PEC 3

HTML y CSS: Universitat Oberta de Catalunya

Ignacio Casares Ruiz

Enero de 2023

Pregunta 1

Sobre técnicas de posicionamiento y modos de diseño, responded a las siguientes preguntas:

Explica los siguientes tipos de posicionamiento: estático, relativo, absoluto, fijo y sticky.

- **Estático**: representa el valor de posicionamiento por defecto. Esto implica que el elemento no es afectado por las propiedades `top`, `left`, `right` y `bottom` y que su posición queda fijada según el flujo estándar de la página.
- **Relativo**: esto quiere decir que el elemento se posiciona según el flujo habitual de la página pero que podrá ser modificado a través de las propiedades `top`, `left`, `right` y `bottom` sin que esto afecte la posición del resto de elementos a su alrededor.
- **Absoluto**: el elemento no se sitúa en el flujo normal de la página, por lo que no ocupa espacio respecto al resto de elementos. Se coloca de forma relativa al ancestro más cercano que ocupa una posición que no sea `static`. Si este no existe, se coloca de forma relativa al bloque de contenido inicial.
- **Fijo**: de forma similar al anterior, el elemento no se sitúa en el flujo normal de la página. Esta vez se coloca de forma relativa al bloque de contenido inicial según el *viewport*. Este permanece en la misma posición incluso cuando se hace *scrolling*.
- ***Sticky***: la posición del elemento se conforma al flujo normal de la página. La posicion final es determinada a través de las propiedades `top`, `left`, `right` y `bottom`, sin afectar la posición del resto de elementos. En este caso, el elemento se pega al ancestro más cercano que tenga un mecanismo de *scroll*. El elemento, por tanto, obtendrá una posición específica que no será afectada por este mecanismo. Su posición es relativa hasta que dadas ciertas circunstancias se queda pegada en el sitio, actuando de forma similar al posicionamiento fijo.

Explica para qué sirven las siguientes propiedades del módulo CSS Grid: grid-template-columns, grid-template-rows y gap.

- `grid-template-columns`: se trata de una propiedad que permite configurar la disposición de las columnas del módulo *grid*. Acepta diversos valores que permiten variar su situación como elemento en la página.
- `grid-template-rows`: de forma similar a la propiedad anterior, permite configurar la disposición de las filas dle módulo *grid*. Sus valores tienen como consecuencia la variación de la disposición de las filas dentro del *grid*, modificando el tamaño de las mismas individual o propocionalmente
- `gap`: aún dentro del *grid layout*, la propiedad `gap` o `grid-gap` permite modificar las características de los espacios que se hallan entre los elementos que configura

las columnas y las filas. Puede hacerse de forma más específica para columnas o filas, según se desee, a través de las propiedades `column-gap` y `row-gap`.

Explica para qué sirven las siguientes propiedades del módulo CSS Flexbox: flex-direction, flex-wrap y order.

- `flex-direction`: dentro de un contenedor que posee la declaración `display: flex`, `flex-direction` permite configurar la disposición de los elementos dentro del mismo. Por ejemplo, `flex-direction: row;` mostrará los elementos en disposición de fila, mientras que `flex-direction: column;` lo hará en forma de columna.
- `flex-wrap`: dentro de un contenedor que posee la declaración `display: flex`, `flex-wrap` permite modificar el comportamiento de los elementos cuando estos se plegan en más de una línea por modificaciones del tamaño del dispositivo. Esta propiedad puede modificarse mediante el valor `nowrap` para evitar que esto ocurra, o a través de `wrap-reverse` de forma que los elementos se coloquen en la dirección opuesta cuando se dé este fenómeno.
- `order`: esta propiedad permite especificar el orden de un elemento dentro de una flexbox. Funciona por un modelo de índices que empieza por 0 (este supondría la primera posición), y acepta valores negativos en el posicionamiento.

Pregunta 2

Sobre diseño responsivo, responded a las siguientes preguntas:

Explica qué se entiende por mobile first.

