Práctica T06 - Resposive Web Desig (Canvas)

Javier Beteta

2022-01

1. Páginas Adaptables (Resposible Web Design), introducción

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.

1.1. Origen

Tanto la idea como el propósito del diseño web adaptable fueron previamente discutidos y descritos por el consorcio W3C en julio de 2008 en su recomendación *"Mobile Web Best Practices"* bajo el subtítulo "*One Web*". Dicha recomendación, aunque específica para dispositivos móviles, puntualiza que está hecha en el contexto de "*One Web*", y que, por lo tanto, no sólo engloba la experiencia de navegación en dispositivos móviles, sino también en dispositivos de mayor resolución de pantalla como dispositivos de sobremesa. Hoy en día, la variedad de dispositivos existentes en el mercado ha provocado que la información disponible no sea accesible desde todos los dispositivos, o bien es accesible, pero la experiencia de navegación es muy pobre. ## Ventajas La principal ventaja que encontramos con el diseño web adaptable es que la web se visualizará correctamente en todos los dispositivos que usemos y se adaptará a los giros en dispositivos móviles. Además, *Google* tiene en cuenta las páginas web que tienen diseños sensibles o adaptables, gracias a su *Googlebot-Mobile*. El personal de *Google* recomienda que se use el Responsive Design para crear páginas web que se adapten a todos los dispositivos con el uso de *@media-queries*. La alternativa sería usar distinto código según el agente del dispositivo, o incluso, distintas URLs. *Google* nos incita a usar este método porque, de esta forma, *Googlebot* no necesita analizar tanto contenido de la misma web, lo que le facilita la labor de asignar el tipo de contenido de la página y de esta forma mejorará el *SEO* de la web. Otro factor, que normalmente no tenemos en cuenta a la hora de hacer un diseño, es la accesibilidad del mismo, que dificulta algunas veces el uso de una web a un disminuido visual. Con un diseño adaptable que aproveche el espacio de la página podemos ofrecer una página de calidad para este colectivo. A continuación, veremos lo esencial para crear un diseño web adaptable: el meta-tag Viewport, las **@media queries**, etc. ## Viewport Esta *meta-etiqueta* fue creada en principio por *Apple* para su móvil predilecto, pero se ha convertido en todo un estándar que es soportado por la mayoría de los dispositivos móviles (*smartphones*, *tablets* y gran parte de móviles de gama media y baja). 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). Como siempre, 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): - *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 en la página. - *maximum-scale*: zoom máximo que podemos hacer en la página. - *user-scalable*: establece si está permitido o no hacer zoom (yes/no).

1.2. 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: - La primera, y la más importante, es dejar de usar píxeles en todos los sitios, 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 será 31,25 %. - Para limitar el ancho (o alto si se terciara) debemos de usar el parámetro *max-width* (*max-height* en el caso del alto). Para establecer el mínimo usaremos *min-width* y *min-height*, aunque estos se utilizan menos. El valor de estas propiedades sobrescribirá el valor de las propiedades *width* y *height*. Por ejemplo, si indicamos *width*: 100 % y *max-width*: 600px, cuando visualicemos la página con una resolución de 1200px, el ancho será de 600px, ya que es el valor máximo que hemos indicado para el ancho (puedes ver un ejemplo de uso en el fichero *max-width.html* de la carpeta de recursos). En este ejemplo, tenemos cuatro elementos configurados con distintas combinaciones de *width* y *max-width*. Si vamos cambiando el ancho de la ventana del navegador, veremos cómo los anchos se van adaptando hasta llegar a los anchos máximos configurados, momento en el cual, el ancho permanecerá invariable. En la siguiente imagen podemos observar cómo se muestra el ejemplo.

