UD3 HTML Y CSS.

DISEÑO DE INTERFACES WEB

3.6 Responsive Design

Contenidos

1. Objetivos
2. Introducción
3. Páginas adaptables (Responsive Web Design)
4. Ejercicios

1. Objetivos

En este sexto tema se pretenden conseguir los siguientes objetivos:

- Entender las ventajas del Responsive Web Design.
- Crear páginas web adaptables que se visualicen correctamente en distintos dispositivos.
- Modificar páginas web no adaptables para convertirlas en adaptables.

TÉCNICAS para un diseño responsive:
- meta viewport
- media queries
- uso de unidades relativas (%, em, rem, vw, vh)
- frameworks (Bootstrap, Tailwind css)

2. Introducción

En este tema, veremos cómo conseguir que nuestras páginas web se **adapten a distintas resoluciones** o, lo que es lo mismo, a distintos dispositivos. Esto se conoce como Responsive Web Design.

Estudiaremos el origen del Responsive Web Design, las **ventajas** que nos aporta y de qué **herramientas** disponemos para llevarlo a cabo. En concreto, hablaremos del meta `viewport` y de `las @media querys` de CSS3.

Finalmente, veremos cómo **poner en marcha estas técnicas** con algunos ejemplos en los que nuestra página pasará de estar en columnas a estar en cascada o cambiaremos el aspecto de nuestra barra de navegación en función del espacio disponible que tengamos.

3. Páginas adaptables (Responsive Web Design)

El diseño web adaptable o adaptativo (en inglés, Responsive Web Design) es una **filosofía de diseño y desarrollo web** que, mediante el uso de estructuras e imágenes fluidas, así como de @media queries en la hoja de estilo CSS, consigue **adaptar el sitio web al entorno del usuario**.

El diseñador y autor norteamericano Ethan Marcotte creó y difundió esta técnica a partir de una serie de artículos en "A List Apart" (http://alistapart.com/article/responsive-web-design/), una publicación en línea especializada en diseño y desarrollo web, idea que luego extendería en su libro Responsive Web Design.

Viewport

Esta meta-etiqueta fue creada en principio por **Apple** para su iPhone, pero se ha convertido en todo un estándar que es soportado por la **mayoría de los dispositivos móviles**.

Su uso es totalmente necesario, ya que sino **el navegador establece el ancho** con el que prefiere visualizar una página, en lugar de **usar el ancho del que dispone**, es decir, si la pantalla de nuestro móvil tiene 400px y el navegador detecta que lo óptimo sería visualizarla con 700px, así lo hará, a no ser que usemos este meta.

El meta lo añadiremos en el `<head>` de nuestra página:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
```

Se pueden usar los siguientes parámetros (separados por comas):

Viewport

- `width`: **ancho** de la página (se puede establecer en píxeles o como `device-width` y usará el ancho del que dispone el dispositivo).
- `height`: **alto** de la página, actúa igual que el `width`.
- `initial-scale`: **escala o zoom inicial** de la página (este y los demás tipos de escala se establecen con valores como 1.0 para no tener zoom o 2.5 para tener un zoom de 2 aumentos y medio, por ejemplo).
- `minimum-scale`: **zoom mínimo** que podemos hacer.
- `maximum-scale`: **zoom máximo** que podemos hacer .
- `user-scalable`: establece si está **permitido o no** hacer zoom (yes/no).

Preparar nuestra página para el diseño adaptable

Para lograr que nuestra web se adapte a los anchos de pantalla, debemos tener en cuenta varias cosas importantes:

- Dejar de usar píxeles en todos los sitios y, en su lugar, **usaremos porcentajes** (por ejemplo: width: 60%). Para saber qué porcentajes debemos de poner, haremos la siguiente operación: Supongamos que tenemos un elemento con un ancho de 960 píxeles y, dentro de este, tenemos otro elemento de 300 píxeles. Al elemento externo le pondremos, por ejemplo, un width del 90% para que se adapte al tamaño de la ventana y para que el elemento interno mantenga la proporción con respecto al externo calcularemos el ancho que debería tener así: **300/960, lo cual, nos dará 0,3125**. Por lo tanto, el width que le pondremos al elemento interno ser **31,25%**.

Preparar nuestra página para el diseño adaptable

- Para **limitar el ancho (o alto si se terciara)** debemos de usar los parámetros **max-width** y **max-height** en el caso del alto y **min-width** y **min-height** para el mínimo. El valor de estas propiedades sobrescribirá el valor de width y height.
- No debemos usar **posiciones absolutas ni fijas** (salvo contadas excepciones).
- Nunca debemos permitir que una imagen de fondo que está pensada para no repetirse, llegue a **repetirse por el cambio de dimensiones**. Para evitarlo, debemos adaptarla, normalmente, usando **@media-queries**.
- No debemos permitir que imágenes y vídeos **se salgan de la estructura**, sino aparecerá un scroll lateral en los dispositivos móviles que destrozará totalmente el diseño.

@media queries

Una media query nos permite apuntar, no sólo a ciertas **clases de dispositivos**, sino realmente **inspeccionar las características físicas** del dispositivo que está renderizando nuestro trabajo.

Las media queries se han convertido en una popular técnica del lado del cliente para entregar una hoja de estilos a medida para el iPhone, los smartphones Android o las tablets. Para hacerlo, podemos incorporar una query al **atributo media** de una hoja de estilos linkeada:

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px)" href="shetland.css" />
```

