¿Qué es MediaQuery?

Son unas configuraciones que nos permiten aplicar *estilos* específicos según las características que le digamos. En este caso: **El tamaño de la pantalla**.

Según se vaya haciendo más chiquita, vamos a ir cambiando el orden de los elementos. (Según sea más chica porque estamos asumiendo que se va a ir viendo en dispositivos con pantalla más chica que una computadora, como un celular).

Como por ejemplo, en esta imagen se ve que configuraron la página para que cuando la pantalla fuera más delgada, los cuadros se acomodaran como una lista vertical. En lugar de un cuadro horizontal.

Y ya al final se ve como una sola fila de "Imagen y texto".

Esa forma de hacer páginas se le dice **Diseño responsivo**.

Suena más difícil de lo que es, no te asustes. La mitad de la dificultad está en planear cómo van a acomodar las cosas. Programarlo es relativamente fácil. Aquí tengo un ejemplo donde:

```
@media (max-width: 600px) {

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

El color de fondo del cuerpo (**body**) se cambia a azul claro (**lightblue**) solo cuando el ancho de la pantalla es de 600 píxeles o menos.

Ejemplo 1 de Media-Query

Página que cambia de colores

Esto va en tu HTML

```html
<html>
<!-- Aquí es la cabecera de la página -------------------------- -->
    <head>
        <link rel="stylesheet" href="Estilo.css">
    </head>
<!-- Aquí empieza el cuerpo de la página ----------------------- -->
    <body>
        <div class="container">
            <h1>¡Hola Mundo!</h1>
            <p>Este es un ejemplo de uso de Media Queries en CSS.</p>
        </div>
    </body>