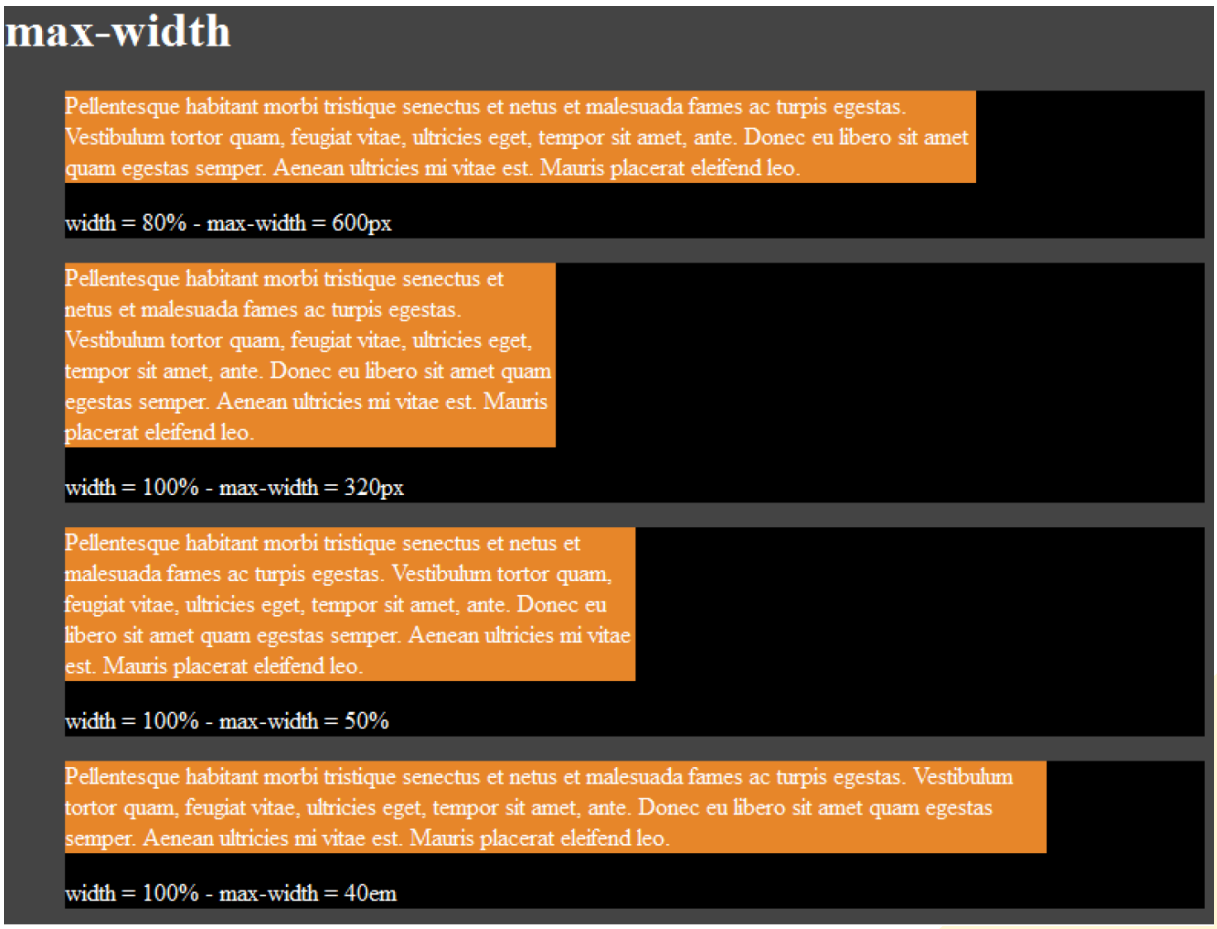

Figura 1: propiedad *max_width*

- 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**.
- Tampoco debemos permitir que las imágenes y los vídeos se salgan de la estructura, sino aparecerá un scroll lateral en los dispositivos móviles que destrozará totalmente el diseño.

1.2.1. media-queries

Desde los días de CSS 2.1, nuestras hojas de estilos han disfrutado de “una conciencia del dispositivo” a través de los *media type**. Por ejemplo cuando creamos una hoja de estilos para impresión, estamos utilizando este concepto:

```
<link rel="stylesheet" type="text/css" href="core.css" media="screen"
↪  />
<link rel="stylesheet" type="text/css" href="print.css" media="print"
↪  />
```

Esperando que utilizáramos este concepto para diseñar algo más que “bonitos formatos de páginas para imprimir”, la especificación CSS nos suministró un grupo de media types aceptable, cada uno de ellos diseñado para una clase específica de dispositivo listo para la web. Pero la mayoría de los navegadores y los dispositivos no se han adherido al espíritu de la especificación, dejando varios *media types* implementados imperfectamente, o completamente ignorados. Por suerte, el *W3C* creó las **media queries** como parte de la especificación CSS3, mejorando la promesa de los *media types*. 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. Por ejemplo, siguiendo el reciente crecimiento de WebKit mobile, 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 los 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" />
```

La query contiene dos componentes: Un media type (screen), y la consulta entre paréntesis, que contiene una característica a inspeccionar (*max-device-width*) seguida por el valor al que apuntamos (480px). En otras palabras, le estamos preguntando al dispositivo, si su resolución horizontal (*max-device-width*) es igual o menor que 480px. En el caso de que visualicemos 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. En el pasado, este tipo de cosas necesitaban de soluciones *javascript*, pero la especificación de **media query** provee una serie de características del *medio* que se extienden mucho más allá de la resolución de la pantalla, ampliando el alcance de lo que podemos testear con nuestras queries. Además, podemos testear múltiples valores de las propiedades 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" />
```

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. Con esto, disponemos de nuevas capacidades que nos permiten definir conjuntos de estilos dependiendo de propiedades comunes de los dispositivos que acceden a nuestros sitios. Propiedades como el alto y el ancho, la relación de aspecto o el número de colores disponible. Las reglas *@media* pueden ser utilizadas para adaptar nuestras páginas, no solo para dispositivos comunes, sino para todo tipo de dispositivos que nuestros lectores usen para visitarlas. Aunque las *@media* queries nos permite conocer muchas propiedades diferentes, las más importantes son las siguientes: - *aspect-ratio*: Hace referencia a las dimensiones relativas del dispositivo expresadas como una relación de aspecto: 16:9 por ejemplo. - *width* y *height*: Se refiere a las dimensiones del área de visualización. Además pueden ser expresadas en valores mínimos y máximos. - orientation: Detecta si el layout es panorámico (el ancho es mayor que el alto) o vertical (el alto es mayor que el ancho). Esto nos permite ajustar los diseños para dispositivos con propiedades de giro de la pantalla. - resolution: Indica la densidad de los pixeles en el dispositivo de salida. Esto es especialmente 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.

1.3. Puesta en marcha

Ya hemos visto lo básico que necesitamos conocer sobre el **Responsive Design**, pero, la mayoría de las veces, lo difícil es aplicarlo de la forma que necesitamos o mostrar el contenido de una forma eficaz sin perder en diseño ni usabilidad, y en esto es en lo que nos centraremos ahora, en estructurar correctamente el contenido de una página que se adapte a distintos tipos de pantalla. Para empezar veremos cómo hacer fluir elementos como imágenes, bloques de texto o cualquier cosa en general, de forma que se reubiquen los elementos 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 margen. - *inline-block*: esta última forma de visualizarse un elemento es una mezcla de las dos anteriores, comportándose como un bloque, pero poniéndose en línea con otros elementos iguales.

