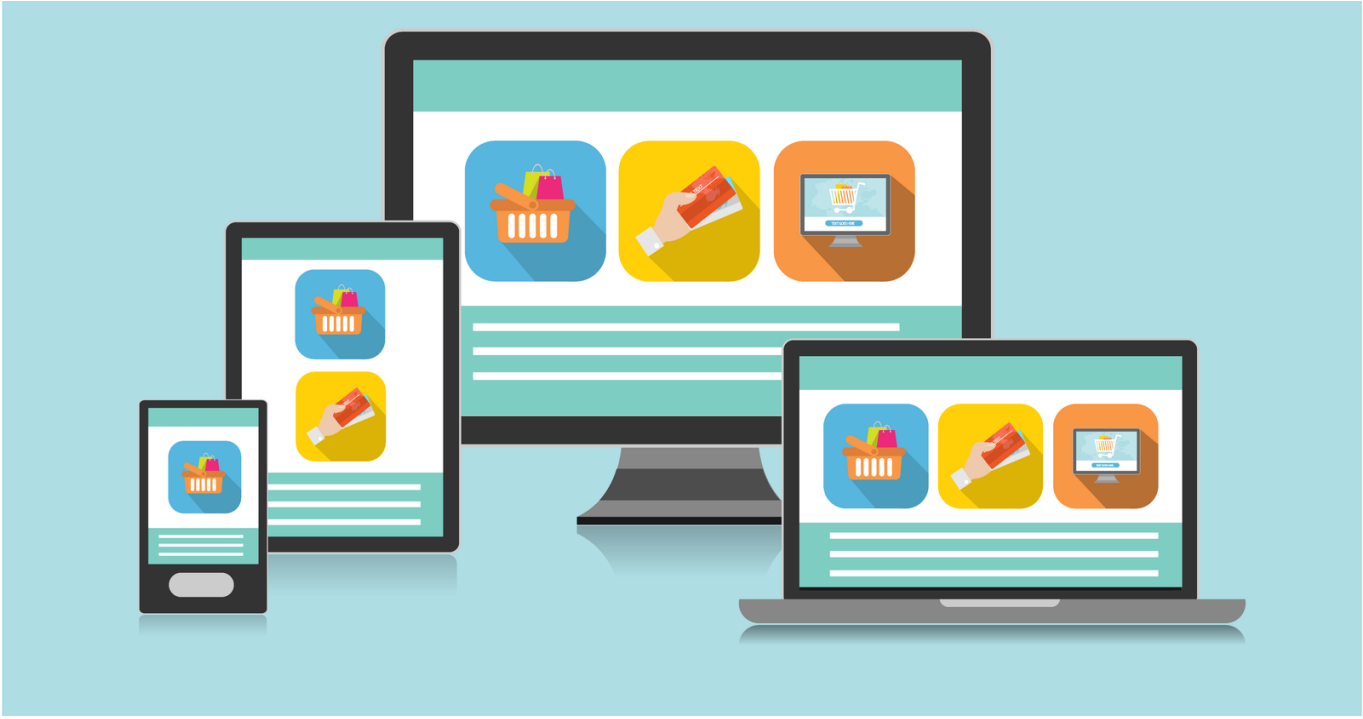
TEMA 6.1. Diseño Web Responsive y Adaptativo

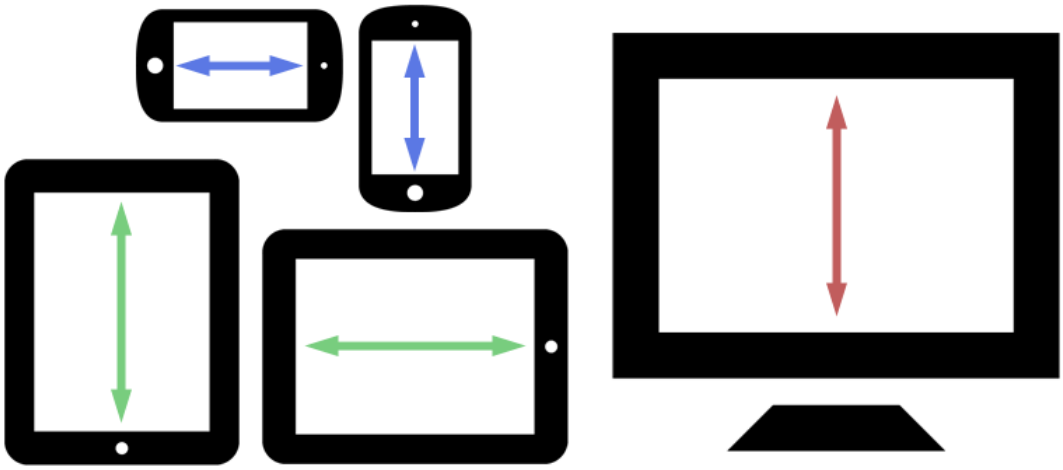


Contenido

1. Responsive Web Design

Una de las características que más se ha popularizado en los últimos años es el **diseño adaptativo** (*Responsive Web Design o RWD*). Se les denomina así a los diseños web que tienen la capacidad de adaptarse al tamaño y formato de la pantalla en la que se visualiza el contenido, respecto a los diseños tradicionales en los que las páginas web estaban diseñadas para un tamaño o formato específico.

De esta forma, se ha comenzado a escapar de los clásicos tiempos en los que se creaban varias versiones de una misma página para diferentes navegadores o resoluciones, donde el principal inconveniente para el desarrollador era que tenía que mantener varias versiones diferentes donde debía hacer los mismos cambios de forma repetitiva o complicaba el desarrollo en general.



El Responsive Web Design es algo muy útil en la era en que vivimos, donde existen dispositivos de todo tipo: smartphones de múltiples gamas y tamaños, tablets, sistemas de escritorio, etc. y sería inviable y poco práctico realizar una versión para cada uno de estos dispositivos.

Aunque en principio el concepto de web adaptativa es muy sencillo de comprender, aplicarlo puede ser todo un quebradero de cabeza si no se conocen bien las bases y se adquiere experiencia. Aquí os dejo algunas webs que destacan algunas interfaces web por su diseño responsive y adaptativo:

- https://www.pipedrive.com/en/blog/responsive-website-examples
- https://www.wix.com/studio/blog/responsive-website-design-examples#viewer-c6sbp50021
- https://www.justinmind.com/web-design/responsive-website-examples

2. Estrategia adaptativa

Antes de comenzar a crear un diseño web adaptativo, es importante tener claro una serie de conceptos:

- Conoce las diferentes resoluciones más utilizadas
- Conoce el público de tu sitio web y tu objetivo
- Elige una estrategia acorde a lo anterior

En primer lugar, es importante conocer los formatos de pantalla más comunes con los cuales nos vamos a encontrar. Podemos consultar páginas como ***Screen Sizes,*** donde se nos muestra un listado de dispositivos categorizados en smartphones, tablets o pantallas de escritorio con las características de cada dispositivo: ancho, alto, sistema operativo, etc...