</html>
```

Aquí sólo estamos haciendo un contenedor con un título y un párrafo. El título es un **h1** que dice "¡Hola mundo!", y el párrafo dice… Lo que dice. No importa mucho en realidad. También estamos referenciando con **link** nuestra página css, que en este caso se llama: `"Estilo.css"`

Esto va en tu CSS

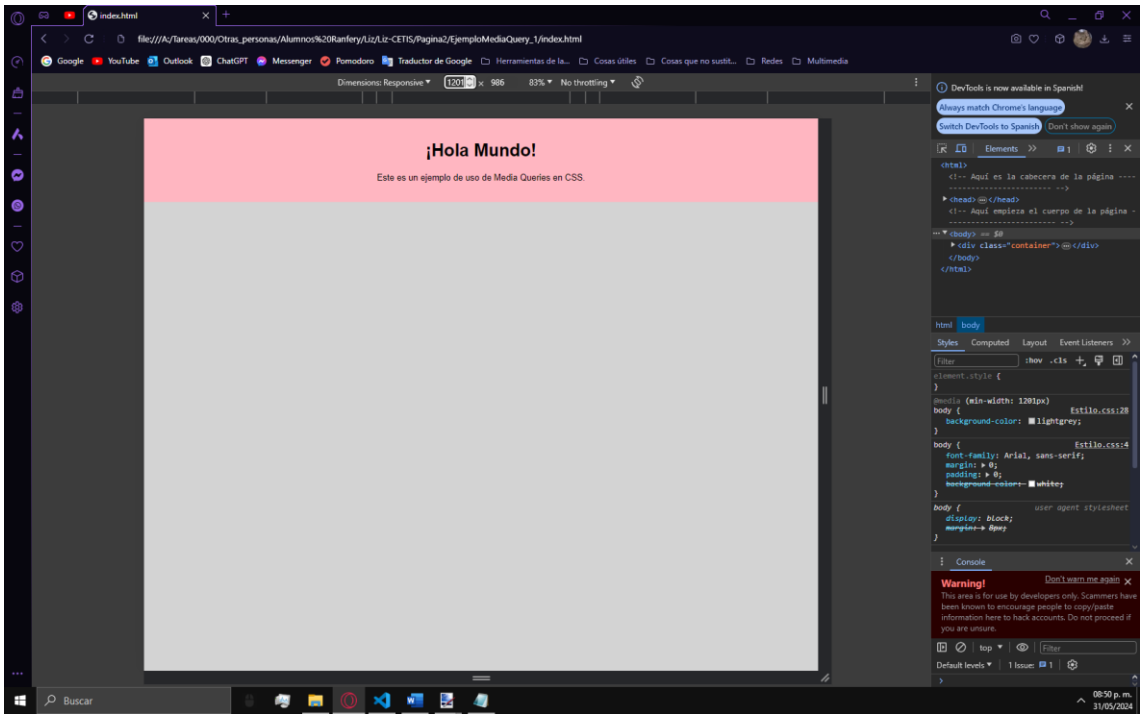
```css
/* Aquí van las configuraciones de toda la página
Antes de cambiarle tamaños y todo, no hay mucho que
especificar. */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: white;
}
.container {
    width: 100%;
    padding: 20px;
    text-align: center;
}
/* ------------------------------------------------------ */
/* Estilo por si la página mide entre 601 y 1200 pixeles*/
@media (min-width: 601px) and (max-width: 1200px) {
    body { /*Se le pone el fondo amarillo*/
        background-color: lightyellow;
    }
    .container { /* Y la cabecera verde */
        background-color: lightgreen;
    }
}
/* ------------------------------------------------------ */
/* Estilo por si la página mide más de 1200 pixeles */
@media (min-width: 1201px) {
    body { /*Se le pone el fondo gris*/
        background-color: lightgrey;
    }
    .container { /* Y la cabecera rosa.*/
        background-color: lightpink;
    }
}
```

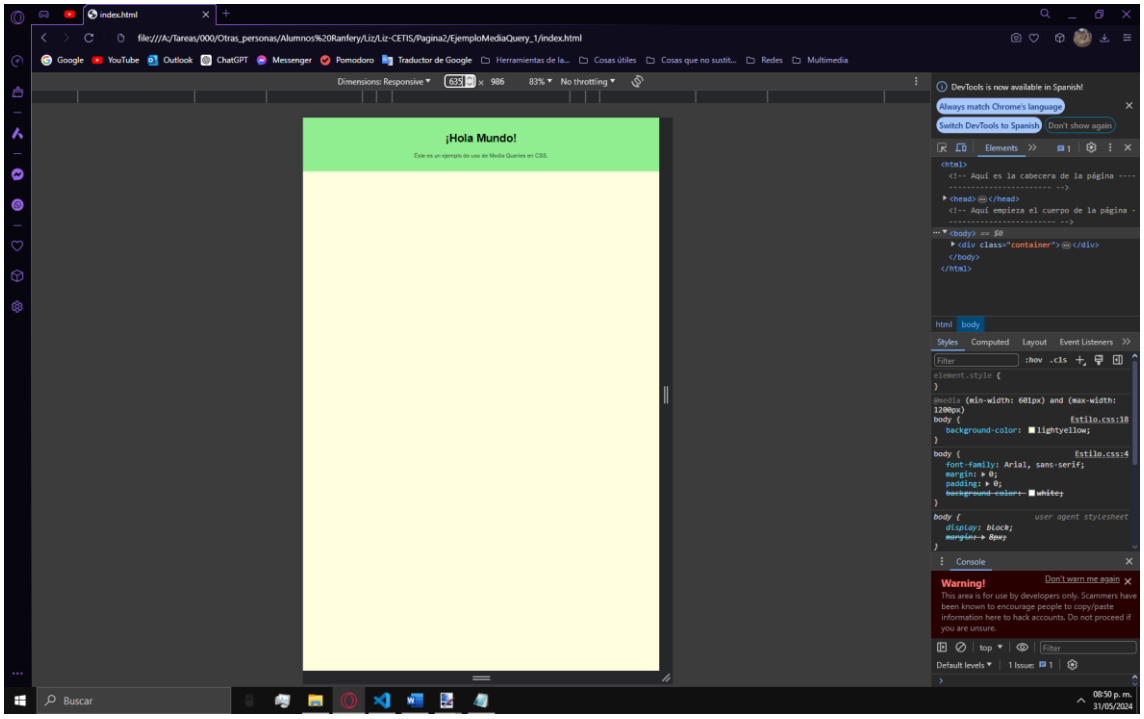
Y aquí estamos definiendo dos configuraciones de MediaQuery. Una por si la pantalla mide cierto ancho, que sería el tamaño de una pantalla mediana.

Y una por si la pantalla es más grande.

En este ejemplo específico sólo se cambian los colores según el tamaño. Pero después haremos cosas más útiles.

También usaremos más de dos configuraciones MediaQuery.

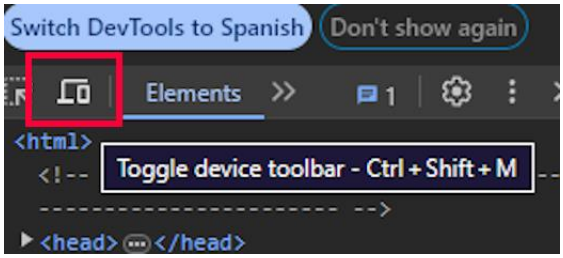




Estas capturas de pantalla deberían representar bien el cambio de estas dos configuraciones MediaQuery.

Para acceder a esta vista dinámica de tamaños en tu navegador(El que sea, en mi caso es OperaGX pero perfectamente funciona en Edge, Brave, Firefox etc).

1. Click Derecho (Donde sea)
2. “Inspeccionar Elemento”
3. Darle a este ícono



Ejemplo 2 de MediaQuery

Columnas que se acomodan según el ancho de la página

Creo que no necesitamos más ejemplos para probar el punto de MediaQuery. Cuando la página se hace más chiquita, cambian cosas. Y ya. Entendimos el punto ¿Cierto?

Vamos rápido a la parte donde ya hacemos páginas responsivas de verdad. La complejidad de **acomodar cosas** según la resolución de nuestra pantalla. Para este ejemplo, no usaremos Boostrap, así que necesitamos conocer cómo usar Flexbox. Pero aún si no sabes usar Flexbox, este ejemplo es sencillo.

Primero vamos a hacer nuestra estructura base de una página con elementos acomodados en una disposición de 3 columnas.
Cuando la página alcance una resolución de 768pixeles de ancho, cambiará a una disposición de 2 columnas.

