Bootcamp Full Stack

CSS avanzado

Viewport y Breakpoints

Viewport

El viewport es el área visible de nuestro navegador. Podemos manipular cómo se ve, súper importante dada la variedad de dispositivos disponibles actualmente.

El viewport nos permite configurarlo de tal manera que pueda **ajustarse dinámicamente al tamaño de cada dispositivo usando el atributo 'device-width'** que es equivalente al **100% del ancho de la pantalla** de dicho dispositivo, independientemente de su tamaño, posición o resolución. La configuración básica del **viewport** es:

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

Este elemento es sumamente importante: aún realizando los estilos correctos, **si olvidamos la etiqueta viewport, nuestro navegador no tomará el cambio de tamaño de pantalla,** por lo que ningún estilo será tomado y la interfaz será siempre la misma para cualquier tamaño.

Notaremos que el **viewport** adquiere más o menos atributos, pero los más importantes son:

- **name**: **viewport**, o área visible.
- **content**: **width**, en este punto podemos trabajar con un ancho específico o jugar entre rangos, pero lo más usual es encontrar el valor **device-width**, que toma el tamaño del dispositivo a través del cual accede el usuario.
- **Initial-scale**: la escala inicial del documento.
- **User-scalable**: permite indicar si el usuario puede o no hacer zoom.

Escalas en Viewport

Se puede trabajar con **diferentes escalas en viewport**, estas propiedades **afectan a la escala inicial y al ancho**, además de l**imitar los cambios a nivel de zoom**.

```
<meta name="viewport" content="width=device-width; initial-scale=2">

<meta name="viewport" content="width=device-width; maximum-scale=2">

<meta name="viewport" content="width=device-width; user-scalable=no">

<meta name="viewport" content="width=device-width; initial-scale=1", maximum-scale=1">
```

Variantes de Viewport

Existen medidas vinculadas al `viewport`.

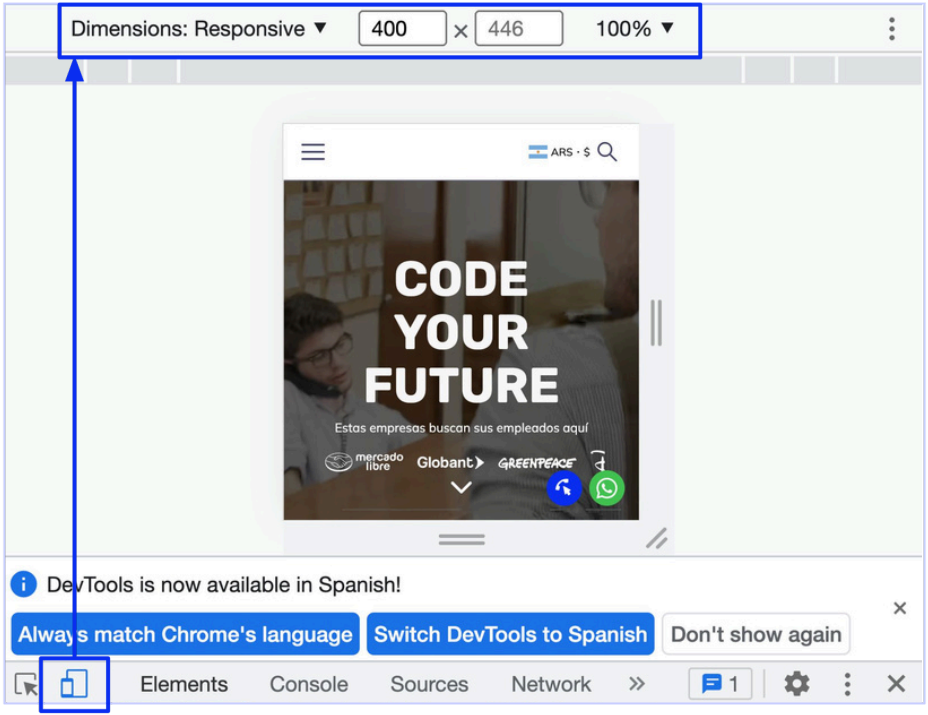
`vw`	Relative to 1% of the width of the viewport
`vh`	Relative to 1% of the height of the viewport
`vmin`	Relative to 1% of viewport's smaller dimension
`vmax`	Relative to 1% of viewport's larger dimension

Para poder analizar con mayor detalle existen **herramientas de visualización**:

- Responsinator
- ResizeMe.
- Firesizer.