Debes decidir la estrategia de desarrollo web que quieres utilizar. Aunque existen otras estrategias, hay dos vertientes principales:

Mobile first	Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
Desktop first	Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

Bien, si ya tenemos la teoría clara, toca volver al código.

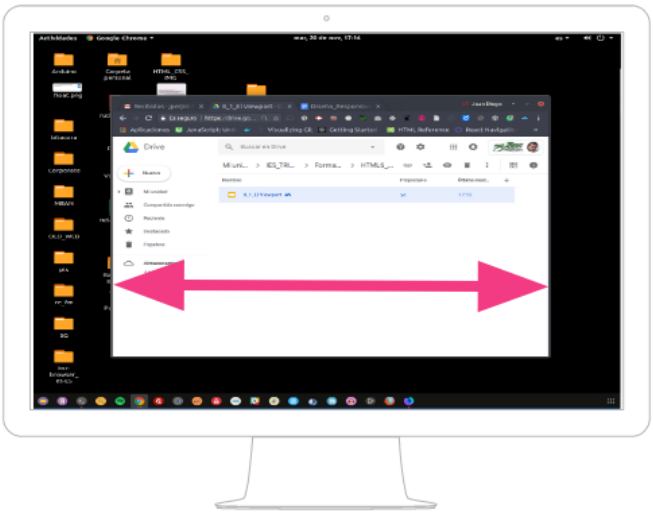
3. El viewport

Uno de los conceptos más importantes que debemos conocer antes de lanzarnos a realizar diseños responsivos es el concepto de ***Viewport***.

Una posible definición sería:

Área de la pantalla en la que el navegador puede renderizar contenido, es decir, el espacio disponible para mostrar mi interfaz web.

Este concepto es muy fácil de comprender cuando nos referimos a él en sistemas de escritorios o en portátiles. En estos casos el **Viewport** coincide con la pantalla de nuestro navegador.

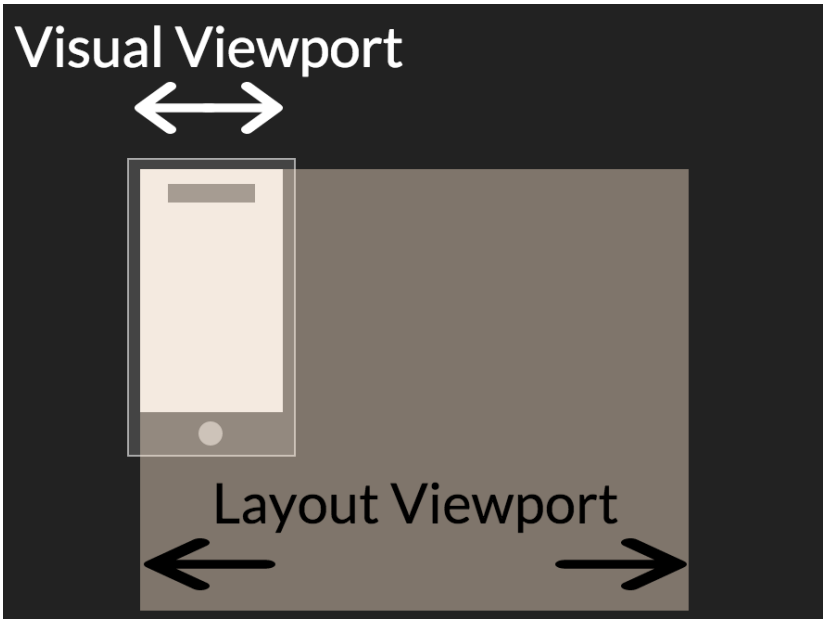


Sin embargo, si nos referimos a móviles y tablets estaremos hablando de otra cosa diferente.

Cuando este tipo de dispositivos irrumpieron, las páginas se maquetaban usando normalmente una anchura fija en píxels que solía variar entre 800px y 1000px que, obviamente, era mayor que la anchura de la pantalla de los móviles.

Ante esta circunstancia los fabricantes hicieron que este tipo de dispositivos tuvieran dos **Viewports**:

- El ***Layout-viewport*** que es el viewport que es tenido en cuenta para la aplicación de estilos. Suele ser de aproximadamente 900px.
- El ***Visual-viewport*** que es el viewport que realmente ve el usuario cuando está navegando.



Además, sucede que en este tipo de dispositivos podemos hacer Zoom mediante gestos lo que provoca que:

- No cambie el ***Layout-viewport***
- Cambie el ***Visual-viewport***.

Y no acaban ahí los problemas. También, en este mundo de miles de dispositivos diferentes, nos vamos a encontrar con que tenemos que lidiar con diferentes valores para medir la pantalla de los dispositivos:

- **Hardware Pixels:** (o píxeles físicos) son los puntos de luz reales que componen la pantalla de un dispositivo. Son las unidades más pequeñas que puede controlar la tarjeta gráfica.
- **DIP (Device Independent Pixels):** (o Pixeles independientes del dispositivo) son unidades de medida abstractas que se utilizan para mantener un tamaño consistente de los elementos en pantalla. Ocupan lo mismo independientemente de la densidad de pixels de la pantalla. Es una unidad virtualizada que puede ser igual, mayor o menor que un píxel físico.

  Por ejemplo, en el sistema operativo **Android** , un **DIP** equivale a un **píxel físico** en una pantalla de 160 dpi (puntos por pulgada), unos 0.15875 mm. Mientras que, en una pantalla estándar, un **DIP** equivale a 1/96 pulgadas.

  Otro ejemplo, un elemento con altura: 200px; ancho: 200px es siempre 200 por 200 en píxeles DIP, incluso si el hardware está configurado para mostrar una resolución más alta o más baja de lo habitual. Con un factor de escala del 200%, si un píxel DIP de 1x1 se asigna a un píxel físico de 1x1 al 100%, se asignará a un cuadrado de píxeles físicos de 2x2.
- **DPR (Device Pixel Ratio): Hw Pixel / DIP**. Indica cuántos píxeles físicos hay en cada dimensión *(tanto ancho como alto)* por cada píxel lógico. Así:
  - Una pantalla con DPR de 1 significa que 1 píxel lógico = 1 píxel físico.
  - Una pantalla con DPR de 2 significa que 1 píxel lógico = 4 píxeles físicos (2x2).
  - Una pantalla con DPR de 3 significa que 1 píxel lógico = 9 píxeles físicos (3x3).

