Diseño web adaptativo

En este documento abordaremos el tema del diseño web adaptativo o *responsive*, aunque en breve veremos que estos dos conceptos no son exactamente iguales. Básicamente, la temática a abordar es cómo diseñar una web para que se vea adecuadamente en diferentes tipos de dispositivos o resoluciones, tales como *smartphones*, tablets o monitores de alta resolución.

1. Introducción al diseño adaptativo

Como acabamos de comentar, abordaremos la problemática de cómo diseñar una web para distintos tipos de dispositivos. Esta afirmación en sí plantea dos preguntas:

- ¿Qué tipos principales de dispositivos podemos o debemos considerar?
- ¿Qué estrategias hay para abordar el diseño web para esos tipos de dispositivos?

1.1. Tipos de resoluciones

Tratemos de responder a la primera pregunta: *¿qué tipos principales de dispositivos podemos o debemos considerar?*. Realmente no es ésa la pregunta adecuada, sino más bien a cuántos tipos de resoluciones diferentes podemos o debemos adaptar nuestra web. Y para responder a esta pregunta, podemos hacer un análisis rápido de los distintos tipos de pantalla en los que habitualmente podemos consultar una página web:

- Por un lado, tenemos *smartphones* de baja resolución o resolución *extra-pequeña* (normalmente abreviada *xs*, *extra small*). La resolución máxima típica en esta categoría no llega a unos **576px**.
- El siguiente escalón estaría formado por dispositivos pequeños (normalmente *smartphones* también) de mayor resolución. Conformarían una resolución *pequeña* (a menudo abreviada como *sm* de *small*), con valores máximos de hasta unos **768px**.
- A continuación están las pantallas de pequeño tamaño, como las de las tablets, que formarían una resolución media (normalmente abreviada como *md*), con valores máximos de hasta unos **992px**.
- Después vendrían pantallas más grandes (abreviadas *lg*), como los monitores de gama media, con resoluciones de hasta unos **1200px**.
- Le siguen las pantallas aún más grandes (abreviadas *xl*), con resoluciones de hasta unos **1400px**.
- Finalmente, podríamos hablar de pantallas todavía más grandes (abreviadas *xxl*), para resoluciones mayores que la anterior.

1.2. Estrategias de diseño. *Responsive* vs *adaptativo*

Vayamos ahora con la segunda pregunta formulada anteriormente: a la hora de abordar el diseño de una web para dar respuesta a todos los tipos de resoluciones posibles (o un grupo de ellos), tenemos dos estrategias:

- La estrategia **responsive** consiste en definir un único diseño que automáticamente se va adaptando a las distintas resoluciones que indiquemos, y re-posicionando sus elementos conforme a esas resoluciones. Para abordar esta estrategia podemos hacer uso de elementos como *flexbox*, que veremos a continuación, y también de frameworks específicos de diseño web, como Bootstrap.
- La estrategia **adaptativa** consiste en definir un diseño para cada una de las posibles resoluciones que queremos tratar, de forma que se activa o carga uno u otro en función de la resolución. Esta estrategia se aborda mediante el uso de *media queries*, que también trataremos a continuación.

1.3. Gestionando el *viewport*

Una de las etiquetas *meta* que comentamos en las primeras sesiones de este curso es la etiqueta *viewport*. Esta etiqueta permite a los diseñadores web controlar el área visible de una web. En general, la etiqueta que se recomienda utilizar tiene este aspecto:

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

El parámetro `width=device-width` hace que la anchura de la página se iguale a la anchura de la pantalla del dispositivo. Por otra parte, el parámetro `initial-scale=1.0` establece el nivel de zoom inicial en 1, es decir, en el zoom del 100%, cuando la página se carga por primera vez.