Inspector de elementos

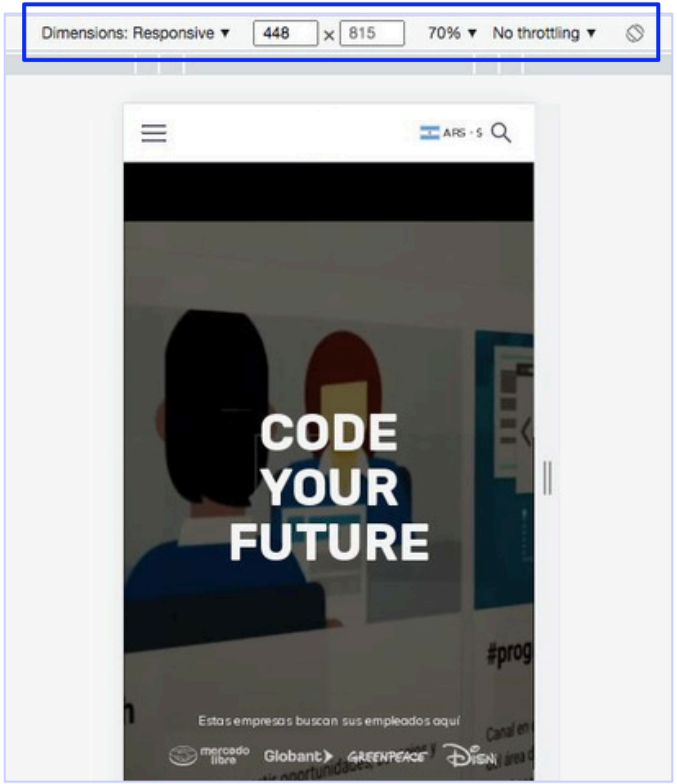
Otra opción, más simple, es usar el **inspector de elementos** de tu navegador (activar con **F12** o con botón derecho del mouse opción ***Inspector de elementos, Inspeccionar,*** o similares, dependiendo del navegador usado), que ofrece la posibilidad de **trabajar y visualizar nuestra interfaz en diferentes tamaños de pantalla** incluso con una lista de sugerencias con los posibles **breakpoints** generales a tener en cuenta.



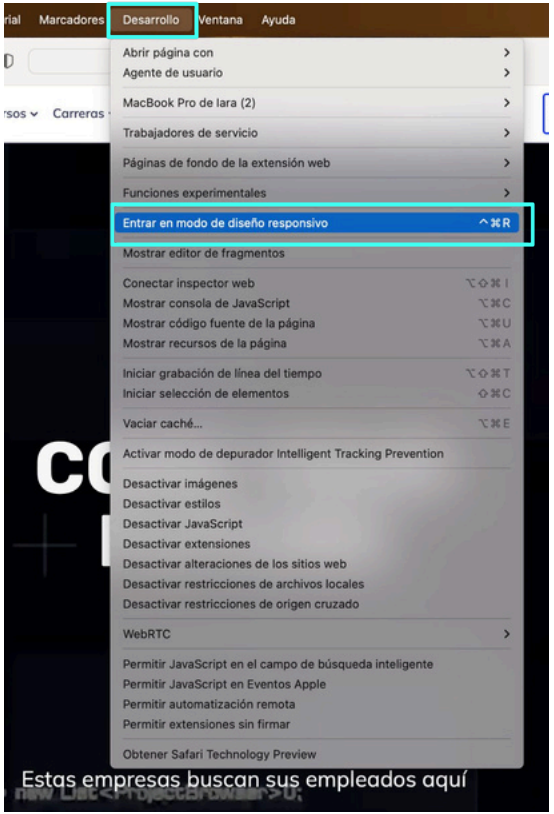
Ejemplo del Inspector de elementos de Chrome.

Así, nos permite trabajar con diferentes **viewport** y analizar si nuestros estilos están o no actuando correctamente de forma más segura.

