1. CSS: Ampliación Selectores Uniwebsidad

Tiene una tabla resumen con ejemplos en W3schools: http://www.w3schools.com/cssref/css_selectors.asp

1. Selectores de atributos

Además de los ya vistos en los apuntes de uniwebsidad tenemos:

- `[attribute^=value]` Comienza por.. Por ejemplo, enlaces seguros: `a[href^=https://]`
- `[attribute$=value]` Termina por.. Por ejemplo, enlaces pdf: `a[href$=".pdf"]`
- `[attribute*=value]` Contiene a.. `img[alt*="logo"]`

2. Selector sibling (~)

```
element1~element2 { /*  General sibling combinator*/
```

Este selector element1~element2 se cumple cuando el elemento2 está precedido por el elemento1. Ambos deben tener el mismo padre, pero el elemento2 no tiene porqué estar justo a continuación del elemento1.

La diferencia con el selector adyacente es esa, puede haber elementos intermedios entre elemento1 y elemento2.

3. Selectores estructurales `:nth`

3.1. `:nth-child(an+b)`

Selecciona los elementos (sin importar el tipo) que sean el hijo "n" de su padre. El valor de n puede ser un número, una palabra clave (odd o even) o una fórmula.

En el siguiente ejemplo se hace referencia al tercer elemento de listas no numeradas:

```
ul li:nth-child(3) {
    color: blue;
}
```

Teniendo en cuenta que el índice del primer elemento es uno, la palabras reservadas odd (impar) y even (par) se pueden usar en vez de un valor numérico. El siguiente ejemplo selecciona las filas pares de una tabla:

```
tr:nth-child (even) { /* Selecciona filas pares*/
    background-color: silver;
}
```

Con este selector tambien se puede usar la *fórmula* (an + b): a representa el tamaño del ciclo y b es el valor de desplazamiento inicial:

```
/* Selecciona cada tres filas, el segundo… */
tr:nth-child(3n+2) {
    background-color: silver;
}
```

3.2. `:nth-last-child(an+b)`

Similar al anterior, pero se empieza a contar por el último elemento del contenedor

3.3. Atajos `:nth-child`

3.3.1. `:first-child`

Equivalente a `:nth-child(1)`

```
p:first-child {
  background-color: yellow;
}
```

3.3.2. `:last-child`

Equivalente a `:nth-last-child(1)`

```
p:last-child {
  background: #ff0000;
}
```

3.4. `:nth-of-type(an+b)`

Numera los elementos hijos de un determinado tipo, y selecciona aquellos que cumplan estar en la posición "n". El valor de n puede ser un número, una palabra clave (odd o even) o una fórmula, de la misma forma que el selector nth-child().

Su uso es recomendado si en el padre, el contenedor, hay elementos de distintos tipos. Por ejemplo, si queremos de color plata los párrafos pares de un artículo, y como es posible que dentro del artículo haya imágenes, citas, cabeceras, usamos nth-of-type() en vez de nth-child()

```
article p:nth-of-type(even) {
    color: silver;
}
```

Tiene ejemplos de la diferencia entre nth-child y nth-of-type en lso ejemplos de clase, y adicionalemente en el fichero css_selectores_nth_ejemplo.html de la carpeta de ejemplos de clase.

3.5. `:nth-last-of-type(an+b)`

Similar a :nth-of-type pero comienza a contar por el último

3.6. Atajos `nth-of-type`

3.6.1. `:first-of-type`

Es equivalente a :nth-of-type(1), selecciona el primer elemento de ese tipo para cada padre.

```
p:first-of-type {
  background: red;
}
```

3.6.2. `:last-of-type`

Es equivalente a :nth-last-of-type(1).

```
p:last-of-type {
  background: #ff0000;
}
```

3.7. Otras posibilidades

3.7.1. A partir de un elemento

```
ul li:nth-child(n+3) {
    color: blue;
}
```

3.7.2. Hasta un elemento

```
ul li:nth-child(-n+3) {
    color: blue;
}
```

4. Sobre Especificidad CSS: resolviendo colisiones

En los apuntes de uniwebsidad no se detalla **cómo** se determina que un selector es más específico que otro, y a veces nos podemos llevar sorpresas.

Simplificando, será más específica

- La regla con más selectores de ID.
- En caso de empate la que tenga más selectores de clase (o de atributo o pseudo-clases).
- Y en caso de empate la que tenga más selectores de tipo (y pseudo-elementos).
- Y si sigue el empate, la última.

Por ejemplo

```
   .resumen { ...  }
```

es más específico que

```
   section > article  h2 + p {...  }
```

> Nota: Recuerde que Visual Code le muestra la especificidad de un selector al situar el ratón sobre el mismo.

Aquí tiene algunos enlaces adicionaless:

- Calculadora: https://specificity.keegan.st/
- Una breve explicación en castellano: https://blog.outbook.es/desarrollo-web/css-especificidad-de-los-selectores

- Lo que dice la norma: https://www.w3.org/TR/selectors-3/#specificity

Colores

Opacidad

La propiedad CSS `opacity' define la transparencia de un elemento, esto es, en qué grado se superpone el fondo al elemento.

```
p {
  color: blue;
  opacity: 0.4;  /* valor entre 0 y 1 */
}
img.difumina {
  opacity: 0.2;
}
```

Es un número cuyo valor se encuentra entre 0.0 y 1.0, ambos incluidos. Este valor representa la opacidad.