2. Diseño adaptativo. Uso de *media queries*

Las *media queries* (en español, *consulta sobre medios*) son una técnica proporcionada por CSS3 que permite adaptar la visualización del contenido de una página a las características del dispositivo en que se va a mostrar. Estas características incluyen, sobre todo, la resolución de la pantalla, pero también podemos hablar de otras, como el tipo de medio. Así, por ejemplo, podemos definir un estilo diferente para una web que se vaya a imprimir en papel y eliminar el color de fondo, entre otras cosas.

2.1. Estructura general de una *media query*

Las *media queries* se definen en el propio documento CSS, a través de la expresión `@media` seguida de las condiciones que debe cumplir el dispositivo para aplicar los estilos incluidos en ella. En concreto, su sintaxis general es:

```
@media not|only tipo_medio and (expresiones)
{
    estilos CSS
}
```

donde:

- `tipo_medio` alude al tipo de medio donde se va a mostrar el contenido. Típicamente es *screen* (pantalla), pero puede valer también *print* (impreso en papel), *speech* (hablado) u *all*.
- Las *expresiones* consisten en una serie de condiciones que debe cumplir el dispositivo. Típicamente se alude a su anchura mínima o máxima (`min-width` / `max-width`).

Por ejemplo, esta *media query* se aplicaría únicamente a pantallas con una resolución mínima de 768px:

```
@media only screen and(min-width: 768px)
{
    ...
}
```

Para el uso que vamos a hacer aquí de las *media queries* nos bastará con especificar una anchura mínima o máxima, por lo que podemos dejarlas con este formato:

```
@media (min-width: 768px)
{
    ...
}
```

2.2. Estrategia de uso de las *media queries*

A la hora de aplicar *media queries* a nuestra web para definir diferentes disposiciones y diseños de elementos acordes a distintas resoluciones de pantalla, una estrategia general es:

1. Fuera de las *media queries* (por ejemplo, al inicio del documento CSS) definimos los estilos generales que no van a cambiar entre resoluciones (colores de fondo, tipos de letra, etc) y también la disposición de los elementos para la resolución más baja (o más alta) que queramos tratar.
2. Definimos una *media query* por cada resolución superior (o inferior) que queramos contemplar, y dentro, redefinimos únicamente los estilos que queramos adaptar a esa resolución. Por ejemplo, la anchura o tamaño de ciertas cajas, etc.

Por ejemplo, supongamos el siguiente contenido HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo media queries</title>
    <link rel="stylesheet" href="estilos.css">
</head>
<body>
    <section id="contenedor">
        <section id="caja1">Caja 1</section>
        <section id="caja2">Caja 2</section>
    </section>
</body>
</html>
```

Vamos a definir dos tipos de resoluciones: para pantallas de hasta 768px de ancho, mostraremos una caja debajo de la otra. En cambio, para pantallas de mayor anchura, las mostraremos en paralelo, usando para ello un *grid* de dos columnas. Nuestro CSS quedaría así:

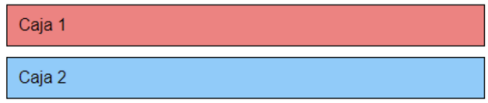
1. Primero definimos los estilos generales y la disposición de las cajas para baja resolución:

```
body
{
    font-family: Arial;
    margin: 10px 5%;
}

#caja1
{
    background-color:lightcoral;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid black;
}

#caja2
{
    background-color:lightskyblue;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid black;
}
```

2. Después, definimos la *media query* para resoluciones mayores, y definimos la disposición en rejilla de las dos cajas en este caso:

```css
@media(min-width: 768px)
{
    body
    {
        margin: 100px 20%;
    }