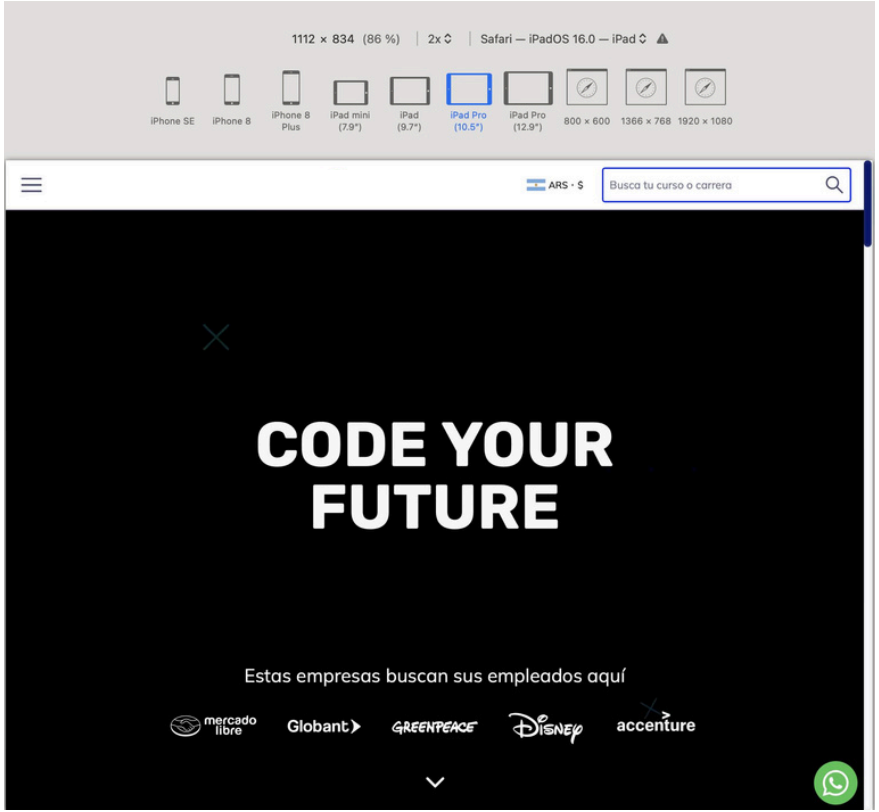
> Ejemplo del Inspector de elementos de Chrome.



Si, en cambio, utilizas Safari, deberás presionar **Ctrl+Command+R** o ir a ***Desarrollo > Entrar en modo de diseño responsivo,*** y se mostrará en la pantalla opciones para modificar las dimensiones de forma manual y botones con posibles **breakpoints** generales para acceder a ellos rápidamente, como se muestra en la siguiente.



Ejemplo de las herramientas de Desarrollo de Safari.



Ejemplo del modo de diseño responsivo de Safari

Breakpoints

Breakpoints son bloques de diseño responsivo. Es importante su utilización para controlar o adaptar la interfaz a un tamaño de pantalla específico.

Existen diversos breakpoints, pero usualmente **se recomienda maquetar para 3 tamaños específicos:**

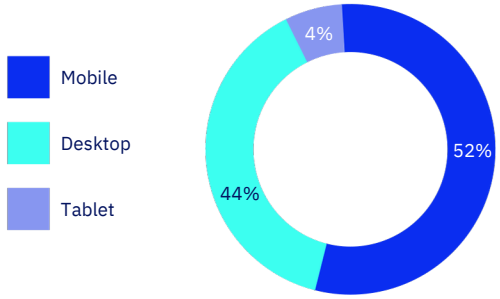
- 768px.
- 992px.
- 1200px.

Sin embargo, como mencionamos antes, existen diferentes tamaños de pantalla y dispositivos. Se hacen constantemente estudios para entender esa accesibilidad variada y el resultado está en el obvio proceso de trabajo responsivo puesto que es imposible generar una sola versión de nuestra interfaz para poder adaptarla a todos estos diferentes accesos.

Testeo constante

Nuestro testeo debe ser constante. Por ejemplo, no hace mucho en los navegadores se podía subir un video con audio y autoplay. Sin embargo, recientemente, primero Google Chrome y luego otros navegadores, **restringieron el autoplay a videos con audio**, por tanto para poder reproducirlos **debemos trabajar con muted** como característica ya preseteada para lograr que nuestro video se trabaje en reproducción automática.