- 0: el elemento es transparente (invisible).
- Cualquier valor entre 0 y 1: El elemento es translúcido.
- 1: el elemento es opaco (sólido).

Colores hsl

HSL es el acrónimo de hue(matiz), saturation (saturación), lightness (luminosidad).

- El valor del matiz se refiere a la frecuencia dominante del color dentro del espectro visible. Es un valor entre 0 y 360 (una rueda de color): rojo vale 0, verde 120, azul 240.
- La saturación se refiere a la intensidad del color, a la «pureza» de éste. 0% means a shade of gray and 100% is the full color.
- La luminosidad es la cantidad de luz de un color. S y L son porcentajes entre 0 y 100%: 0% is black, 100% is white.

```
section {
    background-color: hsl(180, 50%, 50%);
    color: hsl(0, 0%, 100%);
}
```

Usar HSL hace más sencillo crear armonías de colores variando alguno de los tres valores: paletas monocromáticas, paletas de colores análogos, complementarios, triadas, etc. Puedes hacerte una idea en

estas webs:

- http://www.colorhexa.com/,
- http://htmlcolorcodes.com/color-picker/
- http://paletton.com/

Añadir opacidad al color: hsla y rgba

La opacidad se puede especificar al mismo tiempo que el color. Para ello se usan las propiedades rgba, y hsla, que reciben un 4ºvalor, la opacidad. Ejemplos:

```
p.difumina {
  color: rgba(0, 0, 255, 0.5);
}

h2 {
  color: hsla(0, 100%, 50%, 0.5);
}
```

> Nota: incluso es posible encontrarlo en representación hexadecimal, añadiendo un cuarto octeto para la opacidad:

```
p.difumina {
  color: #0000FF80;
}
```

1. Más sobre box model CSS

1.1. Border-radius

Redondea las esquinas del border de un elemento. Se pueden especificar entre 1 y 4 valores y se interpretan de la misma forma que en el `padding` o `margin`:

```
section {
  border: 1px solid blue;
  border-radius: 15px; /* se pueden especificar 1, 2, 3 o 4 valores */
  padding: 15px;
}
```

https://css-tricks.com/almanac/properties/b/border-radius/

1.2. Box-shadow

Se especifica el desplazamiento horizontal, el vertical, el "blur" o desenfoque y el color de la sombra:

```
section {
  border: 1px solid blue;
  box-shadow: 40px 30px 10px #444;
  margin: 40px;
}
```

Incluso se pueden especificar varias sombras (separadas por comas):

```
#example1 {
  border: 1px solid;
  padding: 10px;
  box-shadow: 5px 5px blue, 10px 10px green;
}
```

1.2.1. text-shadow

No tiene nada que ver con el modelo de cajas, pero añadimos esta propiedad en este documento por ser su uso muy parecido a `box-shadow`:

```
h1 {
  text-shadow: 2px 2px 1px #666;
}
```

1.3. box-sizing

Permite que al tener en cuenta la propiedad `width` (y `height`) se incluya el valor del padding y el border juto con el del contenido (no el margen). Esto permite aplicar anchos por ejemplo del 50% sin tener que considerar la anchura del padding o del borde. Es posible aplicarla todos los elementos con el operador universal:

```
* {
  box-sizing: border-box;
}
```

o sólo aplicarla a los elementos necesarios en cada caso, por ejemplo:

```
header, nav, aside, footer, section {
  box-sizing: border-box;
}
```

+info: https://www.w3schools.com/cssref/css3_pr_box-sizing.asp

1.4. Multiple Backgrounds

Se pueden especificar varias imágenes de fondo en un mismo elemento. Para ello basta con usar la propiedad `background` con los valores de cada una de ellas separados por comas. La imagen que se especifica primero es la que aparece más cercana al usuario:

```
#example1 {
    background: url(img_flwr.gif) right bottom no-repeat,
                url(paper.gif) left top repeat;
}
```

O especificando cada una de las propiedades que agrupa `background`:

```
body {
    background-image: url("img_tree.gif"), url("paper.gif");
    background-repeat: no-repeat, repeat;
```

```
    background-color: #cccccc;
}
```

En este enlace tiene un ejemplo de uso:

- https://www.w3schools.com/css/tryit.asp?filename=trycss3_background_multiple

Ampliación T06 Uniwebsidad: CSS y texto

1. text-decoration

En la web de Uniwebsidad no queda claro el uso de `text-decoration`. Realmente es un *shorthand* de varias propiedades. Además permite modificar el color, estilo y el grosor.

```
section strong {
  /* ejemplo inicial */
  /* text-decoration: underline;  */

  /* ejemplo "ampliado" */
  text-decoration: red solid underline 2px;
  /* que equivale a
        text-decoration-color:red ;
        text-decoration-style: double;
        text-decoration-line: underline;
        text-decoration-thickness: 2px */
}
```

2. Web-fonts

En la carpeta de fuentes existe una carpeta de nombre `t06_css_webfonts` donde se explica y aplica el uso de web fonts y algunas propiedades adicionales.