  En desarrollo web, el DPR es crucial por varias razones:

  1. **Calidad de imágenes**: Ayuda a determinar qué versión de una imagen servir para que se vea nítida en pantallas de alta densidad.
  2. **Media Queries**: Permite adaptar el contenido según la densidad de píxeles usando @media (-webkit-device-pixel-ratio: 2)
  3. **JavaScript**: Se puede consultar usando window.devicePixelRatio
  4. **Optimización**: Permite balancear la calidad visual con el rendimiento y consumo de datos, especialmente importante en dispositivos móviles.

Esto, todo junto, es un jaleo y hay multitud de teoría detrás. Sin embargo, lo que necesitamos para empezar a gestionar toda esta situación es simplemente añadir una línea en la cabecera de mi página Web.

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

Con esa simple línea podremos empezar a maquetar diseños responsivos de manera eficiente e incluso haremos que las páginas que no están preparadas para móviles se muestren correctamente, aunque tengamos, eso sí, que hacer continuos ***zooms*** para poder usarlas.

Con esta etiqueta **meta**, estamos estableciendo unos parámetros de comportamiento para el **viewport**:

width	device-width	Indica un ancho para el viewport.
height	device-height	Indica un alto para el viewport.
initial-scale	1	Escala inicial con la que se visualiza la página web.
minimum-scale	0.1	Escala mínima a la que se puede reducir al hacer zoom.
maximum-scale	10	Escala máxima a la que se puede aumentar al hacer zoom.
user-scalable	no/fixed | **yes/zoom**	Posibilidad de hacer zoom en la página web.

Las propiedades **initial-scale**, **minimum-scale** y **maximum-scale** permiten valores desde el **0.1** al **10**, aunque ciertos valores se traducen automáticamente a ciertos números determinados:

- yes=1
- no=0.1
- device-width=10
- device-height=10

Por otra parte, **user-scalable** permite definir si es posible que el usuario pueda «pellizcar» la pantalla para ampliar o reducir el zoom.

4. Tipos de diseños (layouts) y breakpoints

En el apartado anterior hemos conocido el importante concepto del ***Viewport*** y ahora, antes de lanzarnos a hacer diseño responsivo, vamos a continuar introduciendo los diferentes tipos de diseños o layouts que podemos tener.

Hay clasificaciones más complejas, pero si condensamos podemos establecer que hay tres tipos de diseños o layouts a la hora de hacer maquetación web.

- **Fixed:** Donde la anchura de la página es fija y es expresada en pixels.
- **Elastic:** Donde la anchura de la página es fija y es expresada en la unidad **em** (múltiplos del tamaño de letra).
- **Fluid/Liquid/Relative:** Donde la anchura de la página depende del tamaño del Viewport del usuario y se expresa en porcentajes (%).

A continuación, vamos a explicar de manera concisa las ventajas y desventajas de cada uno de estos tipos.

4.1. Fixed Layout

Ventajas: Tenemos un control total sobre el layout ya que especificamos dimensiones fijas.

Desventaja: Puede aparecer un scroll horizontal en pantallas pequeñas (si no escalo). Así, por ejemplo, una imagen a la que se le da ancho y alto usando este layout, puede variar su tamaño cuando establezco un valor mayor que 1.0 en "*initial-scale"* en el viewport.

Sin embargo, este tipo de Layouts puede tener sentido si no quiero cambiar nunca el layout ni sus proporciones en ningún dispositivo (no suele ser el caso para las pantallas de los móviles).

4.2. Elastic Layout

Este layout usa la unidad relativa ***em***. Recordamos que ***em*** tiene como referencia de medida el *font-size* del elemento padre.

Ventajas: Control al hacer zoom-in y zoom-out. Se mantienen las proporciones.

Desventajas:

- Al final es igual que fixed.
- Es difícil calcular las dimensiones reales.
- Ems se calcula en relación al padre, podemos necesitar **rem** (***r***oot ***em***: tamaño de *font*-size del elemento <html>).
- No hay fluidez cuando cambia el tamaño de pantalla. No se adapta bien.

4.3. Fluid Layout

Ventaja: Se adapta al Viewport, al tamaño de lo que está viendo el usuario.

ES EL TIPO DE LAYOUT QUE SE USA PARA DISEÑO RESPONSIVO.

4.4. Breakpoints

Una vez hemos establecido que usaremos diseños de tipo ***Fluid*** debemos recordar que el concepto de diseño responsivo, además de ***adaptación*** al tamaño, implicaba el ***cambio* en el diseño** atendiendo al tamaño u otras características del dispositivo.

La anchura de la pantalla en la que se produce el cambio es lo que se conoce como ***Breakpoint***.

La elección de unos ***breakpoints*** adecuados no es una tarea fácil, pero tenemos la suerte de que ya ha habido empresas que han hecho estudios muy serios al respecto. Destacar los Breakpoints elegidos por **BootStrap**:

- <576px (pantallas pequeñas).
- 576px-768px (móviles apaisados).
- 768px-992px (tablets).
- 992px-120ppx (desktop).
- >1200px (pantallas grandes).