El ejemplo anterior es solo uno entre tantos, del mismo modo que el testeo constante en el diseño responsivo, por lo tanto **siempre debemos estar al tanto de las novedades tecnológicas.**

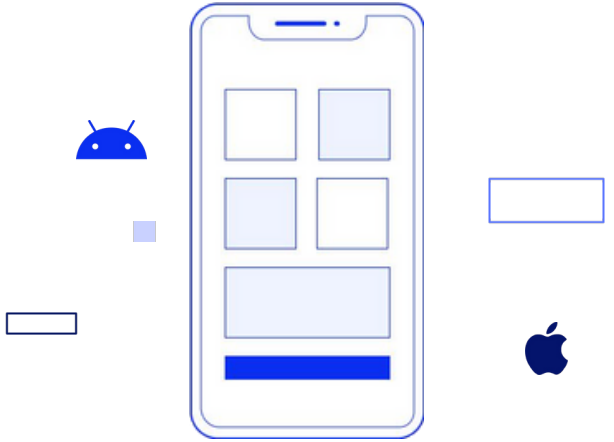


Traffic distribution by device type (Q3 2019)
Fuente: BrowserStack

Aplicaciones, fragmentación

Si bien no es parte de este curso, es importante entender que como maquetadores podemos trabajar en aplicaciones que también deben considerar la fragmentación para diferentes dispositivos.

Sin embargo, también en este trabajo se debe dar la característica colaborativa, de diseñador, maquetador, desarrollador tanto Back como Front, o en el caso de las aplicaciones el trabajo conjunto con un desarrollador Android o iOS, dependiendo el caso.



Regla @media

@media

Para poder trabajar con **breakpoints** desde CSS, debemos trabajar con la regla **@media**. Ésta nos permite **marcar los “cortes” en la interfaz** y avanzar sobre los estilos correctos al momento de adaptar nuestra interfaz.

Si bien puede trabajar con diversas variantes, generalmente se emplean las propiedades **min-width** y **max-width**. Estos rangos establecen los diferentes **breakpoints** hacia cuales irán orientados nuestros estilos.

min-width & max-width

Veamos algunos ejemplos típicos a establecer.

```
@media (max-width: 992px){
    body { background-color: blue;}

}


@media (max-width:768px){
    section { width: 100%; display: flex; flex-direction: column;}
    main { flex-direction: column;}
    body { background-color: yellow;}
}
```

En el segundo ejemplo del slide anterior, ordenamos al navegador que cuando la pantalla sea menor a 768px (inclusive) se modificará el **flex** de los elementos **section** pasándose a **column**. También modificamos la **direction** del **main** y el cambio de color del fondo a amarillo.

Así, conociendo los **breakpoints** y sus particularidades, vamos a destinar estilos específicos para cada uno.

En el ejemplo a la derecha, trabajando con **min-width** establecemos ciertos parámetros para cuando la pantalla sea mayor o igual a 993px, por ejemplo, cambiando el color de fondo, esta vez, a rojo.

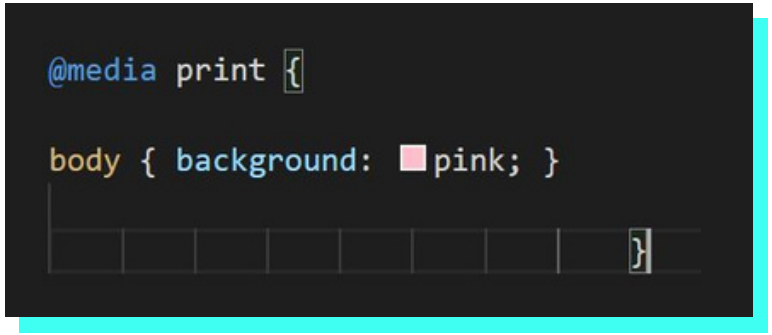
```
@media (min-width: 993px){

    body { background-color: red;}
    section { width: 50%; display: flex;  }

}
```

Otras variantes de @media

También podemos destinar nuestros estilos a medios específicos, por ejemplo, la impresión. Estos estilos podemos confirmarlos cuando vemos la vista previa de impresión donde podemos observar si se están llevando a cabo o no estás indicaciones.