    #contenedor
    {
        display: grid;
        grid-template-columns: 1fr 1fr;
        grid-template-rows: auto;
        gap: 10px;
    }
}
```

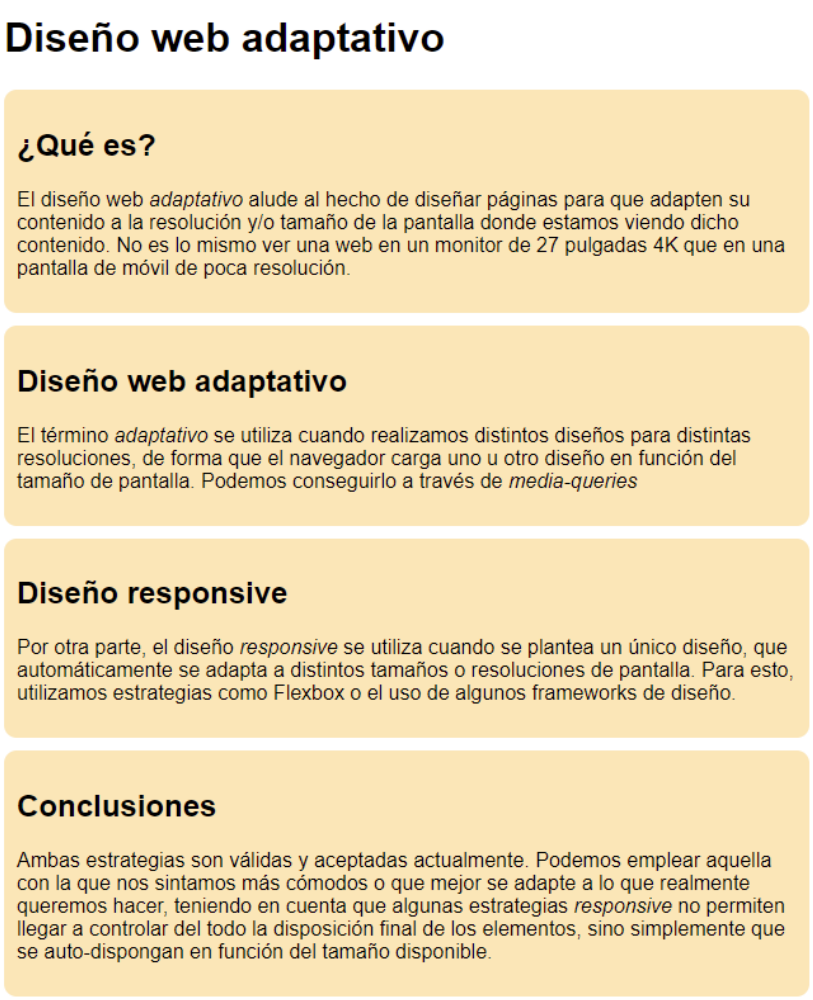
Y obtendríamos un resultado como este (baja resolución / alta resolución):



Ejercicio 1:

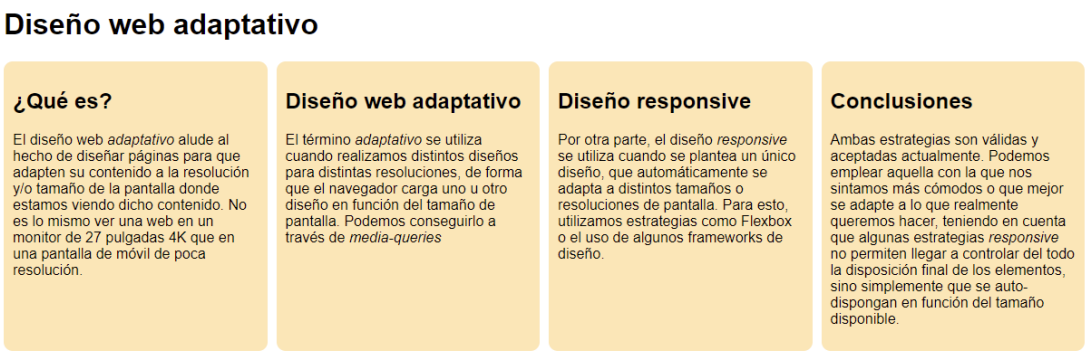
Descarga este ejemplo y define un documento CSS donde, ayudándote de *media queries*, definamos tres posibles disposiciones:

- Para resoluciones bajas (hasta 768px), mostraremos las cajas una debajo de otra



- Para resoluciones intermedias (hasta 992px), mostraremos la caja 1 arriba, la caja 4 abajo, y las dos intermedias en paralelo. Además, cambiaremos el color de las cajas 1 y 4 a gris (ver imagen resultado)

- Para resoluciones mayores, mostraremos las 4 cajas en paralelo con sus colores originales



3. Diseño *responsive*. Uso de *flexbox*

Flexbox (*flexible boxes*) es un mecanismo de diseño web *responsive* facilitado por CSS3, que permite especificar una serie de propiedades en las cajas para que se adapten automáticamente al tamaño de pantalla existente, dependiendo de si van a tener suficiente espacio para mostrarse de un modo u otro.

Del mismo modo que ocurre con el *grid* de CSS, es necesario que las cajas involucradas en el proceso *responsive* estén contenidas en otra, de forma que podemos especificar las propiedades generales de *flexbox* en la caja contenedora, y luego definir, adicionalmente, cómo queremos disponer las cajas en ese contenedor.

3.1. Configurando la caja contenedora

Entre las propiedades que podemos definir en la caja contenedora, podemos destacar las siguientes:

- `display`: en este caso, la propiedad *display* deberá tener el valor de ***flex*** para habilitar el contenedor como un contenedor de elementos flexibles. Alternativamente, también se le puede dar el valor *inline-flex* si queremos que los elementos de dentro se comporten como elementos en línea (no en bloque).
- `flex-direction`: indica cómo se van a disponer las cajas dentro del contenedor. El valor por defecto es *row*, que hace que se dispongan de izquierda a derecha, pero también podemos utilizar *row-reverse* (horizontal pero invertido), *column* (de arriba a abajo) y *column-reverse* (vertical pero invertido). Podemos intuir, por tanto, que con Flexbox vamos a poder disponer los elementos en una dirección (bien horizontal, bien vertical) de forma flexible.