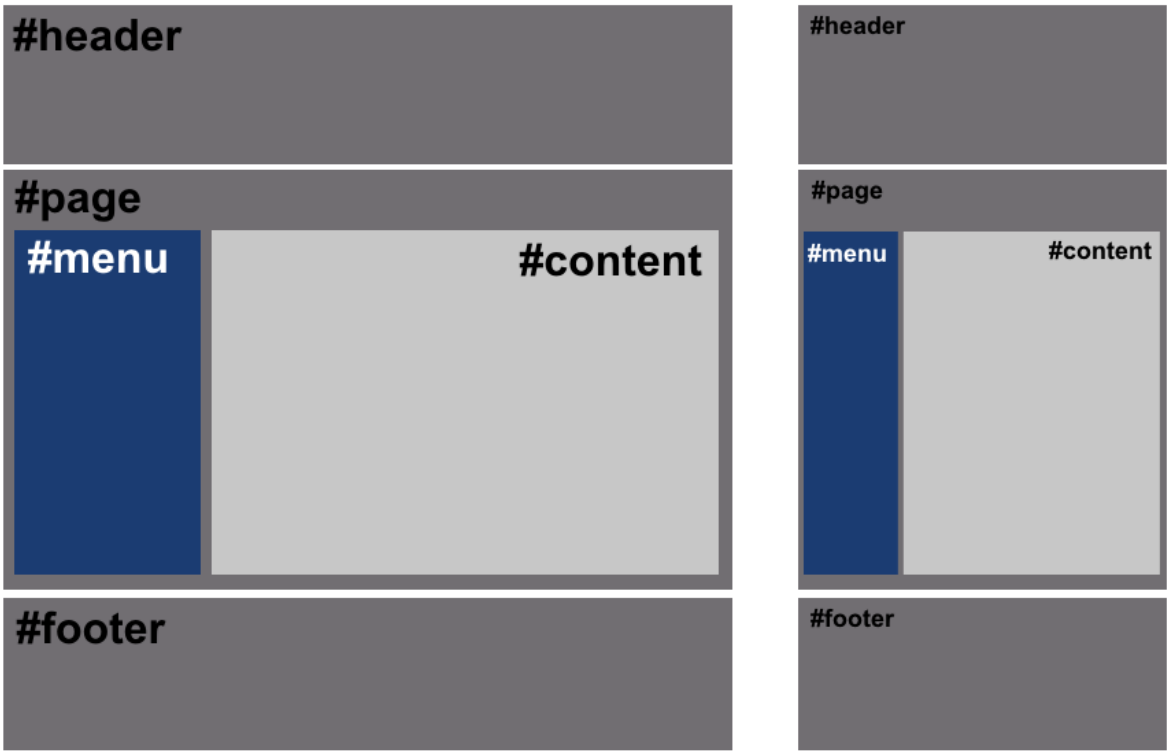
Los anteriores breakpoints están ya un poco anticuados. Actualmente, los breakpoints más comunes son:

- **Mobile** (Smartphones):
  - 320px - 480px: Móviles pequeños.
  - 481px - 767px: Móviles grandes.
- **Tablet**
  - 768px - 1023px: Tablets verticales y horizontales.
- **Desktop**
  - 1024px - 1279px: Laptops y desktop pequeños.
  - 1280px - 1439px: Desktop medianos.
  - >1440px: Desktop grandes y pantallas 4K.

5. Diseño con porcentajes

El primer paso para crear un diseño adaptativo correctamente, es comenzar a familiarizarse con un tipo de unidades: **los porcentajes**. Recordemos que los porcentajes son relativos al contenedor padre, por lo que, si especificamos un porcentaje a un elemento, el navegador va a tomar dicho porcentaje del contenedor.

Podemos comenzar usando porcentajes en las propiedades **width**. De esta forma, se adaptarán dependiendo del ancho de la pantalla o navegador. Por ejemplo, si establecemos un ancho de **100%** a los elementos grises (*#header, #page y #footer*), un **30%** al azul (*#menu*) y un **70%** al gris claro (*#content*), a los cuales se les indicará un **display:inline-block** para estar contiguos, podríamos obtener este diseño:



Sin embargo, utilizar porcentajes no nos garantiza un diseño adaptativo de calidad. El primer problema que encontraremos será que si sumamos el tamaño de los elementos (*70% + 30%*), no es inferior al 100% del contenedor padre, por lo que **no cabe en su interior**, así que el segundo elemento se desplaza a la zona inferior, descuadrando todo el diseño. Esto es algo muy habitual en CSS. Y frustrante.

Comprobaremos, sin embargo, que si reducimos alguno de los dos porcentajes interiores (*70% o 30%*), si conseguiremos el diseño propuesto. Luego, si intentamos aumentar su relleno (*padding*) o el tamaño del borde (*border-width*) veremos que se vuelve a **descuadrar**. Esto ocurre porque el tamaño en porcentaje no incluye rellenos o el tamaño del borde, salvo que se utilice la propiedad **box-sizing:border-box**.

Estar pendiente de realizar todos estos ajustes (*sobre todo en diseños más complejos*) no suele ser algo práctico, por lo que vamos a ver otra serie de propiedades que suelen facilitarnos un poco todo este desarrollo.

6. Tamaños máximos y mínimos

Si buscamos un grado de control aún mayor, podríamos recurrir a las propiedades **max-width** y **min-width**, con las que podemos indicar el ancho de un elemento como máximo y el ancho de un elemento como mínimo respectivamente, consiguiendo así garantizar cierto control del diseño.

```
div {
	max-width: 800px;
	min-width: 300px;
}
```

En este caso, el elemento tiene un tamaño máximo de 800 píxeles, y un tamaño mínimo de 300 píxeles, por lo que, si ajustamos el ancho de la ventana del navegador, dicho elemento ** ** iría variando en un rango de 300 a 800 píxeles, nunca haciéndose más pequeño de 300px o más grande de 800px.

Nota: **Es importante darse cuenta de que este ejemplo funciona porque no hay definido un width (por omisión, es igual a width:auto). Desde que exista un width, las otras propiedades pierden efecto.**

Con las imágenes, videos y contenidos multimedia, se puede hacer lo mismo, consiguiendo así que las imágenes se escalen y adapten al formato especificado o incluso al tamaño de pantalla de los diferentes dispositivos utilizados:

```
img, video, object, embed {
	max-width: 100%; /* Aquí el tamaño máximo de las imágenes */
	height: auto; /* No es necesario, es solo para clarificar */
}
```

7. Media queries

Una de las herramientas esenciales para poder realizar un diseño responsivo de calidad son las ***media-queries*** que son módulos *de CSS3 que nos permiten adaptar la representación del contenido a las características del dispositivo.*

7.1. Sintaxis de las Media-Queries

Las media queries son expresiones en las que indicamos un tipo de medio (impresora, pantalla, proyector, …) y una consulta en relación a las características del dispositivo como alto, ancho e incluso color:

Algo similar a lo siguiente:

```
@media mediatype [condiciones] {
  .... /* Estilos para esas condiciones */ ....;;
}
```

Los distintos tipos de valores que puedo tener para ***mediatype*** son:

- all
- **screen ← Más usado**
- ~~tty (terminal)~~
- print (para impresoras)
- ~~tv (para pantallas de televisión)~~
- ~~projection (para proyectar contenidos)~~
- speech (para sintetizadores de voz, accesibilidad)
- etc…

En cuanto a las condiciones que puedo consultar:

- **width | min-width | max-width ← Más usadas.** Con min-width podremos especificar los dispositivos con ancho de pantalla mínimo o mayor que el asignado a min-width. De la misma forma, con max-width podremos especificar el valor máximo o menor que el asignado a max-width.