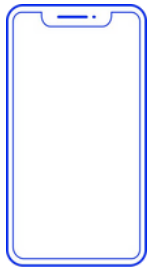
Variantes avanzadas de @media

Podemos realizar una regla más compleja, por ejemplo, incluyendo **varios medios o un rango de medida.**

Es importante recordar que **el último estilo es el que prevalece,** por ello debemos **asegurarnos de que no haya estilos que se sobreescriben** o pisen estilos anteriores por encontrarse dentro del mismo rango o medio.

```
@media print {
    body { font-size: 10cm }
}
@media screen {
    body { font-size: 15px }
}
@media screen, print {
    body { line-height: 1.2em }
}
@media only screen
    and (min-device-width: 576px)
    and (max-device-width: 768px)
{
    body { line-height: 1.4 }
}
```

Un uso muy común utilizado desde el uso masivo de dispositivos móviles es la **orientación de nuestra pantalla.** Para eso utilizamos **landscape** o **`portrait`**:



Portrait

El alto es mayor o igual al ancho.

Landscape

El ancho es mayor al alto.

```css
@media (orientation: landscape) {
  body {
    flex-direction: row;
  }
}

@media (orientation: portrait) {
  body {
    flex-direction: column;
  }
}
```

Maquetación responsiva: medidas de longitud

Trabajando con em

Las unidades **em** no han sido creadas por CSS, sino que llevan décadas utilizándose en el campo de la tipografía. Aunque no es una definición exacta, la unidad **1em equivale a la anchura de la letra M (eme mayúscula) del tipo y tamaño de letra del elemento**.

La unidad em hace referencia al tamaño en puntos (pt) de la fuente que se está utilizando.

Si se utiliza una tipografía de **12 pt**, **1 em** equivale a `12 puntos`.

PX

vs

Para tamaños y espacios fijos.

EM

vs

Depende de su propio tamaño, del tamaño de su contenedor padre y de los tamaños de las etiquetas `body` y `html`.

REM

Depende del tamaño de la etiqueta html. Es ideal para trabajar con ***padding*** y ***margin*** para que no modifique proporcionalmente el tamaño de texto, pero que, de ser necesario, sea escalable.

Como se trata de una **unidad de medida relativa,** es necesario realizar un cálculo matemático para determinar el tamaño en **em**.

La unidad de medida em siempre hace referencia al tamaño tipográfico del elemento padre, ya sea este un contenedor padre directo, o el elemento padre por excelencia el **body**. Considerar que, si el elemento **html** tiene algún tamaño indicado, también afectará esta medida.

Todos los navegadores muestran por defecto el texto (salvo los enunciados) con un tamaño de letra de 16 px. Si realizamos la operación mencionada, 1 em equivale entonces a 16 px.

Los h1 son, por defecto, el doble de la medida base, por eso, si lo analizamos con el inspector de elementos, veremos la siguiente regla predeterminada.

```
h1 {
      display: block; font-size:
      2em;     margin-block-start:
      0.67rem;   margin-block-end:
      0.67rem;      margin-inline-
      start: 0px; margin-inline-
      end: 0px; font-weight: bold;
      }
```

Em puede implementarse a cualquier tipo de propiedad que admita medidas de longitud. En el ejemplo debajo se ha trabajado el valor de un margen en referencia al tamaño tipográfico.

```
p {font-size: 32px; margin: 0.5em;}
```

Dado que nuestros párrafos poseen **32 px** de font-size, entonces **0.5 em** de margen será equivalente a la mitad, por tanto, el margen tiene como resultado **16 px.**

Para decidir si utilizar o no em en *paddings* y *margins,* se debe considerar el diseño a replicar. Si el Diseñador/a desea que en la adaptación a diferentes *breakpoints* (o tamaños de pantalla), al modificar el tamaño tipográfico el *padding* y el *margin* se vean afectados proporcionalmente, entonces se deberá trabajar con em.