- `justify-content`: indica cómo se va a rellenar el espacio entre cajas, si lo hay. Puede tomar los valores *flex-start* (se agrupan las cajas al inicio del espacio disponible), *flex-end* (al final), *center* (centradas en medio del espacio disponible), *space-between* (justificadas con espacio entre ellas) o *space-around* (justificadas con espacio también a los lados).
- `flex-wrap`: indica cómo vamos a adaptar las cajas en caso de que no quepan todas en una misma fila (o columna). Su valor por defecto es *nowrap*, lo que indica que no hay ningún ajuste: las cajas se disponen según lo indicado en las propiedades anteriores, y si no caben se siguen disponiendo en esa dirección. Pero si lo establecemos a *wrap*, entonces se redistribuirán si no hay espacio suficiente. Es el valor más habitual si queremos establecer un diseño *responsive*.
- ...

Existen algunas webs como esta donde podemos probar el comportamiento de estas y otras propiedades, y ver cómo funciona cada una en realidad.

3.2. Configurando las cajas internas

Para cada caja contenida en el elemento contenedor, podemos configurar sus propiedades *flexbox* para indicar dónde se va a ubicar y cuánto espacio va a ocupar. Algunas de estas propiedades son:

- `order`: indica el orden de la caja dentro de la secuencia de cajas contenidas. Si no indicamos nada, cada caja se coloca a continuación de la anterior, siguiendo la disposición indicada en la caja contenedora. Pero podemos alterar ese orden natural reubicando las cajas. Se trata simplemente de un número; cuanto mayor sea, la caja se colocará más hacia el final de la secuencia.
- `flex`: indica el tamaño de la caja respecto al resto. Son tamaños relativos, igual que ocurre con las unidades *fr* en la rejilla *grid* CSS. Si todas las cajas tienen un tamaño *flex* de 1, todas ocuparán lo mismo. Si a alguna le asignamos un tamaño de 2, ocupará el doble que el resto. Alternativamente, también podemos utilizar esta propiedad para dar un tamaño fijo en píxeles (por ejemplo, *flex:200px*) o en porcentaje (*flex: 40%*). Esta última opción es útil si queremos controlar mejor cuántas columnas queremos que haya en cada fila.
- `min-width`: indica la anchura mínima que debe tener la caja para mostrar su contenido. Si esta anchura no se cumple y tenemos habilitada la opción `flex-wrap: wrap` en el contenedor, entonces hará que vaya a la siguiente fila/columna disponible para mostrarse.

Echemos un vistazo a este ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo flexbox</title>
    <link rel="stylesheet" href="estilos.css">
</head>
<body>
    <section id="contenedor">
        <section id="caja1">Caja 1</section>
        <section id="caja2">Caja 2</section>
        <section id="caja3">Caja 3</section>
    </section>
</body>
</html>
```