- **height | min-height | max-height**

- **device-width | min-device-width | max-device-width** → describe la anchura del dispositivo de salida (ya sea la totalidad de la pantalla o página y no el área del documento a renderizar). Idem para **device-height |min-device-height | max-device-height**. La principal diferencia entre **width** y **device-width** es que el ancho del dispositivo no siempre coincide con la ventana gráfica de diseño de dicho dispositivo. Muchas tabletas y dispositivos móviles no siempre tienen 1 píxel de dispositivo por píxel CSS.

- **orientation** (landscape / portrait) → pantalla apaisada (hay más ancho que alto) o no.

- **aspect-ratio | min-aspect-ratio | max-aspect-ratio** → ratios de aspecto de pantalla. Algunos de los valores típicos son: **4/3, 1/1, 16/9, 3/2**, etc. **Aquí** tienes una guía de cómo conseguir esos aspectos.

- **color | min-color | max-color** → profundidad de color expresada en bits (8, 16, 32, 64, …).

- **Otros**: resolution, scan, grid, update-frequency, overflow-block, pointer, hover, scripting, …

Estas condiciones se pueden combinar y modificar utilizando clausulas como:

- And. Concatena condiciones. Para que la media querie se ejecute, deben cumplirse todas las condiciones.
- Not. Niega condiciones.
- Only. El operador only se usa principalmente para prevenir que navegadores antiguos apliquen los estilos. Ejemplo:

```
@media only screen and (max-width: 900px) {…}
```

- ¿Or? Podemos hacer comparaciones más complejas usando varios ***and*** encadenados, pero ¿puedo crear una condición con ***and*** y ***or simultáneamente***? Pues sí, podemos encadenar condiciones ***and*** y pondremos **or** usando **“,” (coma)**, por ejemplo:

```
/*  width entre 600px y 900px O por encima de 1100px: */
@media screen
and (max-width: 900px)
and (min-width: 600px), (min-width: 1100px) {…}
```

Si juntamos todo podemos ver algunos ejemplos:

```
/* Estilos para todo tipo de pantallas con una anchura de 576px o menor*/
@media all and (max-width: 576px) {
  .....;
}


/* Estilos para pantallas con al menos 992px de anchura y que estén apaisadas (más ancho que alto)*/
@media screen and (min-width: 992px) and (orientation: landscape) {
  ......;
}


/* Estilos sólo para pantallas que tengan al menos 768px de anchura*/
@media only screen and (min-width: 768px) {
  ......;
}
```

7.2. Hojas de estilos diferentes

Podemos usar media-queries en la etiqueta *link* para seleccionar hojas de estilos diferentes atendiendo a las características del dispositivo.

Por ejemplo:

```
<!--Para pantallas de hasta 576px de ancho -->
```

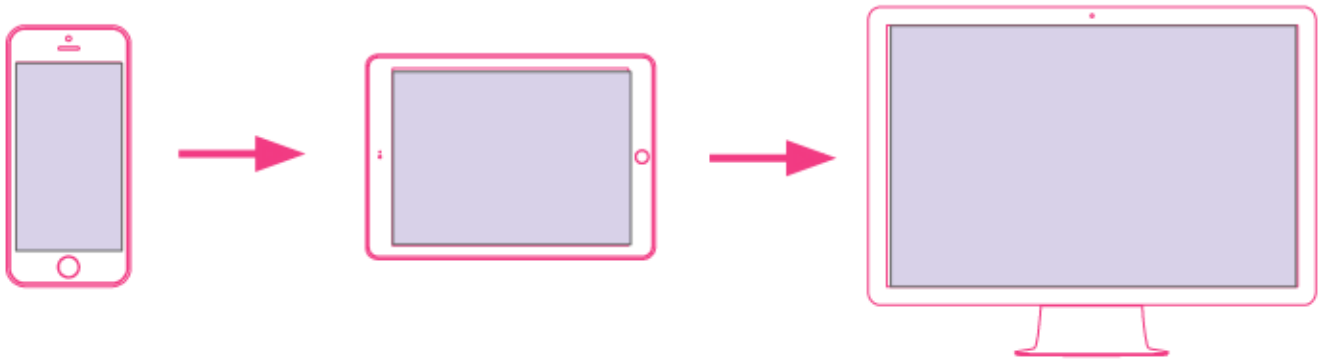
```
<link rel="stylesheet" media="(max-width: 576px)" href="small.css" />

<!--Para pantallas entre 576px y 768px de ancho -->
<link rel="stylesheet" media="(max-width: 768px)" href="medium.css" />

<!--Para pantallas entre 768px y 992pxde ancho -->
<link rel="stylesheet" media="(max-width: 992px)" href="large.css" />
```

8. Patrones responsive

Afrontar un proyecto que requiera el desarrollo de una página web responsiva no es fácil y hay varias estrategias posibles para afrontarlo. No obstante, ya es comúnmente aceptado que lo más recomendable es lo siguiente:



Es decir, que el proceso de diseño de nuestra página web debe empezar haciendo que todo quede correctamente en pantallas pequeñas (**Mobile first**). De esta manera:

- **Priorizo** siempre lo que es importante. Si el proceso fuera al revés es muy fácil que caigamos en el error de quitar cosas que sean realmente importantes.
- Me pregunto en cada diseño si es **necesario un diseño nuevo** para pantallas más grandes.
- Elijo los **breakpoints** más adecuados.

En nuestro proceso de diseñar interfaces responsivas debemos conocer lo que se conoce como ***patrones responsivos***.