@media queries

La query anterior contiene un media type screen (`all | printer | screen | speech`) y una la consulta entre paréntesis. Le estamos preguntando al dispositivo, **si su resolución horizontal (`max-device-width`) es igual o menor que 480px**.

Si visualizamos la página en un dispositivo con una pantalla pequeña como el iPhone, entonces el dispositivo cargará `shetland.css`. **De lo contrario, el link se ignora**.

Además, podemos testear múltiples valores en una sola query, encadenándolos con la palabra clave and:

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px) and (resolution: 163dpi)" href="shetland.css" />
```

@media queries

Además del uso de la etiqueta <link> para **incluir media queries**, también tenemos la posibilidad de incluirlas en nuestro CSS como parte de una **regla @media**:

```
@media all and (max-device-width: 480px) {
    .column {
        float: none;
    }
}
```

O como parte de una **directiva @import**:

```
@import url("shetland.css") all and (max-device-width: 480px);
```

En cada caso, el efecto es el mismo: si el dispositivo pasa el examen planteado por nuestra media query, el CSS que corresponda es aplicado a nuestro código. La palabra all indica que la regla será para **todos los tipos de medios** y no para uno específico.

@media queries

Aunque las @media queries nos permiten conocer muchas propiedades diferentes, las más importantes son:

- aspect-ratio: Dimensiones relativas del dispositivo expresadas como **relación de aspecto**: 16:9 por ejemplo.
- Width y height: **Dimensiones** del área de visualización. Pueden ser expresadas en valores **mínimos y máximos**.