Imaginemos que queremos poner las tres cajas en horizontal, con la caja 2 ocupando el doble de espacio, y que todas tengan una anchura mínima de 250px. Si eso no puede cumplirse, queremos que las cajas se redistribuyan para cumplir con esa anchura mínima (normalmente una debajo de otra).

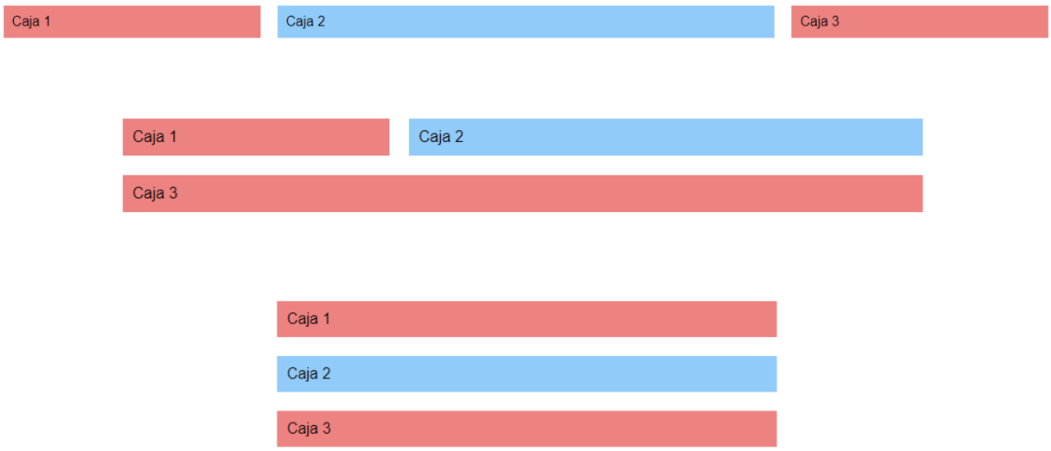
En primer lugar, definimos las características del elemento contenedor:

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

Después, establecemos las propiedades de las cajas contenidas. Además de sus características particulares (color de fondo, *padding*, etc), indicamos sus propiedades *flexbox*: anchura (propiedad *flex*) y anchura mínima.

```
#caja1, #caja3
{
    background-color:lightcoral;
    padding: 10px;
    margin: 10px;
    flex: 1;
    min-width: 250px;
}

#caja2
{
    background-color:lightskyblue;
    padding: 10px;
    margin: 10px;
    flex: 2;
    min-width: 250px;
}
```

Si vamos variando el tamaño de la ventana del navegador, podemos comprobar cómo las propias cajas se van ajustando a distintos tamaños y posiciones para cumplir con lo establecido. Aquí vemos la evolución desde pantallas grandes a pequeñas:



Notar que, si eliminamos la propiedad `flex-wrap: wrap` del elemento contenedor, las cajas ya no se redistribuyen. Se quedan ocupando su espacio original, y si encogemos demasiado la pantalla nos tocará hacer scroll para verlas.