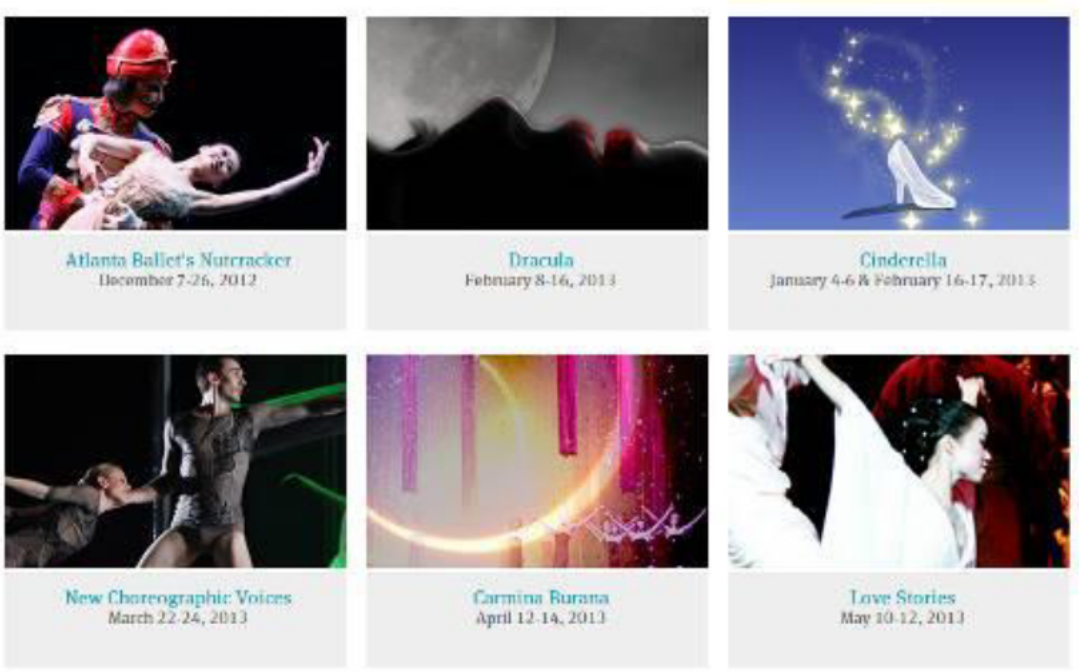

Figura 2: Bloques de imágenes

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:

```
<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.

```
#contenedor .bloque
     {
          display: inline-block;
          height:300px;
```

```
        width: 300px;
        border:1px solid #333;
        background: #999;
        margin:20px;
    }
```

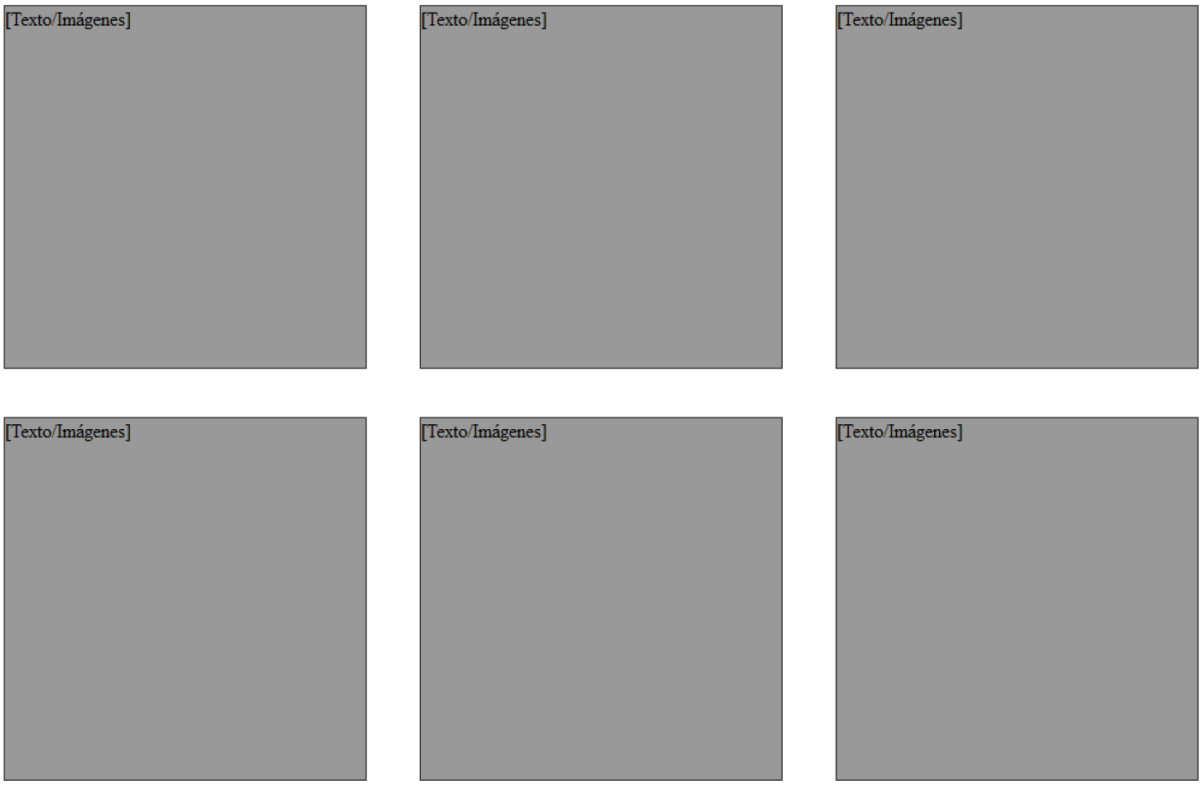

Figura 3: Bloques según el ejemplo

Podéis ver este ejemplo en la carpeta *bloques1* del fichero de *recursos*. Si probáis a ampliar y reducir el tamaño de la ventana de vuestro navegador, veréis que los elementos se van reubicando automáticamente gracias a la propiedad *display*. Muy bien, 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? En ese caso, utilizaremos las **media queries**. Por ejemplo, si quisiéramos que a partir de los 800 píxeles 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;
}
```