```
@media (max-width: 768px) {
    .item {
        flex: 1 1 50%;
    }
}
```

Cuando la página alcance una resolución de 480pixeles de ancho, cambiará a una disposición de una sola columna.

```
@media (max-width: 480px) {
    .item {
        flex: 1 1 100%;
    }
}
```

Vamos al código ✨

Esto va en tu HTML

```
<html lang="es">
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Página con Contenedor Dividido</title>
    <link rel="stylesheet" href="Index.css"> <!-- No olvides vincular tu CSS -->
   </head>
<body>
    <header>
        <!-- Aquí estaría nuestro header. -->
    </header>
    <main class="container">
        <div class="item">Item 1</div>
        <div class="item">Item 2</div>
        <div class="item">Item 3</div>
        <div class="item">Item 4</div>
        <div class="item">Item 5</div>
        <div class="item">Item 6</div>
        <div class="item">Item 7</div>
        <div class="item">Item 8</div>
        <div class="item">Item 9</div>
    </main>
</body>
</html>
```

Esto va en tu CSS

```css
/*
Estos estilos aplican para toda la página, lo único que estamos haciendo
es ponerle unos bordes a cada contenedor y especificando que no considere
ni margen ni sangría para cada contenedor. Solo para distinguir bien dónde inician
y terminan*/
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: Arial, sans-serif;
}

header {
    height: 60px;
    background-color: #f8f8f8;
    border-bottom: 1px solid #e7e7e7;
}
/* Aquí empezamos con el diseño responsivo ------------------------- */
.container {
    display: flex;
    flex-wrap: wrap;
    width: 100%;
    padding: 20px;
}