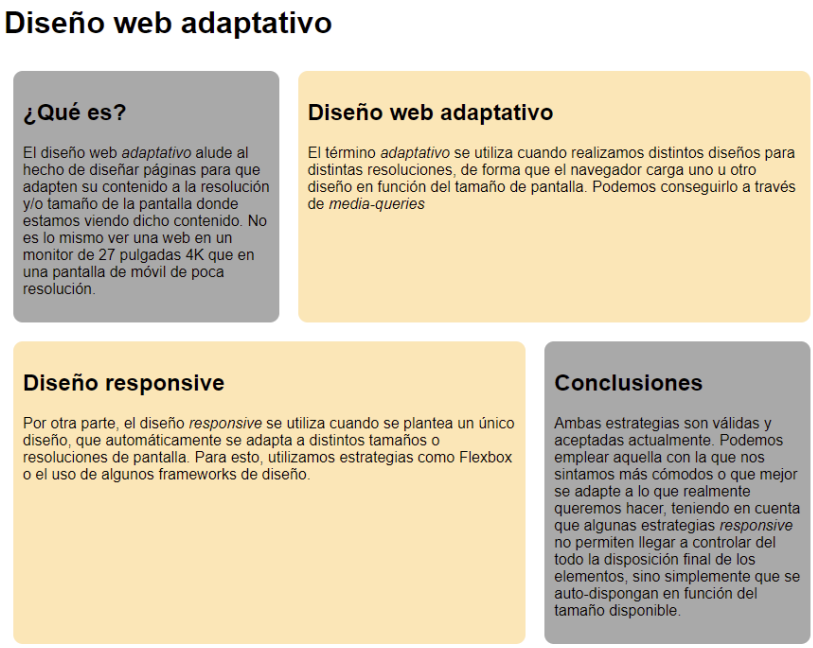
Notar también que, utilizando *flexbox*, podemos reajustar las cajas automáticamente en cuanto el tamaño de la pantalla "choque" con los requisitos de cada caja, pero no podemos controlar del todo dónde va a parar cada caja. En el ejemplo anterior, llega un punto en que la caja 3 baja a la segunda fila, quedando la caja 1 y la 2 en primera fila... pero quizá preferiríamos otra distribución para esa situación intermedia.

Puedes leer más información sobre Flexbox en otras webs interesantes, como por ejemplo esta.

> **Ejercicio 2:**

> Descarga este ejemplo que ya hemos utilizado en el ejercicio previo. Ahora vamos a configurarlo mediante *flexbox* para que:
>
> - Las dos cajas centrales ocupen el doble de tamaño que las laterales. Las laterales van a tener un color de fondo gris y las centrales amarillo-naranja.
> - Las cajas laterales tengan una anchura mínima de 200px, y las centrales de 300px. Cuando esto no sea posible, se redistribuirán

Aquí tienes un ejemplo de cómo puede quedar, a algunas de las resoluciones posibles:



4. Conclusiones. *Flexbox* vs *media queries*

En general, *flexbox* es una buena opción para una distribución flexible *unidimensional* (es decir, colocar componentes en una misma fila o columna y que se auto-distribuyan dependiendo del tamaño disponible). Pero para disposiciones bidimensionales (tablas), *flexbox* se queda corto por sí mismo, si queremos controlar la distribución concreta de los elementos. Una alternativa a esto podría ser utilizar *grid* y *media-queries* para diferentes posibles tamaños.

Con lo visto hasta ahora, podemos optar por un diseño *adaptativo* (*media queries* + *grid*) si queremos controlar mejor la disposición bidimensional de los elementos de la web, o por un diseño *responsive* (*flexbox*) si sólo queremos controlar la disposición unidimensional (fila a fila, o columna a columna). El inconveniente de lo primero es que nos "obliga" a plantear distintos diseños complementarios para elegir automáticamente cuál de ellos cargar. Una alternativa a esto es utilizar algún framework de diseño web, como Bootstrap, que simplifica bastante esta tarea.