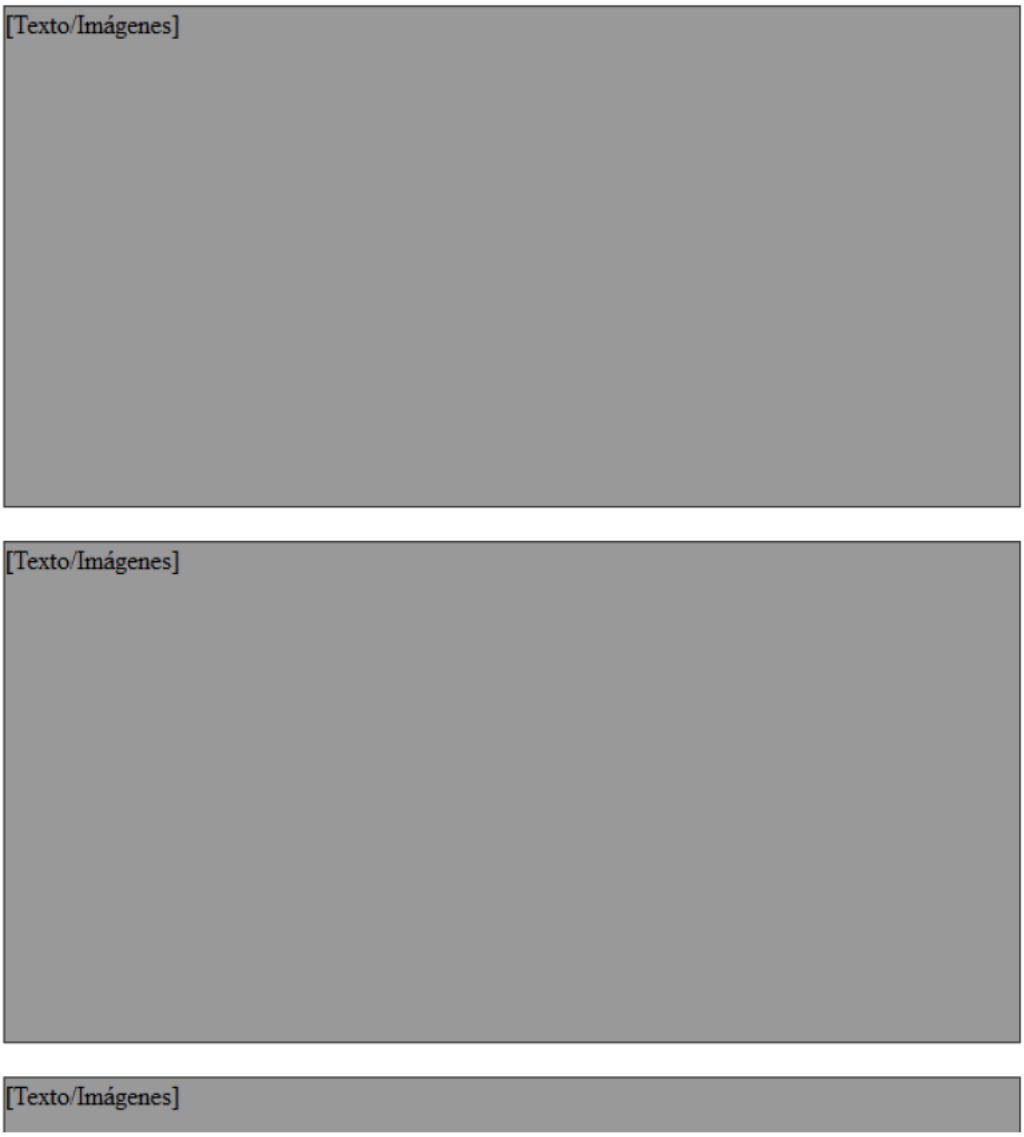

Figura 4: Imagenes en bloque

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. Podéis ver este ejemplo en la carpeta *bloques2* del fichero de recursos. 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 los de dentro:

por lo tanto, el resultado será que 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* se aplica dos veces (*display: inline-block; display: block !important;*) y lo mismo ocurre con la propiedad width ¿cómo sabrá el navegador cuál de los dos aplicar? en estos casos, lo mejor que podemos hacer es indicar la palabra *!important* en aquellas propiedades que queramos que tengan preferencia sobre el resto. De esta forma, nos aseguramos que el navegador aplica la propiedad que nos interesa. Sólo es necesario que lo indiquemos en aquellas propiedades que se repitan en la **media query** y fuera de ella, en las que no se repiten, no será necesario.

1.4. Pasar de columnas a 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 funciún del ancho disponible se cambia la disposición.