Se refiere al tipo de diseño y programación que implica el comienzo del desarrollo teniendo en cuenta dispositivos de menor tamaño (móviles), a partir del cual se trabaja en mayores tamaños de *viewport* de forma que la web sea siempre visualizable y funcional en los espacios más pequeños. Las características y funcionalidades de la página se implementan y conciben de forma que todo el contenido sea accessible a través del dispositivo móvil, y a partir de ahí se aprovecha la mayor cantidad de espacio para distribuirlo en dispositivos de mayor tamaño, lo cual resulta más sencillo que si se hace de la forma contraria. Esto no implica necesariamente que en los dispositivos de mayor tamaño posean la misma cantidad de funcionalidades: es realtivamente común añadir funcionalidades extra en dispositivos de mayor tamaño, pero el proyecto se concibe de forma que se asegura que el usuario que utilice un móvil, que a menudo dispone de una conexión a internet y memoria menos ventajosas, pueda interacturar con la página de forma plena y adaptada.

Explica por qué es importante el uso de la etiqueta meta viewport.

La etiqueta `meta` con el atributo `viewport` permite describir las características del dispositivo para modificar el tamaño y la forma del *viewport*. Algunos dispositivos móviles o de pequeño tamaño renderizan las páginas a través de una ventana virtual o *viewport*, que es habitualmente más ancha que la pantalla, para más tarde reducir el tamaño de forma que todo pueda verse a la vez. Los usuarios pueden entonces hacer zoom en las diferentes áreas de la página.

Esto se hace de esta forma porque no todas las páginas están optimizadas para ser visitadas a través de un dispositivo de tamaño reducido. Esta forma permite que este tipo de páginas se vean mejor desde un dispositivo con estas características.

Explica a partir de un ejemplo de código cómo los atributos `srcset` y `sizes` del elemento `img` permiten mostrar imágenes diferentes según el dispositivo.

`srcset` y `sizes` funcionan de forma conjunta: `srcset` indica la localización de la URL de las imágenes, mientras que `sizes` especifica el tamaño de las imágenes según ciertas condiciones del navegador. `sizes` necesita de una condición `media`, que deberá colocarse entre paréntesis, seguida de un espacio y el tamaño deseado para la imagen; se pueden añadir varios elementos, y cada uno puede ir acompañado de su condición, separados por comas. `srcset` especifica la ruta en la que se encuentra el archivo desde la situación del archivo `html` que se está editando, seguido de la anchura intrínseca (el tamaño real) de la imagen en píxeles (denotada mediante la unidad `w`). Un ejemplo sería:

```
<img
  srcset="imagenpequeña.jpg 400w, imagengrande.jpg 800w"
  sizes="(max-width: 600px) 400px,
         800px"
  src="imagengrande.jpg"
  alt="Ejemplo de texto alternativo" />
```

Los navegadores que no soportan estos atributos simplemente utilizarán la imagen especificada en el atributo `src`.

`srcset` también puede colocarse como atributo dentro de un elemento `source` acompañado igualmente del atributo `media`, el cual especifica las condiciones a cumplir para que se utilice la imagen descrita en `source`. Todo esto se comprende dentro de un elemento `picture`, de la siguiente forma:

```
<picture>
    <source media="(max-width: 799px)"
    ↪  srcset='img/Mémoire_sur_la_polyphagie_498px.jpg'>
    <source media="(min-width: 800px)"
    ↪  srcset='img/Mémoire_sur_la_polyphagie_640px.jpg'>
```

```
    <img class="main" src='img/Mémoire_sur_la_polyphagie_640px.jpg'>
</picture>
```

En este caso, el navegador mostrará el archivo situado en la primera ruta especificada cuando la anchura máxima del *viewport* sea de 799 píxeles, mientras que, a partir de ese tamaño, al ser la anchura mínima 800px, mostrará el archivo situado en la segunda ruta.