Veamos un ejemplo más práctico: en el casoa continuación el tamaño total de ese `padding-top` será el resultado del tamaño tipográfico del `body` sumado al tamaño tipográfico de contenedordel párrafo, junto con el propio tamaño del párrafo.

```css
body {
    font-size: 120%;
}

#contenedor {
    font-size: 2em;
}

p {
    font-size: 2em;
    padding-top: 1.5em;
}
```

En este ejemplo:

- Tamaño tipográfico del **`body:`** `120% 16` = **`19,2`**
- Tamaño tipográfico del **`contenedor:`**
  2 * 19.2 (tamaño tipográfico del body) = **`38,4`**
- Tamaño tipográfico del **p**:
  2 * 38,4 (tamaño tipográfico del contenedor) = **`76,8`**

Entonces:

Tamaño del **`padding-top:`**

`1.5` * `76,8` (tamaño tipográfico del párrafo) = **`115.2`**

Si, en cambio, el padding-top fuese manejado por rem, el padding-top sería solo afectado por la etiqueta html, con su valor por defecto (16px) porque no se indicó otro valor.

```css
body {
    font-size: 120%;
}

#contenedor {
    font-size: 2em;
}

p {
    font-size: 2em;
    padding-top: 1.5rem;
}
```

En este ejemplo:

- Tamaño tipográfico del **body:** 120% 16 = **19,2**
- Tamaño tipográfico del **contenedor:**
  2 * 19.2 (tamaño tipográfico del body) = **38,4**
- Tamaño tipográfico del **p**:
  2 * 38,4 (tamaño tipográfico del contenedor) = **76,8**

Entonces:

Tamaño del **padding-top:**

1.5 * 16 (tamaño tipográfico del html) = **24**

Efemérides: unidades ex

Existe otra medida ya no utilizada en CSS llamada ***ex***. La unidad **ex** funciona igual a **em**, pero en este caso la referencia es la altura de la letra **x (equis minúscula)**, siendo su valor aproximadamente **la mitad que el de la unidad em**.

Esta medida difícilmente se vea utilizada hoy en día dejando, siempre paso a medidas como rem, em, vh, vmin, etc.

Ventajas de las unidades relativas

¿Por qué trabajar con rem en vez de px si no deseamos en el contexto de diseño que el *padding* o el *margin* se vean modificados por el tamaño tipográfico?

La principal ventaja es que si en un futuro, (y esto suele presentarse muchas veces) deseamos que **todos esos *paddings* y *margins* aumenten o disminuyan de forma conjunta, solo nos bastará con modificar los atributos de html en nuestra hoja de estilos.** En cambio, si se hubiesen trabajado con px, deberíamos modificar uno por uno para poder hacer cualquier modificación o cambio.

Por ejemplo:

```css
html {
    font-size: 80%;
}

body {
    font-size: 120%;
}

#contenedor {
    font-size: 2em;
}

p {
    font-size: 2em;
    padding-top: 1.5rem;
}
```

En este caso:

- Tamaño tipográfico del **`html`**`:80%16 =` **`12,8`**
- Tamaño tipográfico del **`body:`** `120% 12,8 =` **`15,3`**
- Tamaño tipográfico del **`contenedor:`**
  2 * 15,3 (tamaño tipográfico del body) = **`38,4`**
- Tamaño tipográfico del **p**:
  2 * 38,4 (tamaño tipográfico del contenedor) = **`76,8`**

Entonces:

Tamaño del **`padding-top:`**

`1.5 * 12,8` (tamaño tipográfico del `html`) = **`19,2`**

Trabajar con rem

rem es una unidad de medida que posee una diferencia muy importante con **em**, **no hereda desde su elemento padre, sino desde el elemento raíz del documento, es decir desde la etiqueta <html>**, de ahí viene su nombre ***root em***.

Veámoslo con un ejemplo: en el caso de **em**, si un contenedor posee **2em** de font-size y a su vez un h1 contenido tiene en el css ese mismo tamaño tipográfico, el resultado final será de **4em**, dado que los valores se multiplican. En cambio, rem no duplica su valor, puesto que siempre toma al *root* como referencia.

Es importante entender esta diferencia para poder dejar elementos que no cambien su tamaño en base al cambio de tamaño base del elemento padre y así controlar en ambos casos la adaptabilidad de los elementos.

Ambas unidades de medida son necesarias al momento de trabajar en la **creación de proyectos responsivos**, permitiéndonos trabajar de forma más cómoda y eficiente.

Trabajar con porcentajes

Como se vio anteriormente, **el porcentaje es una unidad de medida relativa,** que se utiliza mucho para fijar una base de referencia para el trabajo con em y rem en la propiedad font-size.