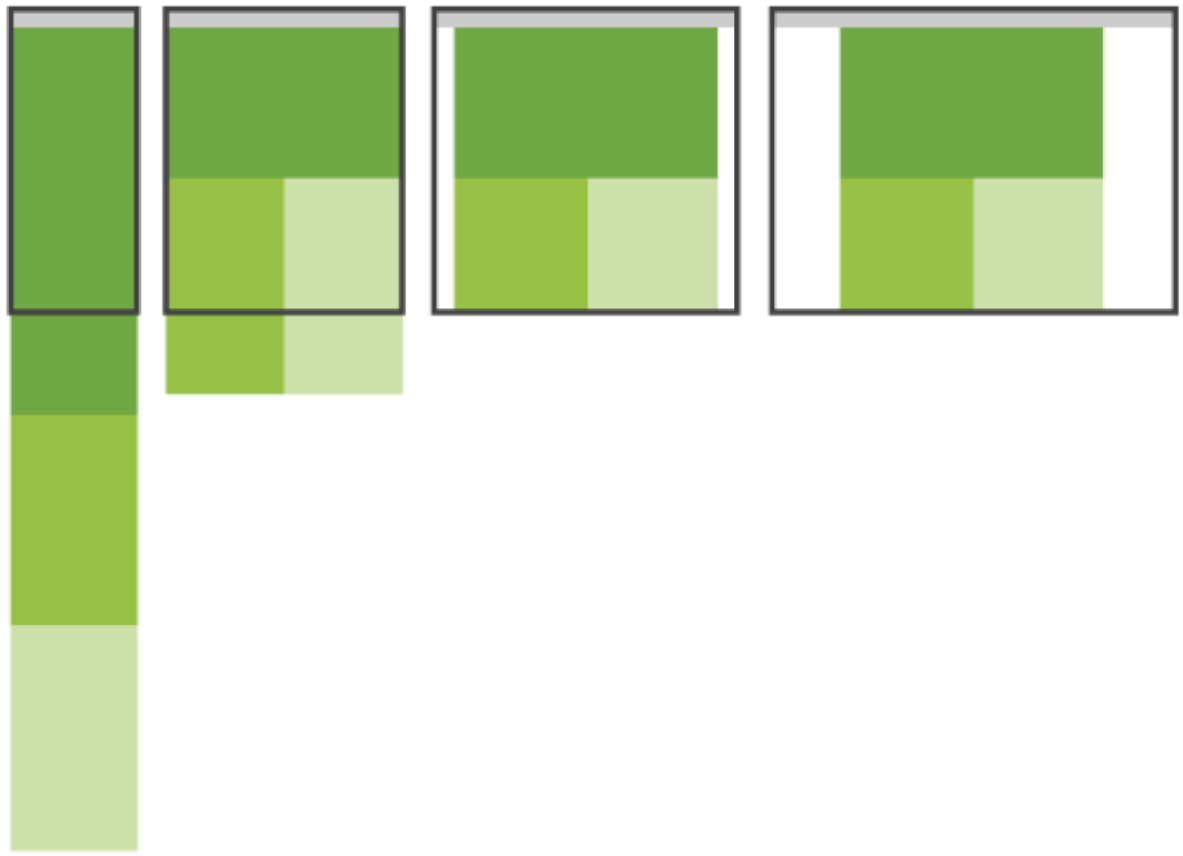

Figura 5: Estructura Web Adaptativa

En la siguiente *url* Senado EEUU podéis ver un ejemplo que utiliza esta técnica. A continuación, veremos cómo hacer una estructura adaptable similar a la de la imagen, que estaría formada por un título con un menú, una cabecera y dos columnas, la primera con una agenda con eventos y la segunda columna con los contenidos. Lo que haremos para que sea totalmente adaptativo es realizar pequeños cambios según las necesidades, pero, para el ejemplo, sólo tendremos dos estados: la estructura bien maquetada y la estructura en cascada, como se representa en la imagen anterior. Nuestro *html* será el siguiente:

```
<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;
     }
}

nav{
     background: #F99;
     position: fixed;
     top: 0px;
     left: 0px;
     width: 100%;
     height: 30px;
}

.pagina{
     /* la página será como máximo de 1000px de ancho */
     max-width: 1000px;
     margin:auto;
     }

header{
     background: #9F9;
     margin-top: 30px;
     height: 200px;
     }

aside{
     background: #99F;
     float: left; /* flotamos el elemento a la izquierda */
     width: 35%;
     height: 600px;
     }
```

```
main{
      background: #AAA;
      width: 65%;
      margin-left: 35%;
      height: 900px;
      }
```

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

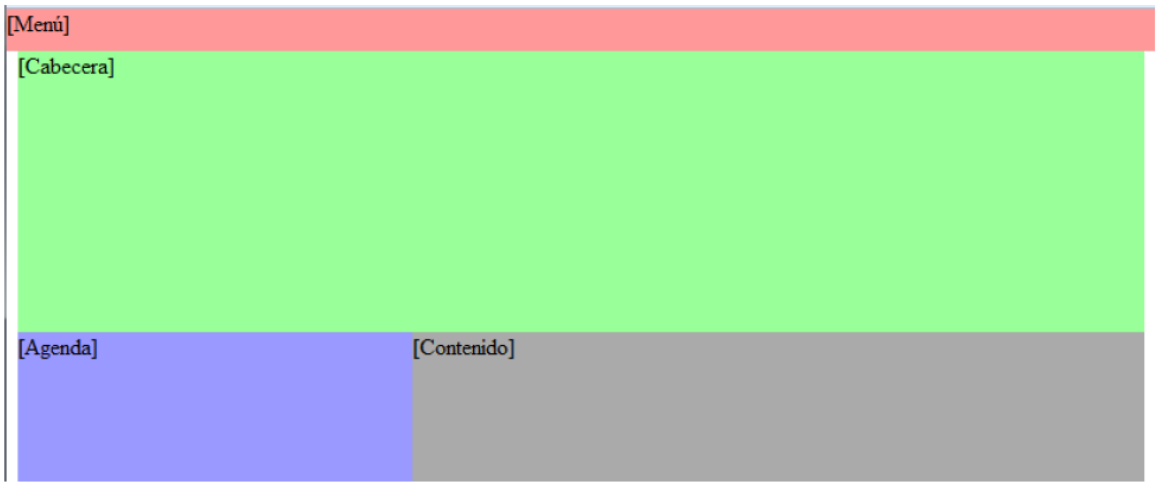

Figura 6: En resoluciones mayores de 600

Y así para menores:

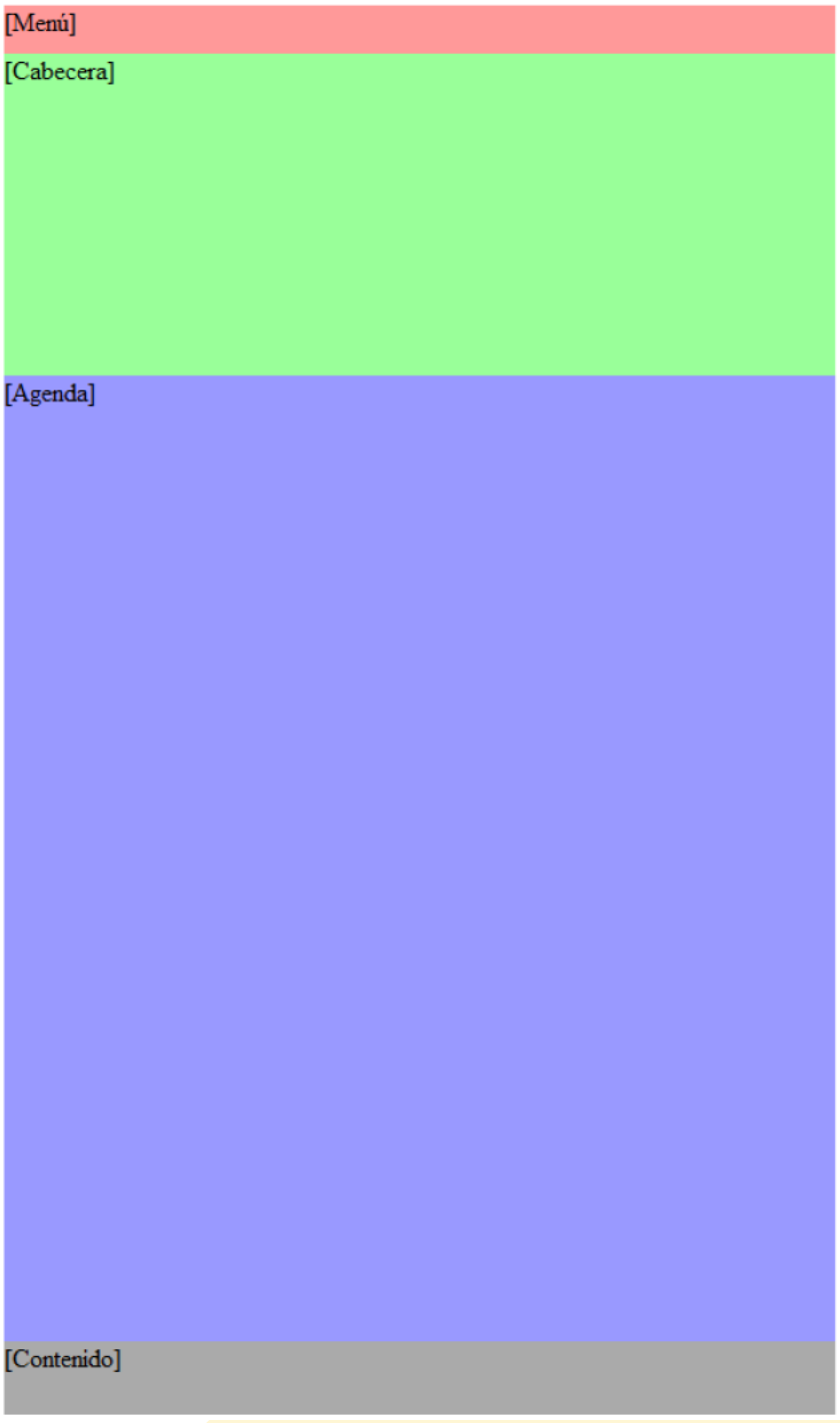

Figura 7: En resoluciones menores de 600

Podéis ver este ejemplo en la carpeta cascada del fichero de recursos. Para probar el ejemplo, simplemente, tenéis que cargar la página en el navegador y reducir la ventana hasta que tenga una resolución

inferior a 600 píxeles.

1.4.1. Adaptar imágenes y vídeos

Muchas veces, para crear diseños muy optimizados puede ser importante usar varias 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 nosotros queramos. Podemos hacer que una imagen se comporte de las siguientes formas: - Ocupar el ancho de la página. Puede ser útil para cabeceras o imágenes principales y 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. Este caso es muy similar al anterior, con la diferencia que pondremos como tamaño máximo el ancho de la imagen. De esta forma, la imagen no se deformará al ser redimensionada a un tamaño superior al original.

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

Para adaptar el tamaño de los vídeos usaremos las mismas técnicas que utilizamos para las imágenes. Una cosa a tener en cuenta, es que si hemos puesto un alto fijo a un vídeo, y después, en una **@media query**, ponemos, por ejemplo, un ancho relativo utilizando porcentajes, el elemento **<video>** se redimensionará para mantener las proporciones al redimensionar la ventana, pero el espacio reservado para el alto inicial se mantendrá reservado, por lo que los elementos que se encuentren alrededor no se reubicarán. Para solucionar este problema, bastará con indicar en la **@media query** que el alto será automático (*height:auto*).

1.4.2. Labarra 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. Tenemos el siguiente código html:

```
<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>
```

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 como se muestra en la siguiente imagen:

Figura 8: Barra de menú

Se puede observar que el enlace con el texto Menú queda oculto (*display:none*). Si reducimos la ventana del navegador, los enlaces de la barra se reubicarán de la siguiente forma:

Figura 9: Barra de menú reducida

lo cual, podría no ser deseable. 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 plegara.