.item {
    flex: 1 1 33%;
    display: flex;
    justify-content: center;
    align-items: center;
    border: 1px solid #ccc;
    padding: 20px;
    box-sizing: border-box;
    text-align: center;
}
/* Aquí empleza la configuracion para 768pixeles ------------------------- */
@media (max-width: 768px) {
    .item {
        flex: 1 1 50%;
    }
}
/* Aquí empleza la configuracion para 480pixeles ------------------------- */
@media (max-width: 480px) {
    .item {
        flex: 1 1 100%;
```

No sé si pueda resultar confuso para ti el:

```
.container {
    display: flex;
    flex-wrap: wrap;
    width: 100%;
    padding: 20px;
}
```

No debería en caso de que ya estés viendo MediaQuery porque obviamente ya debiste haber visto “Flexbox”… Pero no confío en tus profesores. Así que asumiendo que no entiendes bien qué onda con esto, voy a explicar qué está ocurriendo ahí, no es difícil, mira.

`display: flex;`

Significa que usaremos “Flexbox” para acomodar las cosas de nuestra página. “Flexbox” se utiliza para acomodar cosas por filas y columnas. Hay otros valores de `display` como:

`block`**:** Hace que el elemento se comporte como un bloque. Ocupa todo el ancho disponible y comienza en una nueva línea

`inline`: Hace que el elemento se comporte como un elemento en línea. Solo ocupa el espacio necesario y no comienza en una nueva línea, si no que se pone juntito a lo que ya tengas escrito.

`grid`: Convierte el contenedor en un contenedor de cuadrícula, también podríamos usar ese. Pero es innecesariamente sofisticada, estamos bien con flex.

Y… Pues hay varios, ni vale la pena mencionarlos, por ahora. El punto es que nosotros estamos usando el sistema de Flexbox.

`flex-wrap: wrap;`

La propiedad `flex-wrap` controla si los elementos deben estar en una sola línea o pueden romperse en varias líneas. Los valores posibles son:

> `nowrap`: Este es el que trae por defecto si no le ponemos nada. Todos los elementos flexibles estarán en una sola línea, aunque esto pueda provocar el desbordamiento del contenedor.
>
> `wrap`: Los elementos flexibles se romperán en varias líneas si no caben en una sola línea. Esto es útil para asegurar que los elementos se mantengan dentro del contenedor y se distribuyan adecuadamente.
>
> `wrap-reverse`: Similar a wrap, pero las líneas se romperán en el orden inverso.

En nuestro caso, `flex-wrap: wrap`; asegura que si las columnas no caben en una sola línea (por ejemplo, en pantallas más pequeñas), se moverán a una nueva línea, manteniendo el diseño ordenado y legible. O en nuestro caso, cambiándolas de renglón cuando no quepan (Más adelante forzaremos que no quepan…)

```
flex: 1 1 33%;
flex: 1 1 50%;
flex: 1 1 100%;
```
Son tres líneas muy poco intuitivas y muy poco explicativas.

La propiedad `flex` es una abreviatura de tres propiedades individuales que juntas controlan la flexibilidad de los elementos dentro de un contenedor flex. La sintaxis general es:

flex: [Valor de Flex Grow] [Valor de Flex Shrink [Vakir de Flex-basis];

Osea que se le ponen 3 valores. En nuestro caso, un “1”, otro “1” y un porcentaje. ¿Qué significan?

flex-grow (el primer 1 en flex: 1 1 33%):

- Define la capacidad del elemento para crecer si es necesario.
- Un valor de 1 indica que el elemento puede crecer para llenar el espacio disponible.
- Si todos los elementos tienen flex-grow: 1, el espacio adicional se distribuirá equitativamente entre ellos.

flex-shrink (el segundo 1 en flex: 1 1 33%):

- Define la capacidad del elemento para encogerse si es necesario.
- Un valor de 1 indica que el elemento puede encogerse para evitar el desbordamiento del contenedor.
- Si todos los elementos tienen flex-shrink: 1, el espacio se reducirá equitativamente si es necesario.

flex-basis (el 33% en flex: 1 1 33%):

- Define el tamaño inicial del elemento antes de distribuir el espacio restante según flex-grow y flex-shrink.
- En este caso, 33% significa que cada elemento comenzará ocupando un 33% del ancho del contenedor. El 33% es un tercio del contenedor. El 50% es la mitad del contenedor, y el 100% es todo el contenedor.

¡Y ya está! ✨🎉

Ahora vamos a analizar cómo interactúan ambas cosas:

Primero le decimos a la página que nuestro contenedor de clase “.contenedor” va a usar Flexbox:

```
.container {
    display: flex;
```

```
Luego le decimos que cuando los elementos ya no quepan en cada renglón, se van a pasar para abajo:
```

```
    flex-wrap: wrap;
```

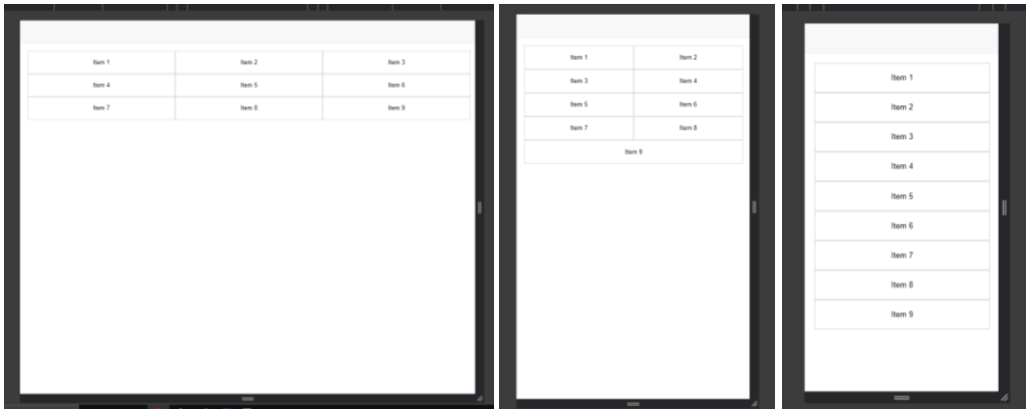
Y luego, establecemos que cada elemento de la clase “.item” va a ocupar un tercio del contenedor al inicio.