Por su importancia, CSS trata al porcentaje de forma separada a em y px. Un porcentaje está formado por un valor numérico seguido del símbolo **%** y **siempre está referenciado a otra medida.**

Cada una de las propiedades de **CSS** que permiten indicar como valor un porcentaje, define el valor al que hace referencia **ese porcentaje**.

Los porcentajes se pueden utilizar por ejemplo para establecer el valor del tamaño de letra de los elementos:

```
html {
      font-size: 80%;
}
body {
      font-size: 120%;
}
```

El uso más común de los porcentajes es establecer la anchura de los elementos:

```
div#contenido {width: 600px;}
div.principal {width: 80%;}

<div id=¨contenido¨>
  <div class=¨principal¨> </div>
</div>
```

En el ejemplo, la referencia del **valor 80%** es la anchura de su elemento padre. Por tanto, el elemento ** ** cuyo atributo class vale principal tiene una anchura de **80% x 600px = 480px**.

Trabajar con porcentajes: imágenes responsivas y elementos multimedia

En el caso de las imágenes y en general cualquier elemento multimedia, el uso de **%** combinado con propiedades como **max-width** nos permiten trabajar de manera responsiva.

Un recurso común es **implementar la propiedad max-width para que los elementos multimedia nunca excedan su tamaño original,** y así no pixelar los mismos ni deformarlos.

Evitar el uso de height también es fundamental para que estos elementos no pierdan su proporción.

Ejemplos:

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

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

Uso de picture

Las imágenes si bien puede resolver su adaptabilidad a través del uso de porcentajes, eso no siempre funciona.

Por ejemplo, una imagen rectangular generalmente nos traerá algún tipo de problema no posible de resolver con **%**.

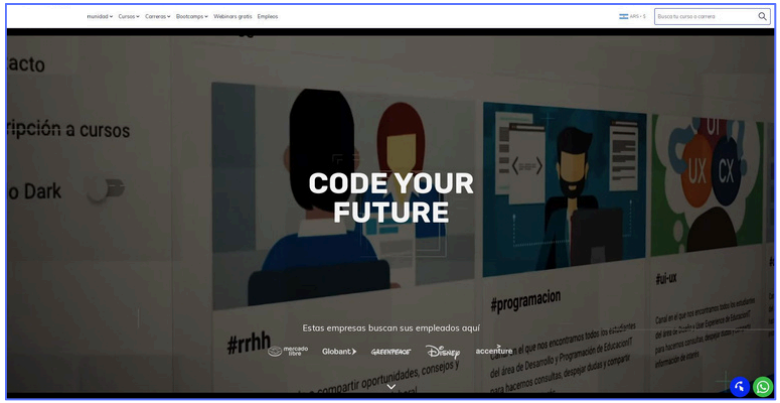
El uso de **picture**, que **permite detectar diferentes *breakpoints* y elegir la imagen deseada**.

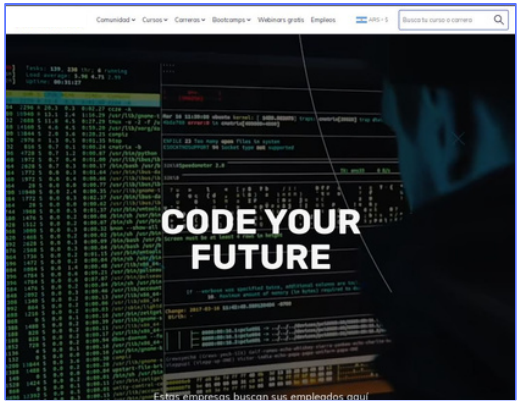
En el ejemplo debajo:

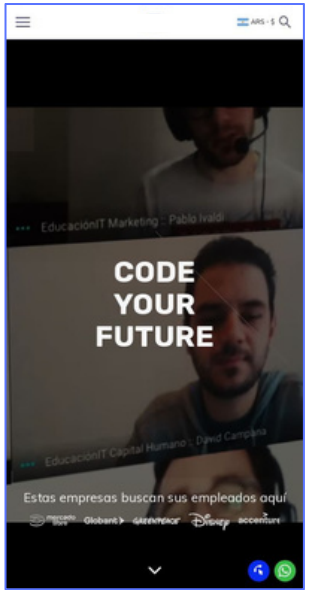
- en pantallas de **768 px o inferiores** se seleccionará ***imagen-1.png***,
- en pantallas entre **769 px** y **1024px** se seleccionará ***imagen-2.png*** y,
- en pantallas **superiores a 1024px** elegir la imagen por defecto, que es la i***magen-3.png***.

```
<picture>
<source media="(max-width: 768px)" srcset="imagenes/imagen-1.png">
<source media="(max-width: 1024px)" srcset="imagenes/imagen.2.png">
<img src="imagenes/imagen-3.png" alt="imagen">
</picture>
```