Tenemos dos formas de incorporar fuentes adicionales en nuestro fichero CSS:

- Fuentes en un fichero local

- Fuentes web u online

2.1. Fuente en fichero local

Si disponemos de un fichero con la fuente a utilizar se necesita

- Una directiva `@font-face` (en el fichero CSS) que le da un nombre a la nueva fuente y asocia dicho nombre el fichero.
- Luego usaremos esa fuente con el nombre indicado en el paso previo

```
@font-face {
  font-family: fNovedades; /* asociamos el nombre fNovedades al fichero siguinte */
  src: url("mona_shark/Mona Shark.otf");
}

.novedades {
  font-family: fNovedades, serif;  /* mismo nombre usado en @font-face font-family */
}
```

2.2. Fuentes web

Para incluir fuentes de Google (o de otros proveedores) accedemos a la url https://fonts.google.com/ que nos ofrece esas fuentes.

Si pulsamos sobre una fuente nos da la opción de descargarla (para usarla como en el caso anterior). Nosotros en este caso vamos a seleccionarla (*Select this Style*) y nos aparece a la derecha las instrucciones para utilizarla en nuestra página sin tener que descargarla. Por ejemplo, seleccionada la fuente `Ole` debemos dar dos pasos:

1. Incluir los enlaces que nos propone en el head del fichero HTML. Serán similares a estos:

   ```
   <link rel="preconnect" href="https://fonts.googleapis.com">
   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
   <!-- Aquí van las fuentes -->
   <link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&family=Roboto&display=swap" rel="stylesheet">
   ...
   <!-- hoja de estilos del usuario -->
   <link href="misestilos.css" rel="stylesheet">
   ```

2. Y usar la propiedad `font-family` en el fichero `misestilos.css` como se observa en este ejemplo

   ```
   p {
       font-family: 'Ole', cursive;
   }

   h2 {
   ```

```
    font-family: 'Roboto', sans-serif;
}
```

2.3. Enlaces relacionados

- Google Fonts: https://fonts.google.com/
- Fuentes:
  - CDN Fonts: https://www.cdnfonts.com/
  - The League of Moveable Type: https://www.theleagueofmoveabletype.com/
  - Font Space: https://www.fontspace.com/

3. pseudoelementos

En el tema 6 se introducen los pseudo elementos `::first-line` y `::first-letter`. Completamos ese apartado con otros dos, `::after` y `::before`, con la propiedad `content` y el método `attr()`.

Por lo tanto en este partado estudiaremos:

- `::after`
- `::before`
- `content`
- `attr()`

3.1. `::after`, `::before` (y `content`)

Para insertar contenido generado desde la hoja de estilo delante (`::before`) o detrás (`::after`) de un elemento debemos usar la propiedad `content`.

En este ejemplo se añade delante de cada cabecera de nivel 1 el texto "Capítulo -"

```
h1::before {
  content: "Capítulo - ";
}
```

3.2. `content` y `attr()`

Además podemos usar `content` junto con la función `attr()` para mostrar el valor de un atributo del elemento (que de otra forma no se muestra en pantalla). Por ejemplo se puede usar para mostrar el significado de las abreviaturas, elemento `abbr` con atributo `title` :

```
abbr::after {
    content: " (" attr(title) ")  "
}
```

O mostrar la URL de los enlaces tras el texto del enlace.

```
a[href]:after {
    content: " (" attr(href) ") ";
```

```
}
```

Observe que en este último ejemplo se ha usado `:after` (obsoleto) en vez de `::after`. La notación `::` para los pseudo elementos se añadió más tarde y los navegadores soportan las dos para mantener compatibilidad hacia atrás.

4. Hojas de estilo para impresión

Los estilos para impresión se pueden añadir de dos formas:

- Añadiendo un fichero CSS con los estilos que se deben aplicar al imprimir, y al enlazarlo indicar que el destino (`media`) es la impresora. Los valores más usados para `media` son `all` (valor por defecto), `screen` (pantalla) y `print` (impresora).

  ```
  <link rel="stylesheet" href="general.css">
  <link rel="stylesheet" href="impresora.css" media="print">
  ```

- Añadiendo los estilos de impresión en el fichero de estilos general usando la directiva `@media print{}`. En este caso quedaría:

  ```
  @media print {
      a[href]:after {
        content: " (" attr(href) ") ";
      }
  }
  ```

Para ver el resultado y las reglas CSS que se aplican en el inspector de elementos (F12 o CTRL+MAY+I) debe acceder al mismo y pulsar sobre el **icono "alternar simulación de medios de impresión para la página"** disponible en la zona de estilos.

Otra opción para ver cómo se aplican los estilos de impresión es usar la opción de imprimir, aunque debe tener precaución porque los navegadores permiten configurar algunas opciones de estilos, como por ejemplo los colores o imágenes de fondo, que por defecto aparecen desactivadas para que no se muestren.

Puede encontrar más detalles en este enlace: https://css-tricks.com/can-you-view-print-stylesheets-applied-directly-in-the-browser/

5. Contadores CSS

Tiene varios ejemplos en el directorio `t06_css_contadores`

5.1. Uso básico

El uso de contadores se basa en el pseudo elemento `::before` (o `::after`), la propiedad `content` para insertar el contador y el uso de estas nuevas propiedades:

- `counter-reset` - crea o reinicia un contador
- `counter-increment` - incrementa el valor de un contador
- `counter()` - muestra el valor de un contador.