- orientation: Detecta si es panorámico (ancho > alto) o vertical (alto > ancho) para ajustar los diseños en dispositivos que permiten **giro de la pantalla**. (orientation: portrait/landscape).
- resolution: **Densidad de pixeles** en el dispositivo. Esto es muy útil cuando queremos aprovecharnos de las ventajas de los dispositivos que tiene una resolución mayor a 72 dpi.
- color, color-index y monochrome: Detectan el **número de colores o bits por color**. Esto nos permite crear diseños específicos para dispositivos monocromáticos.

Puesta en marcha

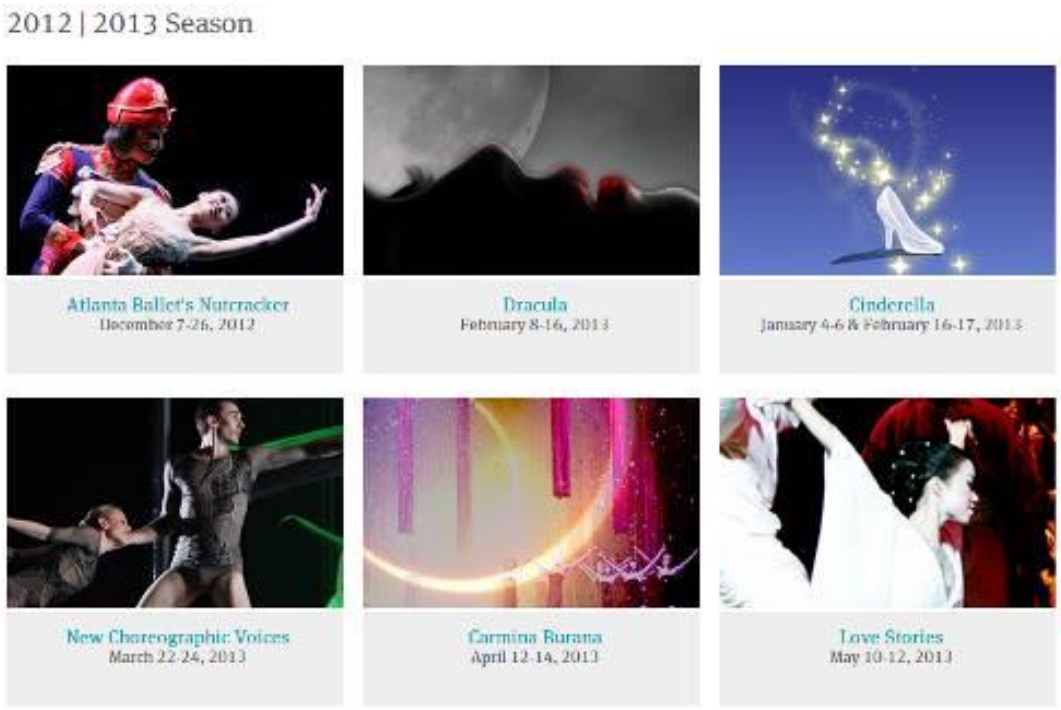
Para empezar veremos cómo hacer fluir elementos, de forma que se reubiquen en función del tamaño disponible en la pantalla. En primer lugar, nos olvidaremos de los float y usaremos en su lugar la **propiedad display**.

La propiedad display de CSS nos permite establecer cómo se **comportará un elemento respecto a los demás**. Nos interesan, principalmente, tres valores posibles:

- block: los elementos se ubican **uno debajo del otro**.
- inline: los elementos se ubican **uno junto al otro**, pero sin respetar las propiedades de alto y ancho.
- inline-block: **mezcla de las dos anteriores**, comportándose como un bloque, pero poniéndose en línea con otros elementos iguales.

Puesta en marcha

Una de las cosas que se dan mucho en los diseños adaptativos es crear un conjunto de bloques con imágenes y texto que, **según el tamaño de la ventana, se van recolocando**. Por ejemplo, en la web de la imagen se ven seis bloques, tres encima y tres debajo, pero si los visualizáramos con un ancho de pantalla mayor se verían en línea. Veamos cómo podríamos crear una estructura como esa:



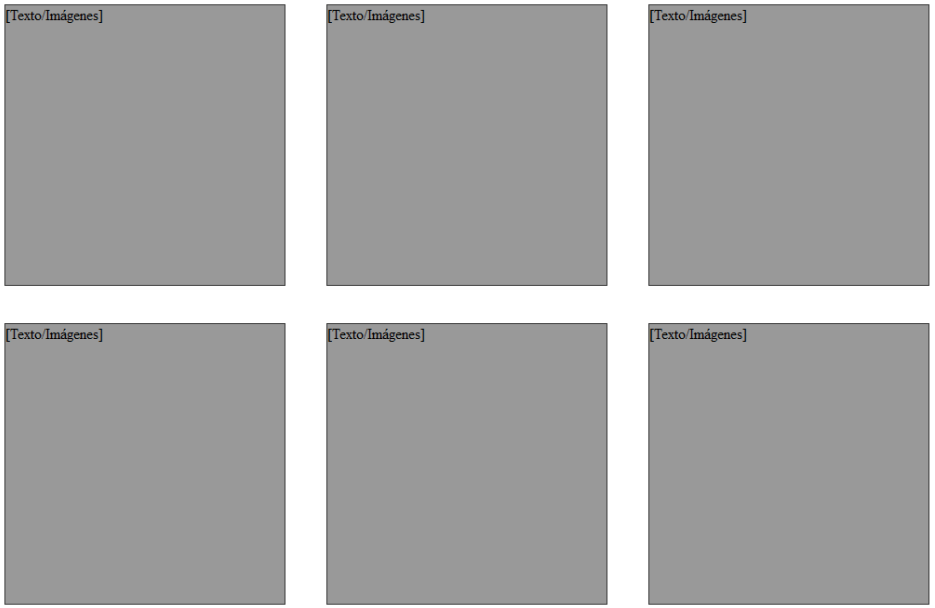
Puesta en marcha BLOQUES 1