Añadiremos la siguiente **@media query**:

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

Figura 10: Imagen menú colapsado

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

```
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')*.

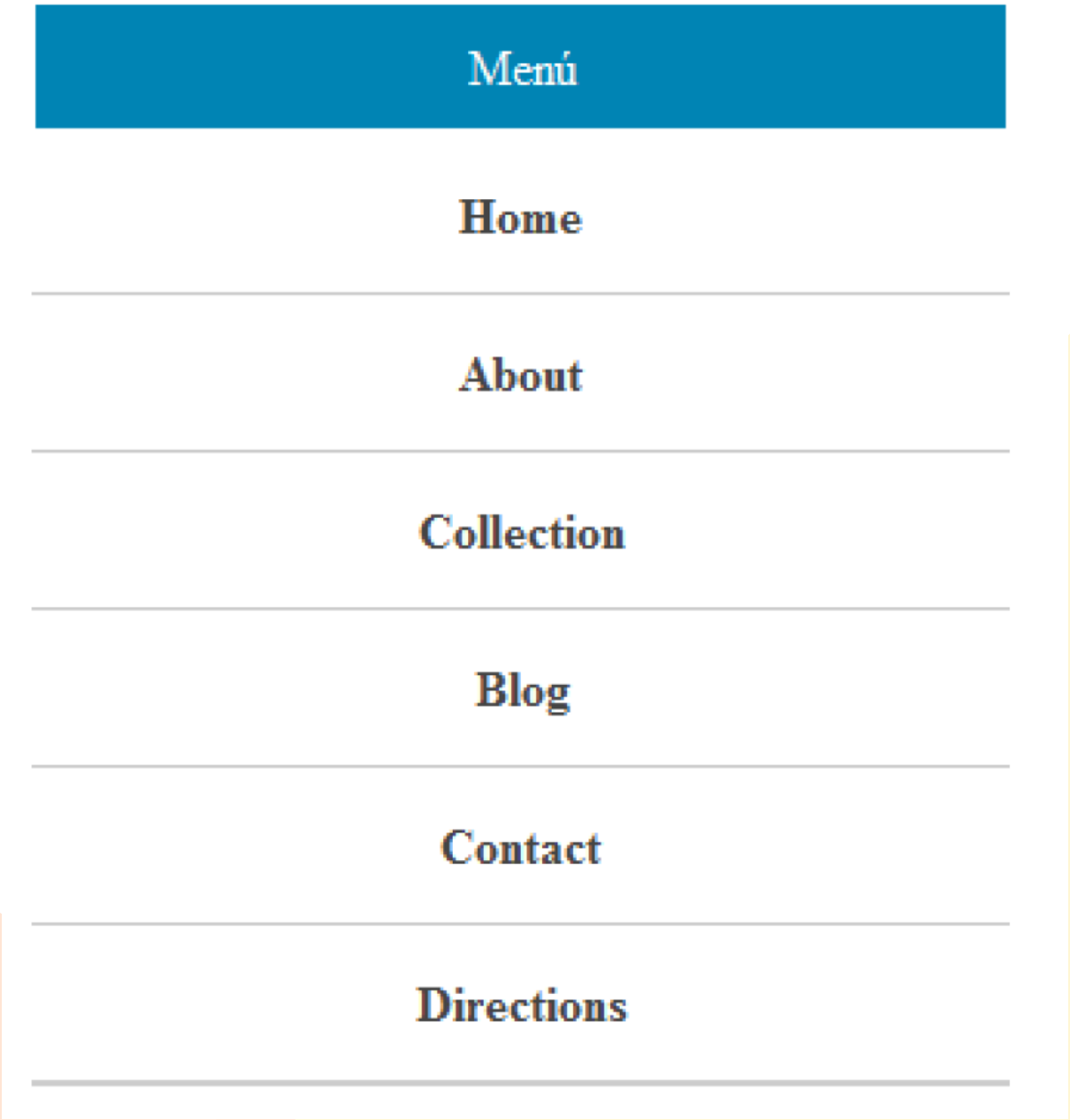

Figura 11: Menú desplegado

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

2. Ejercicios

2.1. Ejercicio 1

Vamos a modificar nuestro sitio web para convertirlo en un sitio web adaptable. Comenzaremos por añadir el meta *viewport* a todos los documentos: *index.html*, *videos.html*, *contacto.html* y *serpiente.html*.

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

2.2. Ejercicio 2

En este ejercicio vamos a preparar nuestra página *index.html* para que se adapte al tamaño de la ventana. Trataremos de que nuestra página se vea correctamente en cualquier resolución de pantalla, con lo que conseguiremos que se vea bien en todos los dispositivos, ordenadores, portátiles, tablets y/o móviles. Para ello, crearemos tres **@media querys**:

- La primera con un ancho máximo de 1024 píxeles. En esta **@media query** haremos los siguientes cambios:
  - El ** ** tendrá un ancho del 90 % y un ancho mínimo de 320 píxeles.
  - El **<header>** tendrá un tamaño de fuente de 0.7em.
  - Los **<li>** del menú existente en la barra de navegación *(#page>nav>ul>li)* tendrán los siguientes márgenes externos: 0 % el superior, 8 % el izquierdo y el derecho y 2 % el inferior.
- La segunda con un ancho máximo de 700 píxeles. En esta **@media query** haremos los siguientes cambios:
  - Los **<li>** del menú existente en la barra de navegación *(#page>nav>ul>li)* tendrán los siguientes márgenes externos: 0 % el superior, 4 % el izquierdo y el derecho y 1 % el inferior.
- La tercera con un ancho máximo de 475 píxeles. En esta **@media query** prepararemos nuestra barra de navegación para que se oculte y se pueda desplegar como hemos visto en el tema. Antes de nada prepararemos nuestro documento:
  - Añadiremos un enlace vacío dentro del **<nav>**, justo antes del **<ul>**: ****
  - Enlazaremos el fichero de script *menu.js* que podéis encontrar en la carpeta scripts del fichero de recursos. `html <script src="scripts/menu.js"></script>`

- En el fichero de estilos principal (*estilos.css*) ocultaremos (*display:none*) el enlace que hemos introducido anteriormente en el **<nav>** para que, inicialmente, este oculto y sólo aparezca cuando el ancho de la ventana sea inferior a 475 píxeles.

A continuación, crearemos la **@media query** siguiendo los pasos que se han explicado durante el tema. Tenemos que tener en cuenta que: - Los márgenes de los **<li>** de la barra de navegación estarán a 0. - El enlace vacío que hemos introducido tendrá las siguientes características: + Tendrá la imagen de fondo *menu.png* que podéis encontrar en la carpeta *imgs* del fichero de recursos. + La imagen no se repetirá y estará centrada y a 10 píxeles de distancia del borde superior. + El ancho y alto del enlace será de 80px. + El enlace tendrá unos márgenes externos superior e inferior de 0 y el izquierdo y derecho serán automáticos. + Debes crear la clase .desplegado, tal y como hemos visto en el tema.

Estas **@media querys** las crearemos en el fichero estilos.css. De esta forma, se aplicarán a todos los documentos de nuestra web.

2.3. Ejercicio 3

En este ejercicio vamos a convertir en adaptable la página *videos.html*. Para ello, haremos los siguientes cambios en el fichero *reproductor.css*: - En el elemento *#reproductor*: + Eliminaremos la flotación y lo mostraremos como bloque en línea (inline-block). + El ancho será del 95 % con un ancho máximo de 720 píxeles. + Tendrá unos márgenes externos superior e inferior de 20 píxeles y el izquierdo y derecho serán automáticos. - Crearemos una regla de estilo para el elemento **<video>**: + El ancho será del 95 % con un ancho máximo de 720 píxeles. + El alto será automático. + Los márgenes externos serán de 5 píxeles. - En la barra de navegación del reproductor: + Eliminaremos la flotación y lo mostraremos como bloque en línea (inline-block). + Eliminaremos el ancho. + La alineación vertical será arriba (*vertical-align:top*). - En el elemento #barra: + El ancho será del 95 % con un ancho máximo de 705 píxeles. + Eliminaremos el margen interno. Finalmente, crearemos una **@media query** para un tamaño máximo de 970 píxeles. En esta **@media query**, simplemente, indicaremos que los botones de la barra de navegación del reproductor se muestren como bloques en línea (*inline-block*). Una vez finalizado el ejercicio nos encontraremos con un problema, y es que, si recordamos, en el tema 3 iniciíbamos la variable máximo en la función iniciar y le dábamos un valor de 700. Esta variable la utilizábamos para calcular el ancho de la barra de progreso del vídeo en las funciones redimensionaBarra y desplazarMedio. Ahora el ancho de esta barra no es fijo y dependerá del tamaño de la ventana, por lo que, en lugar de iniciar la variable máximo al principio, lo que haremos será calcular el ancho cada vez que lo necesitemos. Las funciones redimensionaBarra y desplazarMedio quedarán así:

```
function redimensionaBarra() {
     if(!medio.ended) {
          var maximo=parseInt(getStyle('barra', "width"));
          var total=parseInt(medio.currentTime*maximo / medio.duration);
```

```
            progreso.style.width=total+'px';
            }
      else {
            progreso.style.width='0px';
            play.value='\u25BA';
            window.clearInterval(bucle);
      }
}
function desplazarMedio(e) {
      if(!medio.paused && !medio.ended){
            var ratonX=e.pageX-barra.offsetLeft;
            var maximo=parseInt(getStyle('barra', "width"));
            var nuevoTiempo=ratonX*medio.duration/maximo;
            medio.currentTime=nuevoTiempo;
            progreso.style.width=ratonX+'px';
      }
}
```

Sólo nos falta la función *getStyle*. Esta función es necesaria porque estamos obteniendo el valor de la propiedad de estilo width que no se inicia con un valor fijo, sino que tiene un valor relativo al tamaño del elemento que la contiene (recordad que hemos puesto un ancho del 95 %). Para obtener el valor del ancho de la barra en un momento determinado, tenemos que llamar al método *window.getComputedStyle*. Este método nos devolverá el valor calculado de una propiedad en un momento determinado. Con todo esto, crearemos la función *getStyle* de la siguiente forma:

```
      function getStyle(nombreElemento, nombrePropiedad){
            var elemento = document.getElementById(nombreElemento);
            return win-
             ↪  dow.getComputedStyle(elemento,null).getPropertyValue(nombrePropiedad);
      }
```

2.4. Ejercicio 4

En este ejercicio convertiremos en adaptable la página contacto.html. Para ello, haremos los siguientes cambios en el fichero contacto.css: - Crearemos una **@media query** para un tamaño máximo de 1024 píxeles en la que modificaremos las siguientes reglas: + El elemento *#contenedorFormulario*: - Márgenes externos del 4 %. - Ancho del 58 %. - Márgenes internos superior e inferior del 2 % e izquierdo y derecho del 4 %. + El elemento

del #contenedorFormulario: - Posición estática. - Márgenes internos superior e inferior del 0 % e izquierdo y derecho del 1 %. - Márgenes externos superior e inferior del 2 % e izquierdo y derecho del 0 %. + Los input que no sean submit: - Alto del 5 %. - Ancho del 55 %. + Los input que no sean submit

cuando tienen el foco: - Ancho del 61 %. + El elemento **<textarea>**: - Ancho del 75 %. + El submit: - Se mostrará como un elemento en bloque *(display:block)*.

2.5. Ejercicio 5

Para terminar, convertiremos en adaptable la página *serpiente.html.* Para ello, haremos los siguientes cambios en el fichero *videojuego.css*: - Crearemos una **@media query** para un tamaño máximo de 1024 píxeles en la que modificaremos las siguientes reglas: + El elemento #contenedorVideojuego: - Márgenes externos del 5 %. - Márgenes internos superior e inferior del 2 % e izquierdo y derecho del 5 %. - Ancho del 80 %. + El elemento **<canvas>**: - Ancho del 80 %.

Con esto habremos terminado los ejercicios del tema. En el aula virtual podéis encontrar un vídeo de demostración en el que podéis ver cómo debería quedar nuestra página web después de finalizar los ejercicios.