Por ejemplo, si desea numerar los párrafos de una sección (ver ejemplo `css_contadores_01.html`)

```
section { /* modificar section/article para ver la diferencia */
    counter-reset: np;
}
section p {
    counter-increment: np;
}
section p::before {
    content: "(" counter(np) ") ";
}
```

5.2. Contadores en cabeceras

Si va a numerar las cabeceras de nivel 2, debe iniciar el contador en una regla CSS asociada al elemento padre de esas cabeceras, e incrementarlo y mostrarlo en una propiedad del elemento a numerar usando la pseudoclase ::before

Por ejemplo, si quiere numerar las cabeceras de nivel 2 de una sección de la clase `unnivel`

```
section {
    counter-reset: nh2;
}
section h2 {
    counter-increment: nh2;
}
section h2::before {

    content: "Apartado " counter(nh2) ": ";
}
```

5.3. Contadores en listas anidadas

Si tenemos listas anidadas

```
<ol>
    <li>Lorem.</li>
    <li>Quaerat
        <ol>
            <li>Lorem, ipsum.
                <ol>
                    <li>Lorem, ipsum dolor.</li>
                    <li>Assumenda, minus ipsam!</li>
                </ol>
            </li>
            <li>Ipsam, quasi!</li>
            <li>Fuga, commodi.</li>
        </ol>
    </li>
    <li>Sit.</li>
</ol>
```

podemos usar la función `counters` (ojo a la `s` final). Esta funciñon recibe dos argumentos: el contador y el separador a usar si hay varios contadores con el mimos identificador:

```
ol {
    counter-reset: nli;
    list-style-type: none;
}
li::before {
    counter-increment: nli;
    content: counters(nli,".") " ";
}
```

5.4. Ejercicios

1. Revisar ejemplos de código proporcionados en la plataforma
2. Hacer ejercicio propuesto en ejemplos de código proporcionados en la plataforma

5.5. Enlaces de ampliación

Puede obtener más información en:

- W3Schools: Contadores CSS https://www.w3schools.com/css/css_counters.asp
- MDN: Using CSS counters https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Counter_Styles/Using_CSS_counters
- Ejemplos https://www.smashingmagazine.com/2013/04/css-generated-content-counters/

6. Fuera de temario: Familias de fuentes en Linux

6.1. Enlaces:

- https://geekland.eu/instalar-fuentes-en-debian-linux/
- https://h4ckseed.wordpress.com/2022/01/31/how-to-gestion-de-fuentes-en-gnu-linux/

6.2. Instalar las fuentes de Microsoft o variantes

En el caso que queramos o tengamos necesidad de instalar algunas de las fuentes de *Microsoft*, tan solo tenemos que abrir una terminal y ejecutar el siguiente comando:

```
sudo apt-get install ttf-mscorefonts-installer
```

Una vez instalado el paquete `ttf-mscorefonts-installer` pasaremos a disponer de las siguientes fuentes:

> Andale Mono, Arial Black, Arial, Comic Sans MS, Courier New, Georgia, Impact, Times New Roman, Trebuchet, Verdana, Webdings.

La totalidad de fuentes mencionadas son fuentes privativas que se hallan en el repositorio Contrib de Debian.

Existen variantes libres de las tipografías Times, Arial y Courier, etc. Se pueden instala el paquete `fonts-liberation` ejecutando el siguiente comando en la terminal:

```
sudo apt-get install fonts-liberation
```

6.3. Gestionar las fuentes

Puede usar `font-manager` y sólo tienen que ejecutar el siguiente comando en la terminal:

```
sudo apt-get install font-manager
```

En el caso que precisen sacar un listado de la totalidad de fuentes instaladas, pueden ejecutar el siguiente comando en la terminal:

```
fc-list
```

> TODO: Algunas fuentes tiene un mapeado, por ejemplo Times a Nimbus Roman o Arial a Liberation Sans.

Amplia t12 Uniwebsidad: Layout HTML con CSS

1. Código ejemplos *layout*

Los ejemplos están disponibles en el directorio `t12_01_layout`. Para probarlos basta modificar el fichero CSS enlazado en el fichero `layout.html`:

- Usando grid: `grid.css`
- Usando grid con nombres de áreas: `grid_areas.css`

En la carpeta `otros` tiene otras posibilidades que no son parte de los contenidos evaluables:

- Usando flex: `flex.css` (no tanto para layout sino para el flujo en una única línea)
- Usando float: `float.css` => obsoleto pero se encuentra todavía en uso.
- Usando display: table-cell: fichero `table-cell.css` => obsoleto
- Usando display: inline-blocks fichero `inline-blocks.css` => obsoleto

Por lo tanto en este tema **nos vamos a centrar en el uso de grid y grid areas**.