```
<div id="contenedor">
   <figure class="bloque">[Texto/Imágenes]</figure>
   <figure class="bloque">[Texto/Imágenes]</figure>
   <figure class="bloque">[Texto/Imágenes]</figure>
   <figure class="bloque">[Texto/Imágenes]</figure>
   <figure class="bloque">[Texto/Imágenes]</figure>
   <figure class="bloque">[Texto/Imágenes]</figure>
</div>
```

En la hoja de estilos, simplemente, estableceremos el modo de visualización `display:inline-block` para el elemento figure, y cuando los elementos **no quepan** en el contenedor se reajustarán sin perder su tamaño.

Puesta en marcha

```
#contenedor .bloque {
    display: inline-block;
    height:300px;
    width: 300px;
    border:1px solid #333;
    background: #999;
    margin:20px;
}
```



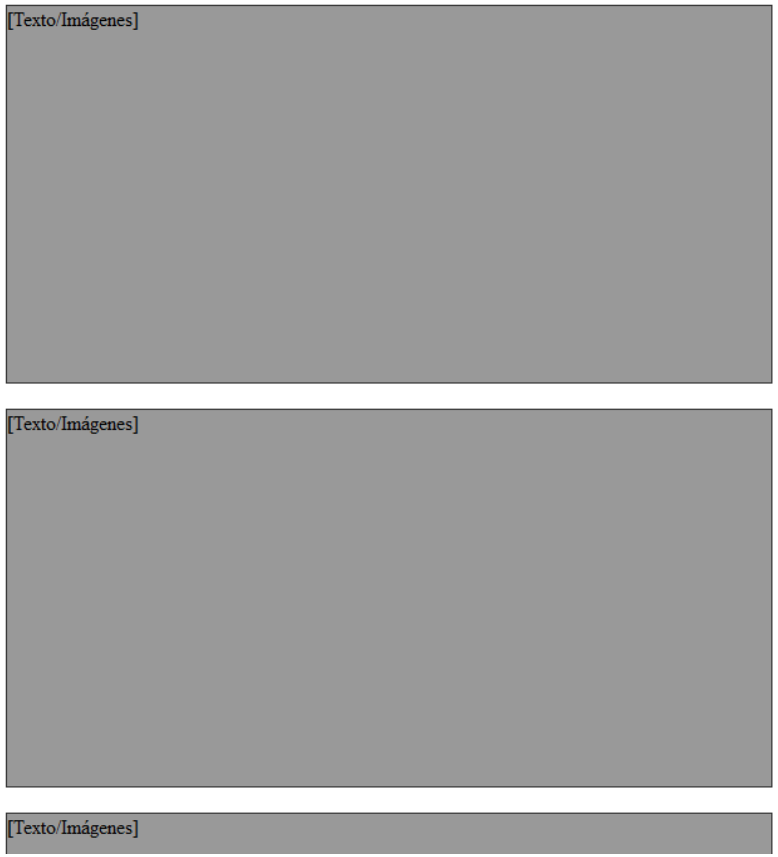
Podéis ver este ejemplo en la **carpeta bloques1** de Aules. Si probáis a ampliar y reducir el tamaño de la ventana de vuestro navegador, los elementos se van reubicando automáticamente gracias a la propiedad `display`. Pero ¿qué ocurre si queremos que los elementos se comporten de una forma diferente cuando el tamaño de ventana disponible sea inferior a un valor determinado?

Puesta en marcha

BLOQUES 2

En ese caso, utilizaremos las **media queries**. Por ejemplo, si quisiéramos que **con menos de 800 pixeles** los bloques anteriores se mostraran unos debajo de otros, centrados y ajustando su ancho al espacio disponible, haremos lo siguiente:

```
@media all and (max-width: 800px)
{
   #contenedor .bloque
   {
      /* Cuando el ancho sea inferior a
      800px el elemento será un bloque */
      display: block !important;
      width: auto !important;
   }
}
#contenedor .bloque
{
      display: inline-block;
      height:300px;
      width: 300px;
      border:1px solid #333;
      background: #999;
      margin:20px;
}
```



Puesta en marcha

Indicamos que los elementos **se comporten como bloques**, es decir, que se ubiquen uno debajo de otro (display: block) y, para que se adapten al espacio disponible, indicamos el ancho automático (width: auto). Al poner ancho automático se ubicará el elemento dejando 20 píxeles de margen a cada lado (margin:20px) y ocupará el resto del espacio.

Es importante observar que cuando se cumple la condición indicada en la media query, en primer lugar, se aplican los estilos que hemos indicado fuera de ella y, posteriormente, se aplicarán los estilos de dentro.

Podéis ver este ejemplo en la carpeta bloques2 del fichero de recursos.

Puesta en marcha !important

El elemento recibirá los estilos que se indican en ambos sitios:

```
#contenedor .bloque
{
	display: inline-block;
	display: block !important;
	height:300px;
	width: 300px;
	width: auto !important;
	border:1px solid #333;
	background: #999;
	margin:20px;
}
```

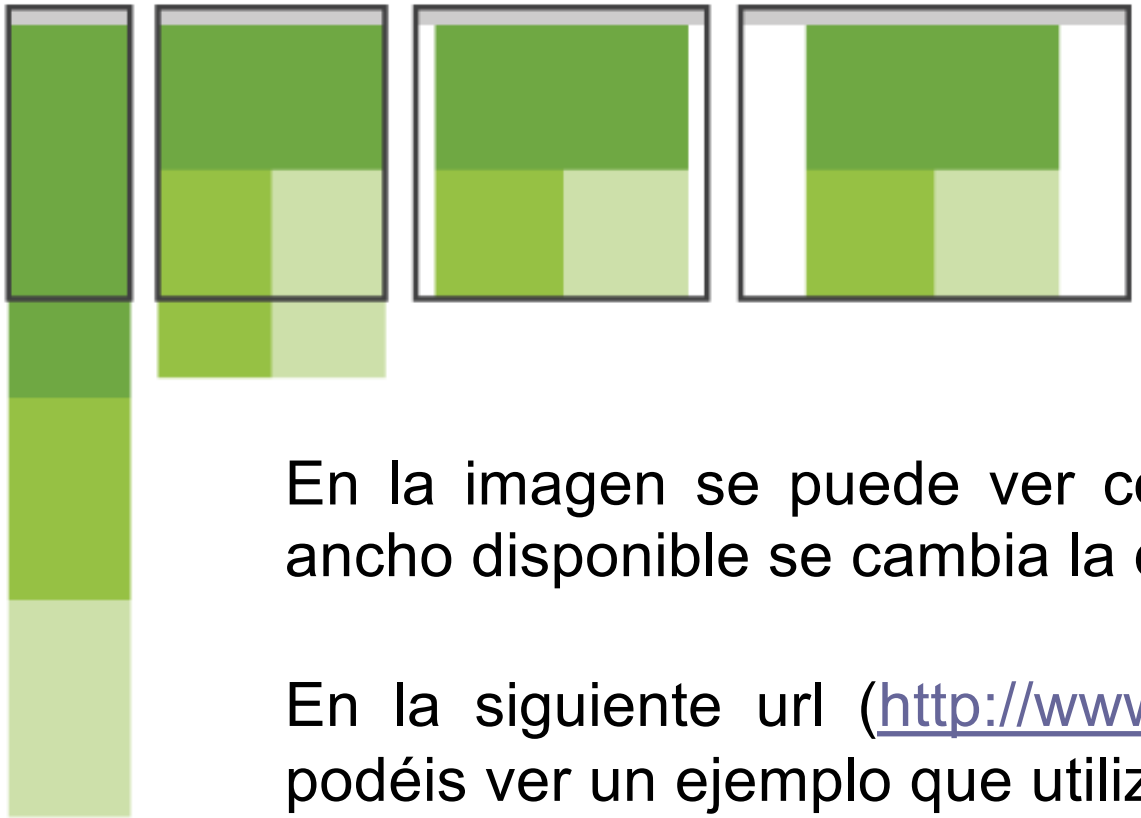
Como vemos, la propiedad **display y width se aplica dos veces**. ¿Cómo sabe el navegador cuál aplicar? Lo que hacemos es indicar la palabra **!important** en aquellas propiedades que queramos que tengan preferencia sobre el resto.

Situando la media query al final de nuestro código, evitamos tener que usar la palabra !important.

Pasar de columnas a cascada

CASCADA

Uno de los métodos que se utiliza más a menudo para cambiar la estructura de una web adaptativa, que suele tener una cabecera y dos o tres columnas (con el menú, el contenido, enlaces, etc.) es el **paso de la disposición en columnas a la disposición en cascada**.



En la imagen se puede ver cómo, en función del ancho disponible se cambia la disposición.

En la siguiente url (http://www.hsgac.senate.gov/) podéis ver un ejemplo que utiliza esta técnica.

Pasar de columnas a cascada

A continuación, veremos cómo hacer una estructura adaptable* similar a la de la imagen anterior. Para el ejemplo, sólo tendremos dos estados: la estructura bien maquetada (ancho suficiente) y la estructura en cascada (ancho insuficiente).

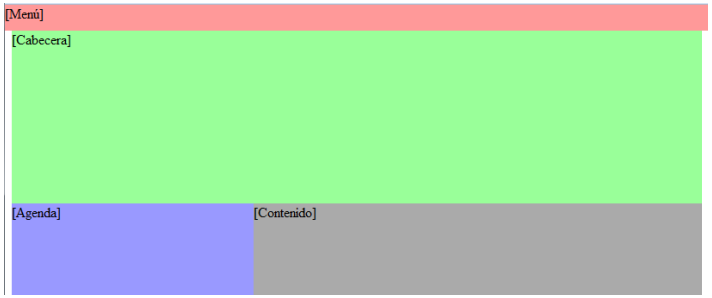
```
<nav>[Menú]</nav>
<div class="pagina">
   <header>[Cabecera]</header>
   <aside>[Agenda]</aside>
   <main>[Contenido]</main>
</div>

