, modi? Et inventore reprehenderit fuga repudiandae repellat.

15. Prefijos CSS

Cuando una propiedad a aplicar está en modo borrador o pueden variar en la especificación definitiva, algunos navegadores implementan una serie de **vendor prefixes**, que facilitan la segmentación de funcionalidades.

De esta forma, podemos utilizar prefijos para asegurarnos que aunque las funcionalidades tengan un comportamiento o sintaxis diferente en cada navegador, podemos indicar el valor para cada uno de ellos por separado:

```
div{
  transform: ...; /* Navegadores que implementan especificación oficial */
  -webkit-transform: ...; /* Versiones antiguas de Chrome (Motor Webkit) */
  -moz-transform: ...; /* Versiones antiguas de Firefox (Motor Gecko) */
  -ms-transform: ...; /* Versiones antiguas de IE (Motor Trident) */
  -o-transform: ...; /* Versiones antiguas de Opera (Motor Presto) */
}
```

En el ejemplo anterior, vemos cómo aplicar la propiedad transform dependiendo del navegador que sea. Si queremos las funcionalidades que hay implementadas en cada navegador, podemos utilizar la web: [Can I Use](#), que te permite saber el estado actual, previo e incluso futuro de las propiedades CSS o elementos HTML en cada navegador.

16. Otras propiedades avanzadas de CSS

En CSS3 encontramos muchísimas más propiedades de la vistas a lo largo del curso, hay propiedades relacionadas con los [colores](#), [degradados lineales](#), [degradados radiales](#), [transformaciones](#), [transiciones](#), [animaciones](#) entre otras.

Además, recientemente hay una característica que no estaba en las versiones previas, como son las [variables de CSS](#) (por las que se popularizó preprocesadores Less o Sass que las incorporaban).