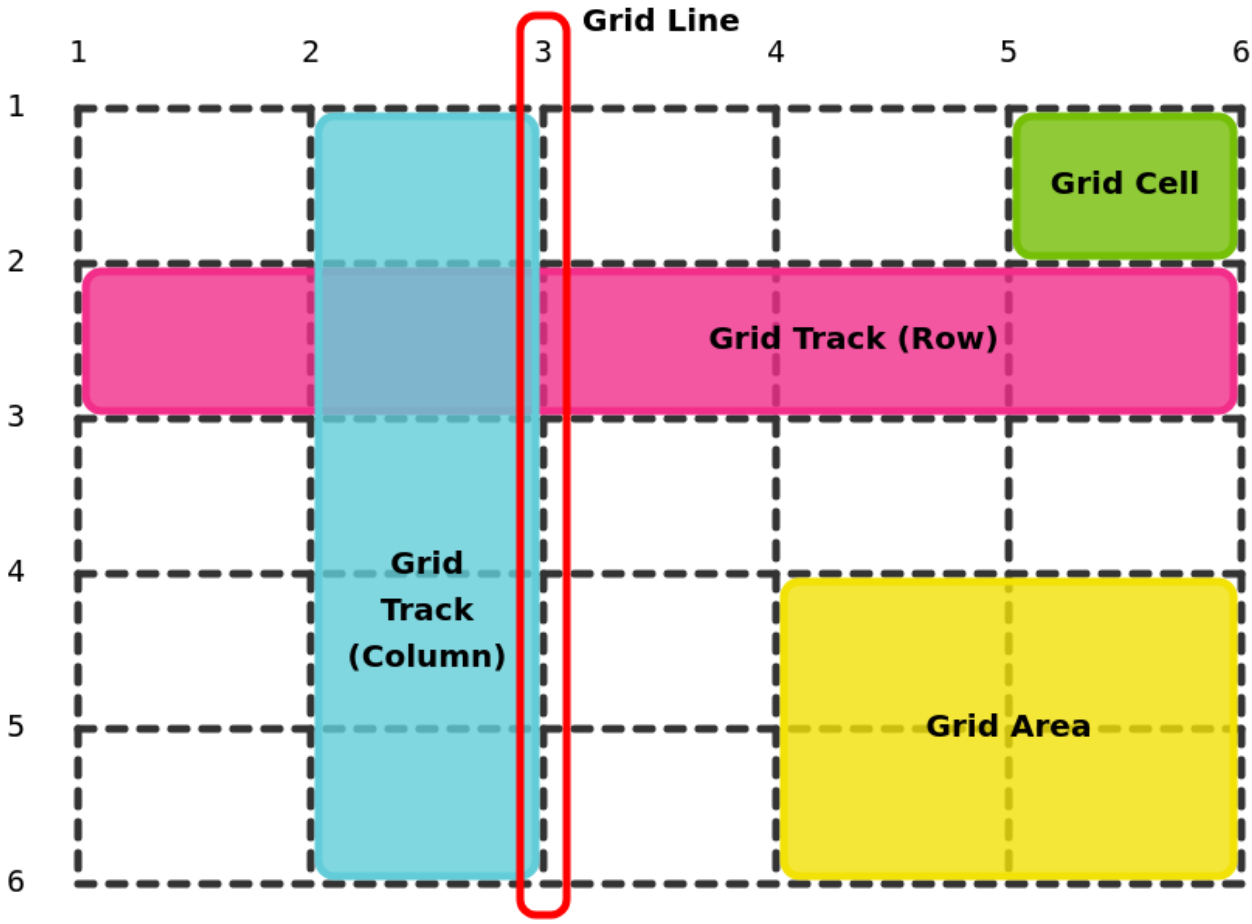
2. *Layout* con grid

Al aplicarle la propiedad `display:grid` a un elemento HTML lo convertimos en un grid o rejilla. Por defecto esa rejilla tiene una única columna, y **cada elemento hijo de la rejilla** se dispone en una celda (en el orden que aparecen en el fichero HTML).

Con la propiedad `grid-template-columns:` se define el número y tamaño de las columnas de la rejila. Las líneas que delimitan la rejilla se numeran a partir de 1. Si hemos definido 3 columnas, las líneas irían de la 1 a la 4. Esos números de líneas se usan en las propiedades `grid-column` y `grid-row` para indicar:

- donde comienza / donde termina una celda (p.e. `grid-column: 1 / 4;` indica que comienza en la línea 1 y termina en la 4)
- o dónde comienza y a cuantas cuadriculas se expande una celda (p.e `grid-column: 1 / span 3;` significa que comienza en la línea 1 y ocupa el espacio de tres celdas)

En la siguiente figura puede observar una rejilla y marcadas las líneas, celdas, filas y columnas.



En el código de ejemplo tiene una explicación de:

- dónde definir el grid (`display: grid`)
- cómo definir el número de columnas (`grid-template-columns`) con porcentajes o con la unidad `fr`.
- Y como situar cada bloque en el grid (propiedades `grid-column` y `grid-row`), ya sea dejando su posición por defecto, posicionando con los número de líneas o usando `span` para ocupar más de una celda.

Veamos un ejemplo en el fichero `grid.css` para definir una rejilla de tres columnas. La propiedad `display:grid;` se aplica en el elemento padre de los elementos que queremos disponer en rejilla, que en el caso del ejemplo es un bloque `<div>` con `id="contenedor"`:

```
/* rejilla 3 columnas: líneas de columna de la 1 a la 4 */
#contenedor {
  display: grid;
  grid-template-columns: 20% 50% 30%;  /* tres columnas y reparto tamaños
columnas */
}
```

Y observe el posicionamiento de los elementos HTML hijos de ese `#contenedor`:

- Se ha movido el `footer` a la primera fila con `grid-row: 1;`
- Se ha modificado el tamaño de `footer` y `header` para que ocupen tres celdas con `grid-column: 1 / span 3` o `grid-column: 1 / 4;`

- Al resto de elementos hijos no se les ha aplicado ninguna propiedad por lo que ocupan una única celda siguiendo el orden en el que aparecen en el contenedor.