@media all and (max-width: 600px)
{
   div, nav, header, aside, main
   {
      /* Cuando el ancho sea inferior a 600px el elemento será un bloque */
        display: block !important;
      /* Se ajustarán al ancho de la ventana */
        width: 100% !important;
        margin: auto !important;
      /* La posición será estática (flujo normal del documento) */
        position: static !important;
      /* los elementos dejarán de ser flotantes */
        float: none !important;
   }
}
```

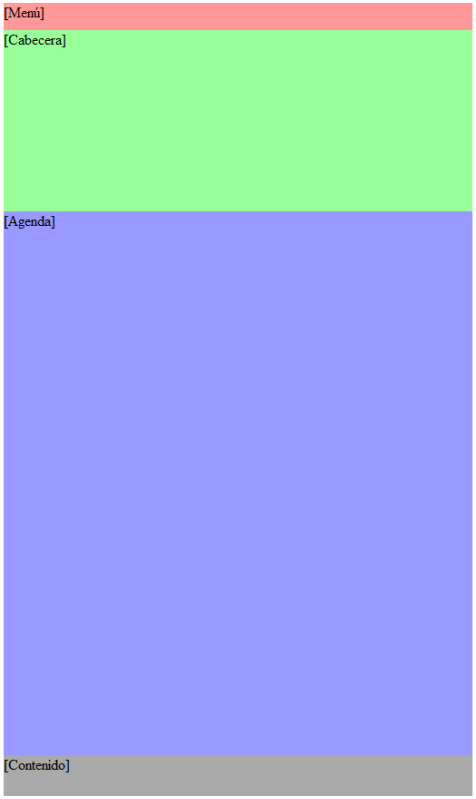
*Sólo se muestra el código de la estructura en cascada (cuando no cabe en el ancho). El resto lo podéis ver en el ejemplo cascada de Aules.

Pasar de columnas a cascada

La página se mostrará así para resoluciones de más de 600 píxeles:



Y así para resoluciones menores:



Adaptar imágenes y vídeos

A veces, puede ser importante usar imágenes distintas, una con un tamaño superior y otra con uno más pequeño, pero vamos a centrarnos en el **uso de una misma imagen** que, en la medida de lo posible, debería de estar **optimizada** y se **adaptará** como queramos.

Podemos hacer que una imagen se comporte de las siguientes formas:

- Ocupar todo el ancho de la página. Lo haremos estableciendo su ancho al 100%.

```
img { width:100%; }
```

- Tener una imagen con un tamaño máximo. En este caso, la imagen tendrá un tamaño que no sobrepasará, pero cuando se disminuya el tamaño del contenedor se encogerá adaptándose a la página.

```
img { width:100%; max-width:400px; }
```

- Tener las imágenes con su tamaño original como máximo. Similar al anterior pero pondremos como tamaño máximo el ancho de la imagen para no deformarla al redimensionarla.