```
.item {
    flex: 1 1 33%;
    display: flex;
```

Pero, que al alcanzar la pantalla los 768 pixeles, entonces ya no va a ocupar un tercio, si no la mitad.

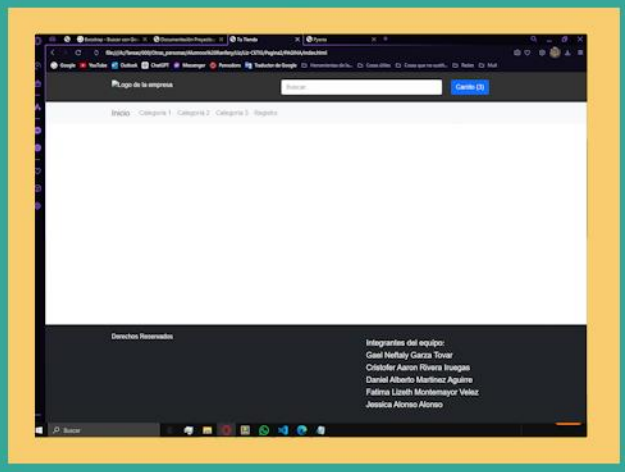
```
@media (max-width: 768px) {
    .item {
        flex: 1 1 50%;
    }
}
```

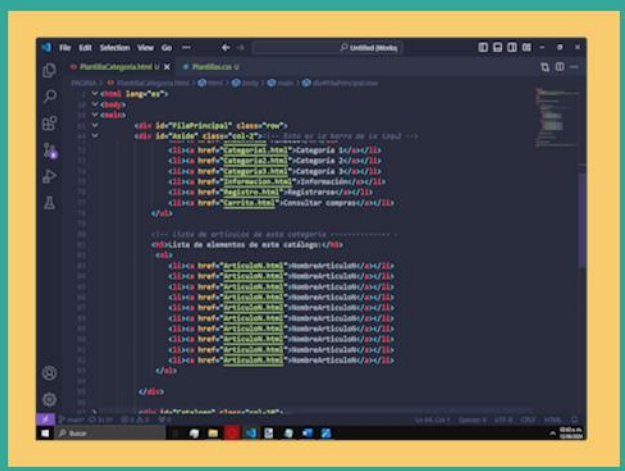
Pero sabemos que cada renglón tiene tres elementos, y en una fila **no caben tres mitades**. Sólo dos (por eso son mitades…) Y como no caben, se pasan para abajo como establecimos en `flex-wrap: wrap;`



1200pixeles 768pixeles 480pixeles

Documentación del proyecto





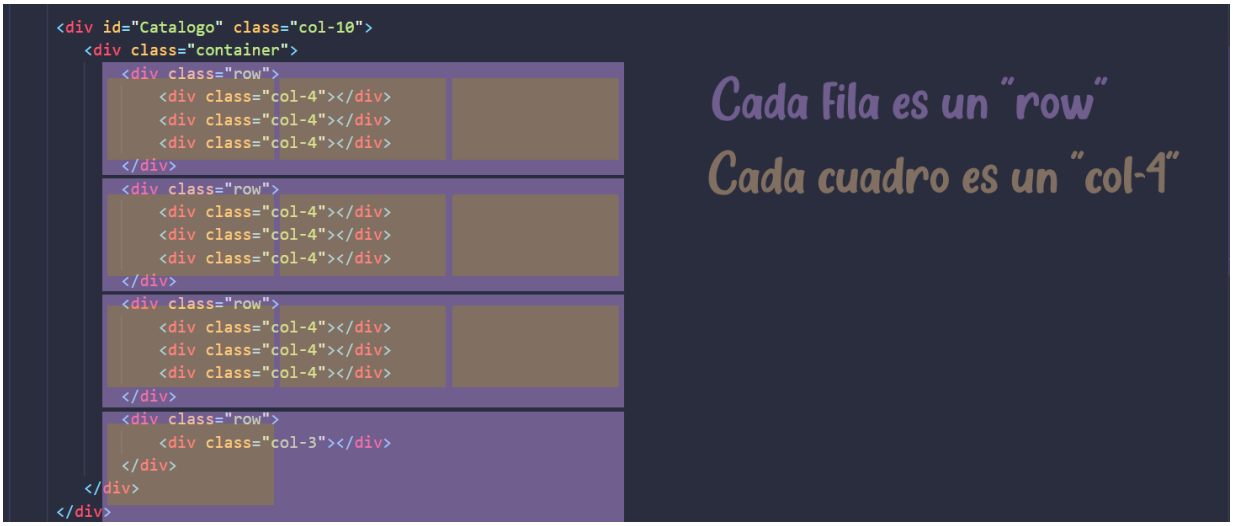
Catálogo (PlantillaCategoría.html)

Para la parte del catálogo hice 4 filas de 3 elementos cada una. Pero la última fila sólo tiene un elemento. Así hago 3 filas de 3 elementos (9 elementos) y una última fila con sólo un elemento (Para completar las 10).

Cada fila es un `<div class="row">` y dentro de cada fila metí 3 elementos `<div class="col-4">`. Cada elemento de clase col-4 es una división de la fila, específicamente una que mide una tercera parte de la fila. Por eso sólo hay 3 por cada fila.

Por si se les pasa la teoría del grid… "4" es la tercera parte de 12. En cada fila "row" caben 12 espacios imaginarios. O sea, caben 3 elementos de 4 espacios cada uno.

Les puse colores para que se entienda mejor.



Aquí los espacios amarillos están vacíos. Pero ya en la página, pues cada uno tiene un elemento de un artículo. En mi caso usé "Cards" de Bootstrap para cada artículo.