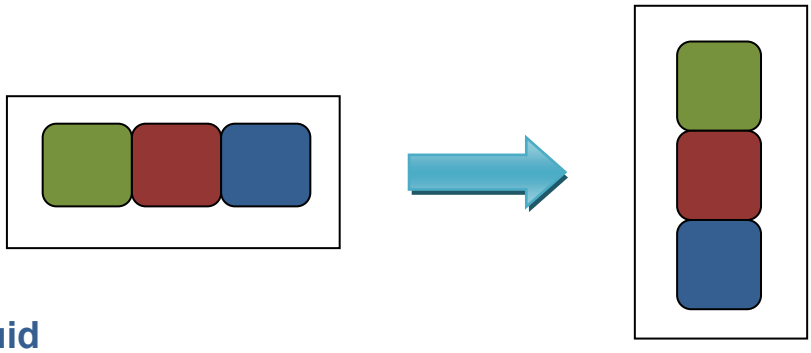
Los ***patrones responsivos*** son soluciones que se han dado por buenas para el problema de diseñar páginas web responsivas. Hay muchos, pero los más comunes son los siguientes:

- Column Drop
- Mostly Fluid
- Layout Shifter
- Off Canvas
- …
- Mezclar varios de ellos
- Pequeños ajustes (Tiny Tweaks)

A continuación, pasamos a comentar los principales:

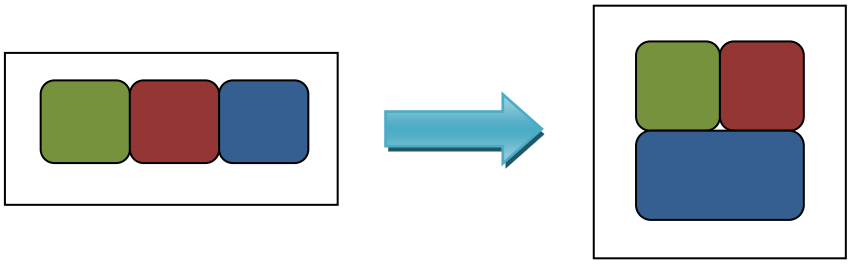
8.1. Column Drop

Es el patrón más básico y común, y consiste en que, en cada breakpoint, se va apilando (uno debajo de otro) un elemento. Así, en la pantalla más pequeña tendré todos los elementos apilados.



8.2. Mostly Fluid

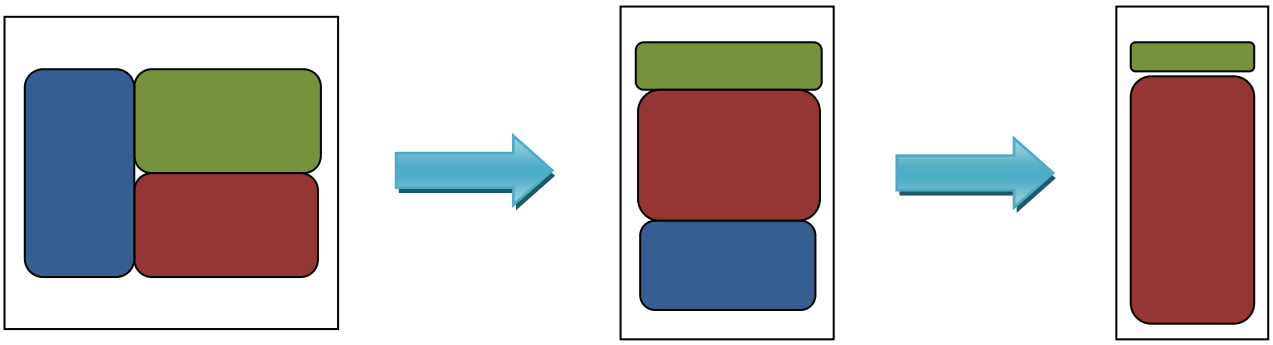
Parecido a Column Drop. Es una cuadrícula fluida y en cada breakpoint hay redimensionamiento de una o varias columnas.



8.3. Layout Shifter

Es el patrón más responsivo, en cada breakpoint, **y suele tener varios**, cambia el diseño del layout, no únicamente el flujo y la anchura de los elementos. Hay redimensionamiento y nuevos posicionamientos de los elementos.

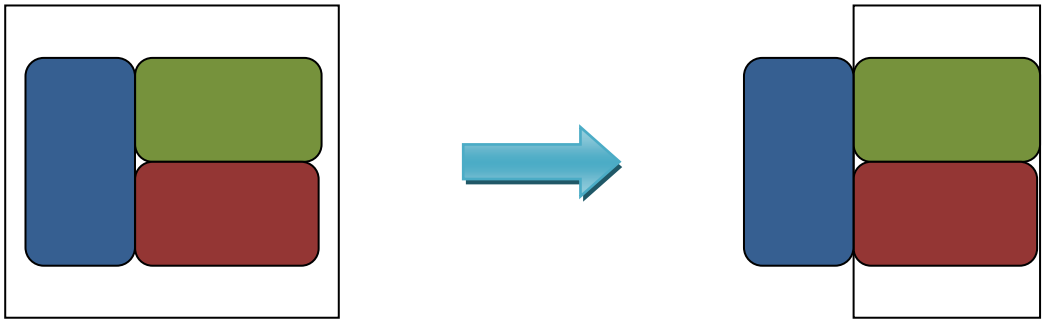
La clave para este diseño es el desplazamiento del contenido, en lugar de su reprocesamiento y colocación debajo de otras columnas. Debido a las diferencias significativas entre cada punto de interrupción principal, es más complejo de mantener, y es posible que se deban realizar cambios dentro de los elementos, no solo en el diseño de contenido general.



8.4. Off Canvas

En vez de apilar contenidos, éstos se colocan fuera de la pantalla cuando el tamaño de pantalla no es lo suficientemente grande.

En lugar de apilar contenido verticalmente, el patrón Off canvas coloca contenido menos usado (tal vez menús de navegación o de apps) fuera de la pantalla y solo lo muestra cuando el tamaño de la pantalla es suficientemente grande. En las pantallas más pequeñas, el acceso al contenido es posible con solo dar a un clic.



8.5. Tiny tweaks

El patrón Tiny tweaks permite realizar pequeños cambios en el diseño, como ajustar el tamaño de la fuente, cambiar el tamaño de las imágenes o desplazar el contenido de maneras muy poco significativas.

Funciona correctamente en diseños con una sola columna, como los sitios web lineales de una sola página y los artículos con mucho texto.

9. Imágenes responsivas

Las imágenes son un elemento **fundamental** de todas las páginas y representan una gran parte del **peso (más o menos el 60%)** o tamaño en MBs de la misma.

El diseño web adaptable significa que no solo nuestros diseños pueden cambiar según las características del dispositivo: también lo puede hacer el contenido. Por ejemplo, en pantallas de alta resolución (2x), los gráficos de alta resolución garantizan nitidez. Una imagen con un ancho del 50% puede funcionar bien cuando el navegador tiene 800 px de ancho, pero en un teléfono ocupará mucho espacio en pantalla y **requerirá el mismo ancho de banda** cuando se reduzca para entrar en una pantalla más pequeña.

Esta situación plantea ciertos **retos** a la hora de hacer un diseño responsivo**.**

Estos retos son principalmente dos:

- Retos a nivel de **optimización** de la página web.
- Retos a la hora de **diseñar** la página.