```
img{width:100%; max-width:100%;}
```

picture y srcset

Mostrar diferentes imágenes para diferentes tamaños del navegador

usando <picture> y srcset

```
<picture>

    <source media="(min-width: 500px)"  srcset="car.jpg">
    <source media="(min-width: 300px)" srcset="calabaza-mac.jpg"
    <img src="kikopalomares.jpg">

</picture>
```

La barra de navegación

En las versiones para menores resoluciones o tamaños de pantalla, el menú suele convertirse hoy en día, casi un estándar de facto, en un **icono con tres rayitas** que se despliega. Pasamos de tener un **menú lineal** y visible a un **menú desplegable**.

A continuación, veremos cómo podemos implementar este menú basándonos en un ejemplo.

```
<nav>
   <a href="#">Menú</a>
   <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Collection</a></li>
      <li><a href="#">Blog</a></li>
      <li><a href="#">Contact</a></li>
      <li><a href="#">Directions</a></li>
   </ul>
</nav>
<script src="menu.js"></script>
```

La barra de navegación

A este código le aplicaremos los siguientes estilos:

```
* {
   margin: 0;
   padding: 0;
}
nav {
   width:100%;
   background-color: #fff;
   border-bottom: 1px solid #ccc;
}

nav ul {
   list-style: none;
   padding: 0px;
   margin: 0px;
   font-weight: bold;
   text-align: center;
}
```

```
nav ul li {
      display: inline-block;
      text-align: left;
}
nav ul li a {
      display: block;
      padding: 15px 10px;
      text-decoration: none;
      color: #444;
}
nav ul li a:hover
{
      background-color: #ccc;
}
nav > a {
      display: none;
}
```

Con estos estilos nuestra barra de navegación quedará así:

El enlace con el texto Menú queda oculto (display:none). Si reducimos la ventana del navegador, los enlaces de la barra se reubicarán:

La barra de navegación

Para mejorarlo, vamos a hacer que cuando el tamaño de la ventana sea menor de **475 píxeles, el menú se oculte y se muestre sólo el enlace con el texto Menú**. Después, al hacer click con el ratón sobre este enlace el menú se desplegará y al volver a pulsar se plegará.

Añadiremos la siguiente @media query:

```
@media all and (max-width: 475px)
{
   nav ul { display: none; } /* ocultamos el menu´de navegación */

   nav > a /* mostramos el enlace con el texto Menú */
   {
      display: block;
      padding: 0 1em 0;
      text-align: center;
      padding: 10px 15px;
      color: #fff;
      background-color: #0084B4;
      text-decoration: none;
      margin: 3px;
}
```

La barra de navegación

```
   /* Con la clase desplegado el menú se mostrará verticalmente */
   ul.desplegado
   {
      display: block;
      list-style: none;
   }
   ul.desplegado li
   {
      display: block;
      text-align: center;
   }
   ul.desplegado li a
   {
      display: block;
      border-bottom: 1px solid #ccc;
   }
}
```

Con esta @media query cuando reducimos la ventana el menú se mostrará así:

Ya sólo nos queda hacer que al pulsar el enlace **el menú se despliegue**. Para ello, utilizaremos el siguiente código javascript:

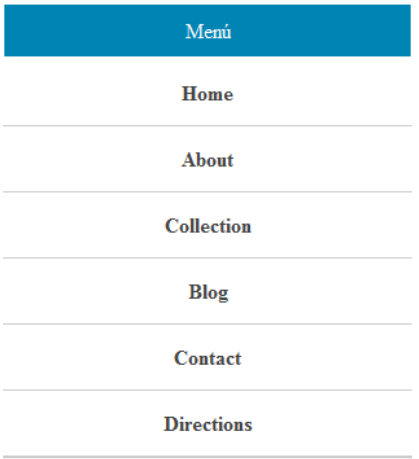
La barra de navegación

```
var enlaceMenu;

function iniciarMenu()
{
   enlaceMenu = document.querySelector("nav>a");
   enlaceMenu.addEventListener("click", despliegaMenu, false);
}

function despliegaMenu()
{
   document.querySelector("nav>ul").classList.toggle('desplegado');
}

window.addEventListener("load", iniciarMenu, false);
```



En el código anterior, al pulsar el enlace menú **se le aplicará la clase desplegado** al <ul> si no la tiene aplicada o se eliminará dicha clase si ya la tiene aplicada. Esto lo hacemos con el método `classList.toggle('desplegado')`.

Podéis ver el ejemplo completamente funcional en la carpeta nav del fichero de recursos.