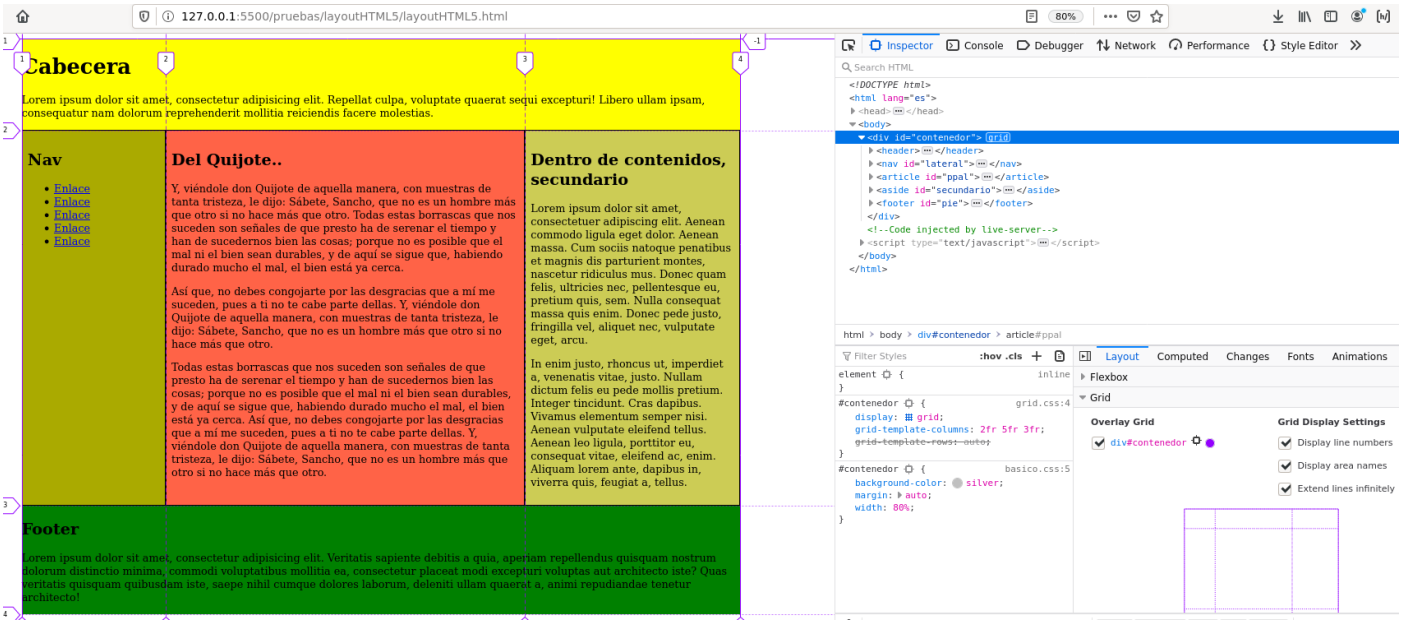
```
footer {
  grid-row: 1; /* modifica la fila dónde se posiciona footer */
  grid-column: 1 / span 3;  /* equivale a 1 / 4 */
}

header {
  grid-column: 1 / 4;
  /* equivale a
        grid-column-start: 1;
        grid-column-end: 4;   */
}
```

Es posible usar en la plantilla de columnas (`grid-template-columns`) la unidad `fr` (fraction), que simboliza una fracción de espacio total del grid.

```
#contenedor {
  display: grid;
  grid-template-columns: 2fr 5fr 3fr;  /* tres columnas y reparto tamaños
columnas */
}
```

Puede activar la visualización del grid en el inspector de elementos del navegador, pestaña layout, y ver las celdas, número de líneas, etc. Se puede apreciar en la siguiente figura:



2.1. Grid areas

Se facilita un fichero de ejemplo (`grid_areas.css`) con el uso de la propiedad `grid-template-areas`. Esta propiedad permite **definir una plantilla de la rejilla con nombres simbólicos**. Además hay que asociar esos nombres de áreas a elementos HTML con la propiedad `grid-area` .

```
#contenedor {
  display: grid;
  grid-template-columns: 2fr 5fr 3fr;
  grid-template-areas: /* definimos la plantilla de áreas */
    "cabecera cabecera   barra"
    "ads      articulo   barra"
    "ads      footer     footer";
}
/* se asocian áreas de la plantilla a elementos HTML */
header {
  grid-area: cabecera;
}
footer {
  grid-area: footer;
}
...
```

Los nombres de área los define el desarrollador y deben ser significativos (en ocasiones se usa como nombre de área el nombre del elemento HTML).