9.1. Optimización de Imágenes en el Diseño Responsivo

Cuando nos referimos a optimizar el uso de las imágenes nos referimos a:

- Consumir el menor ancho de banda posible.

- Elegir la versión de una misma imagen más adecuada para la resolución.

En este proceso de optimización debemos tener en cuenta:

- Ancho del dispositivo.
- Dimensiones de la imagen.
- Resolución de la imagen.

Optimización. Imágenes SVG

Para solucionar este tipo de problemas lo más fácil es usar imágenes **SVG** que son gráficos vectoriales que escalan y encogen sin perder resolución.

El problema es que no siempre disponemos de gráficos SVG.

Optimización. Imágenes PNG/GIF/JPEG

En estos casos, si no nos importa la optimización, es suficiente con usar una imagen de gran resolución y dimensiones y acotarla dentro de un contenedor.

Por ejemplo:

```
<div><img src="......." /></div>
```

```
div {
  /* dimensiones deseadas */
}

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

Pero, sin embargo, si nos importa la optimización utilizaremos los atributos ***scrset*** y/o ***sizes*** de la imagen que queremos mostrar.

- **srcset**: podemos establecer la imagen a mostrar **según la resolución ("x": densidad de bits o número de pixels de la pantalla física por cada pixel de tamaño CSS)** o **dimensiones de la pantalla ("w": ancho de la ventana)**. Permite que el navegador elija la mejor imagen según las características del dispositivo; por ejemplo, el uso de una imagen de 2x en una pantalla de 2x, o una imagen de 1x en un dispositivo de 2x en una red con ancho de banda limitado.
- **sizes**: es un equivalente a media queries. Podemos indicar la imagen que debe elegir el navegador según el tamaño del viewport. Su sintaxis es la siguiente:

  ```
  sizes="(condición-media) tamaño, tamaño-por-defecto"
  ```

  Ejemplo:

```
<img srcset="small.jpg 300w,
        medium.jpg 600w,
        large.jpg 900w"
 sizes="(max-width: 500px) 100vw,
       (max-width: 900px) 50vw,
       33vw"
 src="fallback.jpg"
 alt="Imagen">
```

Esto significa:

- En pantallas < 500px: imagen ocupa 100% del viewport.
- Entre 500-900px: ocupa 50% del viewport.
- > 900px: ocupa 33% del viewport.

El navegador usa esta información junto con srcset para seleccionar la imagen más apropiada.

En navegadores que no son compatibles con srcset, el navegador simplemente usa el archivo de imagen predeterminado especificado por el atributo src. Por esta razón, siempre es importante incluir una imagen de 1x que se pueda mostrar en cualquier dispositivo, independientemente de las capacidades. Cuando se admite srcset, la lista de imágenes o condiciones separadas por comas se analiza antes de realizar cualquier solicitud, y solo se descarga y se muestra la imagen más apropiada.

Por ejemplo:

```
<!—Considerando resolución: →
<div class=”container”>
  <!—Se muestra la imagen big.jpg para pantallas de doble densidad  y small.jpg para el resto de pantallas…-->
  <img src=”img/small.jpg” srcset=”img/big.jpg 2x, img/small.jpg 1x” />
</div>
```

El atributo **srcset** permite usar imágenes alternativas en función de la pantalla que el usuario esté usando. Simplemente le indicamos una url de imagen y el **tamaño real de la imagen**.

```
<img srcset=”
        /imgs/foto3x.jpg 996w,
        /imgs/foto2x.jpg 667w,
        /imgs/foto1x.jpg 400w “
src=”/imgs/foto.jpg” alt=”imagen adaptativa”>
```

¿Cómo se interpreta esto? Fácil, foto1x.jpg es la imagen más pequeña con 400 pixels de ancho, foto2x.jpg es de tamaño medio 667pixels, foto3x.jpg tiene un tamaño mayor, 996 pixels. Pero ojo, **la**

unidad usada es w no los habituales pixels. Esta unidad se refiere a los pixels de definición del visor, es decir, el ancho de la ventana. El navegador adaptará el tamaño mostrado al tamaño de la pantalla.

Por lo tanto, si el viewport tiene un ancho menor de 400px, se mostrará la imagen foto1x.jpg. Con un viewport con ancho menor de 667px, se mostrará la imagen foto2x.jpg y, para pantallas con ancho menor de 996px, se mostrará foto3x.jpg.

El navegador elegirá automáticamente la imagen más apropiada basándose en:

- El ancho de la ventana del navegador.
- La densidad de píxeles del dispositivo.
- Las condiciones de red.

Otro ejemplo:

```
<!—Considerando dimensiones: →
<div class=”container”>
  <img
    src=”img/small.jpg”
    <!—Aquí se usa una imagen u otra según el tamaño en “W”, “W” es una unidad especial que representa el tamaño real del dispositivo  →
    srcset=”img/big.jpg 2000w, img/small.jpg 1000w”
    <!—Para pantallas grandes, la imagen ocupa el 100% del viewport →
    sizes=”(min-width: 960px) 960px,100vw”
  />
</div>
```

NOTA: Solo cuando usamos el ancho “w” para *srcset*, podremos usar el atributo “*sizes*”.

Las medidas vh y vw son medidas relativas de acuerdo al viewport. vh hace referencia a la centésima parte de la altura del viewport y vw a la centésima parte del ancho del viewport.

- 1vh = 1% de la altura del viewport
- 100vh = altura del viewport
- 1vw = 1% del ancho del viewport
- 100vw = ancho del viewport

Otro ejemplo:

```
<img srcset="
        /imgs/foto3x.jpg 996w,
        /imgs/foto2x.jpg 690w,
        /imgs/foto1x.jpg 400w "
sizes = "
        (max-width: 400px) 400px,
        (max-width: 720px) 600px,
```