Con grid areas mover los elementos de posición o hacer que ocupen más de una celda es simplemente hacer cambios en la plantilla de `grid-template-areas` (y en `grid-template-columns` si cambia el número de columnas). No es necesario trabajar con los números de fila y columna. Esta forma es menos sensible a errores, y mucho más legible. Observe en el siguiente ejemplo como se modifica el layout simplemente modificando el *template*:

```
#contenedor {
  display: grid;
  grid-template-columns: 2fr 5fr 3fr;
  grid-template-areas: /* definimos BUEVA plantilla de áreas */
    "barra   cabecera  cabecera "
    "barra   articulo  articulo"
    "footer  footer    ads";
}
```

2.2. Grid anidados

A veces es interesante tener un grid dentro de una de las celdas del grid. Tiene ejemplos de esta situación en el código de la carpeta `grid_anidados`:

- Ejemplo 01: el elemento nav es una celda del grid *principal*. La lista de enlaces de ese nav se ha definido como grid para disponer sus elementos en horizontal.
- Ejemplo 02: el elemento footer es una celda del grid *principal*. Al mismo tiempo se ha definido el footer como grid para disponer las secciones que hay dentro en columnas.

3. Enlaces de ampliación

- CSS Grid:

    - https://lenguajecss.com/p/css/propiedades/grid-css
    - https://css-tricks.com/snippets/css/complete-guide-grid/
    - https://gridbyexample.com/
    - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout

- Flex (o flexbox):

    - https://www.w3schools.com/css/css3_flexbox.asp
    - https://css-tricks.com/snippets/css/a-guide-to-flexbox/

1. Diseño Adaptativo (RWD)

1. Introducción

Antes de seguir, el significado de *RWD*: **Responsive Web Design**. El objetivo es adaptar nuestros contenidos y presentación al dispositivo que utilice en cada momento el usuario. Se suele tener en cuenta el ancho de la pantalla, pero también podría considerar la orientación, etc.

Hay tres aproximaciones posibles:

- Mejora progresiva o **“First Mobile”**: iniciar el diseño desde el layout de un móvil e ir "mejorando" hasta llegar a layouts tipo escritorio. La anchura disponible va creciendo. Esta aproximación es la más actual y la que usaremos.
- Degradación progresiva: justo al revés.
- Crear layouts totalmente independientes para cada escenario. Más díficil de mantener

Para usar RWD en nuestra página HTML necesitamos

- Usar una directiva de CSS, las conocidas como *CSS Media Query*.
- Introducir en la cabecera de la página la etiqueta `meta viewport` para que los contenidos funcionen correctamente en móviles, tablets, etc. Esto es algo que ya hacía nuestro editor por defecto.
- Definir los **puntos de ruptura (breakpoints)** en los cuales el layout cambia,

2. Código de ejemplos

Tiene disponible todos los ejemplos de este documento en la Unidad Compartida, `carpeta t12_RWD`. Para ver cómo se adapta el diseño al ancho puede:

- Reducir el ancho de la ventana del navegador
- o quizás más cómodo: abrir el inspector de elementos y jugar con el ancho con la barra que separa la ventana de visualización de la de desarrollo.

3. CSS2 Media Types

Antes de ver las *Media Queries* es frecuente presentar a sus predecesoras, las *Media Types*.

La directiva `@media` de CSS permite aplicar diferentes estilos a diferentes medios, principalmente `screen` y `print`. Si tenemos un fichero de nombre `impresora.css` y añadimos reglas CSS dentro de la directiva

`@media print` hacemos que esas reglas CSS sólo se apliquen si la página se va a imprimir:

```
@media print {
  body {
    font-size: 10pt;
  }
  aside {
    display: none;
  }
}
```

Otra posible opción ya comentada en temas previos es indicar el destino (`print|screen|all`) al enlazar el fichero CSS, y todo el fichero se vería afectado por esa restricción:

```
<link rel="stylesheet"  media="print" href="impresora.css" />
```

Y en el fichero `impresora.css` no es necesaria la directiva `@media print`, simplemente las reglas CSS:

```
/* impresora_mod.css: No hace falta la directiva @media print:
   se enlaza especificando  media="print" */
body {
  font-size: 10pt;
}
aside {
  display: none;
}
```

4. CSS3 Media Queries

Las *Media Queries* extienden la idea de las *media types*: en vez de mirar sólo el tipo de dispositivo se pueden consultar otras características del dispositivo: anchura, altura, orientación, etc.

4.1. Sintaxis de una Media Query

Se añade a la sintaxis de una *media type* una o más expresiones, todas las cuales dan como resultado un valor lógico. El resultado es verdadero si el tipo coincide y la expresiones son verdaderas. Si es verdadera, las reglas de estilo incluidas en la directiva *media query* se aplican.

En el siguiente ejemplo las reglas CSS se aplican si:

1. el destino de la página es la pantalla y
2. la anchura disponible es mayor de 480px (de al menos 480px).

```
@media screen and (min-width: 480px) {
  /* Reglas CSS */
}
```

El tipo del media (`screen` en el ejemplo previo) es opcional: si no se incluye su valor por defecto es `all`. Por lo tanto estas dos directivas son equivalentes:

```
@media (orientation: portrait) {
   …
}

/*es equivalente a */
@media all and (orientation: portrait) {
  …
}
```

4.2. Ejemplos basicos

Vemos tres ejemplos de uso que además puede probar en los siguietes enlaces

1. Modificar el color de fondo al alcanzar una anchura mínima (`min-width`): https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_mediaquery. El código CSS necesario sería:

   ```
   /* se aplica siempre */
   #contenedor {
     background-color: lightblue;
   }
   /* se aplica si ancho >480px. Entra en colisión con la anterior y se
   aplica esta última por tener misma especificidad pero aparecer después
   */
   @media screen and (min-width: 480px) {
     #contenedor {
       background-color: lightgreen;
        }
   }
   ```

2. Ahora el cambio de color depende de la orientación del dispositivo (`orientation: landscape`): https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_mediaquery_orientation

   ```
   @media only screen and (orientation: landscape) {
       body {
         background-color: lightblue;
       }
   }
   ```

3. Modificar el layout: Si el ancho es suficiente el menú de navegación pasa a la izquierda del contenido: https://www.w3schools.com/css/tryit.asp?filename=trycss3_media_queries2

   ```
   @media screen and (min-width: 480px) {
     #leftsidebar {
       width: 200px;
   ```

```
      float: left;
    }
  }
```

5. Layout CSS y RWD

Nuestro objetivo es modificar el layout de la página para que se adapte al ancho disponible en el dispositivo. Y como se indicaba en la introducción, usaremos un diseño **First Mobile**.

En el diseño **First Mobile** se parte de un diseño apilado (*stacked*) para móvil, y se va añadiendo puntos de ruptura usando *media queries* con `min-width`.

Importante: En cada salto o punto de ruptura

- se mantienen las reglas CSS de los layouts previos
- se pueden añadir nuevas reglas CSS a las del layout previo
- se pueden modificar propiedades del layout previo (y al estar definidas más tarde en el fichero son las que se aplican si hay colisión).

Por eso es importante el orden en el que aparecen las *media queries* (los puntos de ruptura), de menor a mayor.

5.1. Ejemplo 01: RWD con grid y 2 puntos de ruptura

En este ejemplo, disponible en los ficheros `grid-rwd.html` y `grid-rwd.css` fijaremos dos puntos de ruptura, a 580px y 992px. Esto provoca tres layouts distintos:

- uno con ancho hasta 580px.
- otros con ancho mayor que 580px pero menor que 992px
- y el último con ancho mayor que 992px

```
/* grid-rwd.css */
/* layout 01 : No se indica nada, apilado por defecto */

/* layout 02:  */
@media (min-width: 580px) {
  #contenedor {
    display: grid;
    grid-template-columns: 1fr 2fr;
  }
  header {
    grid-column: 1 / span 2;
  }
  aside, footer {
    grid-column: 1 / span 2;
  }
}
/* layout 03: tener en cuenta que se aplica
   todo lo anterior   más lo que se define aquí */
@media (min-width: 992px) {
  #contenedor {
    /* no es necesario especificar display:grid,
       ya especificada en layout 2*/
```

```
    grid-template-columns: 1fr 3fr 2fr;
  }
  header, footer {
    grid-column: 1 / span 3;
  }
  aside {
    grid-column: 3;
  }
}
```

En el ejemplo puede observar:

- Hay colisiones en el color de fondo de `header`, y se aplica el del último layout activo.
- En el layout 3 se modifica el número de columnas, y como hay colisión con el previo se aplica el último.
- En el layout 3 no es necesario especificar la propiedad `display: grid;` ya que está activa del layout previo.

5.2. Ejemplo 02: RWD con `grid areas`

Puede acceder al código en los ficheros:

- `layout_rwd_ga.html` y
- `layout_rwd_ga.css`

En este ejemplo se usan grid con nombres de áreas. En RWD suele ser más sencillo y legible usar áreas, ya que sólo es necesario en cada punto de ruptura modificar la plantilla de áreas (y el número o tamaño de las columnas).

Por ejemplo en el segundo layout del ejemplo tenemos dos columnas y la plantilla:

```
/* Segundo layout: si el ancho es 600 o más..*/
@media screen and (min-width: 600px) {
  #contenedor {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-areas:
      " footer footer "
      " nav    section"
      " aside  section"
      " header header";
  }
```

Y en el siguiente layout tres columnas y una plantilla diferente:

```
@media screen and (min-width: 900px) {
  #contenedor {
    grid-template-columns: 1fr 2fr 2fr;
    grid-template-areas:
    " aside  header   nav "
    " aside  section  section"
    " aside  footer   footer";
```

```
    }
}
```

Realizar esa misma distribución usando `grid-colum` y `grid-row` es mucho más complejo, propenso a errores, y además menos legible.

5.3. Ejemplos "fuera de temario"

- Ejemplo de uso de RWD con Bootstrap. El código está disponible en `otros/layoutHTML5-BS-RWD.html`.
- Ejemplo práctico de cómo ir mejorando una lista de enlaces con direcciones de correo electrónico a medida que aumenta el tamaño disponible: https://css-tricks.com/css-media-queries/.
- Galería de tarjetas: uso de `grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));`. El código está disponible en el fichero `otros/grid-rwd-galeria.html`.

6. Enlaces de consulta adicional

- Qué dice Google sobre RWD: https://web.dev/responsive-web-design-basics/.
- Explicando el elemento `meta viewport`: https://dev.opera.com/articles/an-introduction-to-meta-viewport-and-viewport/.