```
        (min-width: 721px) 1024px"
```

```
src = "foto.jpg" alt="imagen adaptativa">
```

Para cada imagen posible existe un atributo **size** con dos partes:

1. Una **condición** entre paréntesis que comprueba el tamaño de ventana.
2. La **anchura** disponible que se usará para la imagen.

Podemos poner en la condición de size las mismas opciones que con media querys, por ejemplo:

- **(min-width: 900px)**
- **(not (orientation: landscape))**
- **(orientation: landscape) and (min-width: 900px)**
- **((orientation: portrait), (max-width: 500px))**

Nota: Si usamos el ancho (width) en la condición, podemos usar casi cualquier valor de longitud, p. Ej. em, rem, píxeles y ancho de la ventana gráfica. Sin embargo, los valores % no están permitidos, "para evitar confusiones sobre a qué serían relativos". El valor de vw se recomienda como alternativa si se necesita un valor relativo.

Al usar los atributos **srcset** y **sizes** el navegador actúa como sigue:

1. **Comprueba** el tamaño de la ventana.
2. Verifica la **primera condición** de medios que se cumple.
3. Determina el **ancho disponible** para la imagen.
4. Elige la imagen con tamaño más próximo de las declaradas en **srcset.**

9.2. Diseño *"ART-DIRECTOR"*

La técnica de diseño responsivo **Art Director** consiste en elegir una u otra imagen utilizando la etiqueta **source** dentro la etiqueta **picture** y sus atributos **srcset** para indicar la imagen y **media,** que funciona de manera similar a una media query.

Se ve muy claro con un ejemplo:

```
<div class="art">
  <picture>
    <source media="(min-width: 576px)" srcset="img/big-art.jpg" />
    <source media="(max-width: 575px)" srcset="img/small-art.jpg" />
    <img src="img/small-art.jpg" />    <!-- En caso de no soportar picture -->
  </picture>
</div>
```

Normalmente este diseño consiste en fotos que se acercan al objeto importante de la misma a medida que la pantalla es menor. Es como si se hiciera un zoom, de ahí el nombre de ART DIRECTOR.

Al final, en casos reales siempre hay que combinar ambas técnicas y probar, en la medida de lo posible, en dispositivos reales, incluidos móviles con Retina Display (doble densidad de pixels).

10. Tablas responsivas

Las tablas son un elemento problemático en el diseño responsive, ya que en cuanto tienen más de unas pocas columnas van a provocar un scroll horizontal en pantallas pequeñas.

Posibles soluciones:

- Esconder columnas.
- Convertir filas a listas.
- Scroll controlado.

No hay una solución universal. Debemos experimentar según el Layout que tengamos.

10.1. Esconder columnas

Ventajas: Diseño responsivo y priorizo el contenido.
Desventajas: Pierdo información.

```
@media screen and (max-width:576px) {
.hidden td.primero,
.hidden th.primero {
  display:none;
 }
}
```

```
@media screen and (max-width:768px) {
.hidden td.segundo,
.hidden th.segundo {
  display:none;
 }
}
```

10.2. Convertir filas a listas

La tabla pasa a convertirse en una lista vertical. Esto se puede conseguir asignando a ***todos*** los elementos de una tabla (thead, tbody, tr, td, th, …) la propiedad **display:block**.

Ventajas: diseño responsivo y no pierdo información.
Desventajas: no priorizo la información y "desplazo" hacia abajo el resto del layout.

10.3. Scroll controlado

Aparecerá scroll horizontal. Esto se puede conseguir con la propiedad CSS **overflow-x: auto** al elemento contenedor de la tabla.

Ventajas: diseño responsivo y no pierdo información.
Desventajas: no priorizo la información y aparece scroll horizontal.

11. Texto responsivo

Cuando hablamos de diseño responsivo solemos dar más importancia al layout, pero el texto que vamos a presentar es igual de importante para conseguir un buen diseño.

Si no tenemos cuidado nos podemos encontrar con varios problemas:

- Líneas cortas con pocos caracteres, lo que dificultan la lectura.
- Líneas largas con muchos caracteres, lo que también dificulta la lectura.
- Caracteres muy pequeños.

Lo ideal, según estudios, es una letra de un tamaño adecuado, para poder leer sin forzar la vista, y que se disponga formando líneas de entre 60-80 caracteres.

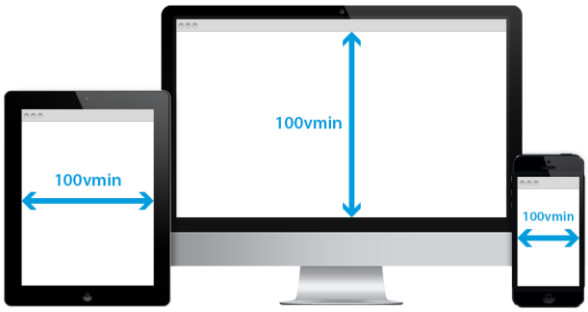
Para intentar solucionar esto con garantías lo más recomendable es utilizar para el texto unidades de tamaño relativas al viewport. Tendremos:

- **vw** en relación a la anchura del viewport.

- **vh** en relación a la altura del viewport.
- **vmin** el valor menor en relación a la dimensión pequeña del viewport (anchura o altura).
- **vmax** el valor máximo en relación a la dimensión más grande del viewport (anchura o altura).

Teniendo en cuenta, por ejemplificar, que 1vw es un 1% de la anchura del viewport.

En el caso de las unidades **vmin** y **vmax**, **obtenemos el valor porcentual mínimo o máximo de la ventana gráfica**, ya sea en **altura o anchura**. Es decir, que, dependiendo de la orientación del dispositivo, obtenemos el valor en porcentaje de altura o de anchura, el que menos o más mida en ese momento. Si nos imaginamos un *smartphone* en posición vertical (vista *portrait*), **vmin sería la medida porcentual en anchura** de la ventana gráfica, mientras que **vmax sería la medida porcentual en altura** de la ventana gráfica. Si por el contrario giramos el dispositivo y lo colocamos en posición horizontal (vista *landscape*), *vmin* sería la medida porcentual en altura, mientras que *vmax* sería la medida porcentual en anchura.



No obstante encontrar siempre el tamaño perfecto de texto es algo que puede ser complicado sino imposible. Sin embargo, aquí van unas posibles pautas:

- Calcular el tamaño mínimo y máximo que me puedo permitir usando ***calc()** (función css)
- No importa si el texto hace **"reflow"** (pasa a la siguiente línea) en determinadas ocasiones.
- Mantener el tamaño perfecto de línea puede que sea imposible.
- Si tenemos que elegir, es conveniente **priorizar móviles y tablets**.
- En algunos elementos, por ejemplo, en un menú de navegación, puede tener sentido usar **tamaño fijo de letra en vez de unidades relativas al viewport**.

Nota: **calc() → función css que realiza cálculos. Requiere una expresión matemática. El resultado se utilizará como valor. Se pueden utilizar los siguientes operadores: + - * /. Véase el siguiente ejemplo:**

```
#div1 {
  position: absolute;
  left: 50px;
  text-align: center;
  width: calc(100% - 100px); /*Dejará un ancho del 100% menos 50px por cada lado*/
}
```