Ejemplo de diseño responsivo







Object Fit

Propiedad Object-Fit

Propiedades adicionales

Agregaremos algunas propiedades a tener en cuenta al trabajar con CSS.

`object-fit`

Esta propiedad es similar al uso que damos a **background-size** en sus valores posibles.

Sin embargo en este caso, lo hacemos sobre la propia imagen inserta.

Si tenemos inserta una imagen, en nuestro HTML, pero el tamaño indicado no es el que verdaderamente tiene la imagen, ésta se deformará.

```
<img src="foto.jpg"  width="400" height="100">
```

La propiedad **object-fit** permite adaptarla a formas diversas acorde al contenedor donde se encuentra inserta.

Probemos en nuestros estilos los siguientes valores posibles:

```
img {object-fit: fill;}
img {object-fit: contain;}
img {object-fit: cover;}
img {object-fit: scale-down;}
```

Nota: No utilizar todos los valores en forma conjunta porque se sobreescribirán unos a otros, es decir, **prevalecerá el valor scale-down, porque está en último lugar.** Sugerimos probar los valores de a uno para apreciar su diferencia. También siempre **los resultados serán distintos si la imagen es *portrait* o cuadrada o, en su defecto, rectangular.**

Ahora, tenemos un `header` con estas medidas:

```
header {

    width: 100%;
    height: 50vh;
}
```

E insertamos en el header una imagen:

```
<header>

    <img src="imagenes/foto.jpg" alt="">


</header>
```

En principio, la imagen se mostrará excediendo los límites de su contenedor porque esa es la condición natural de las imágenes.

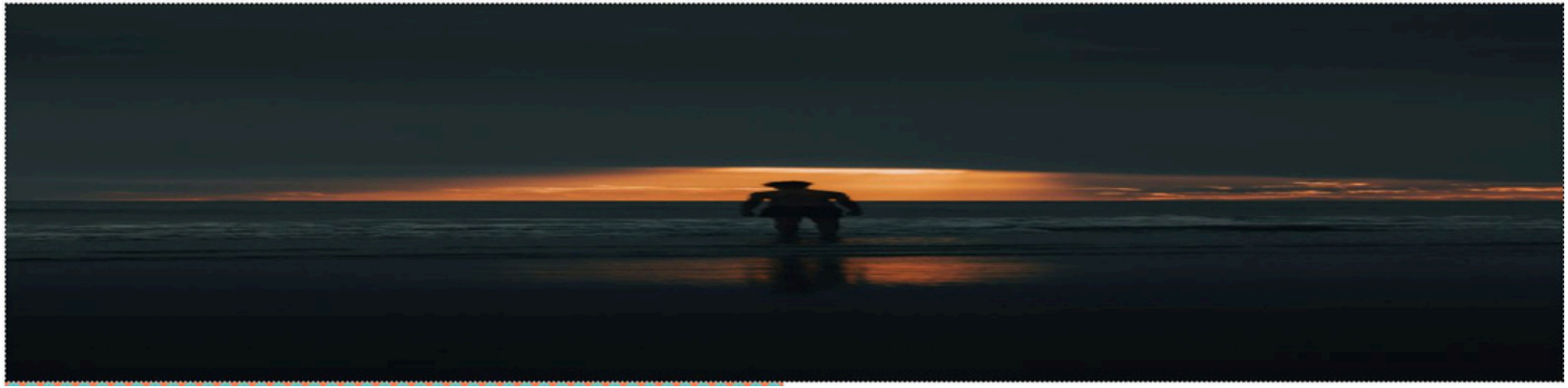
Ahora, veamos cómo aplicando un tamaño, ésta toma la referencia de su contenedor:

```
header {

    width: 100%;
    height: 50vh;
    border:1px dashed black;
}


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

Probablemente, el resultado no será el deseado, porque la imagen se deformará:



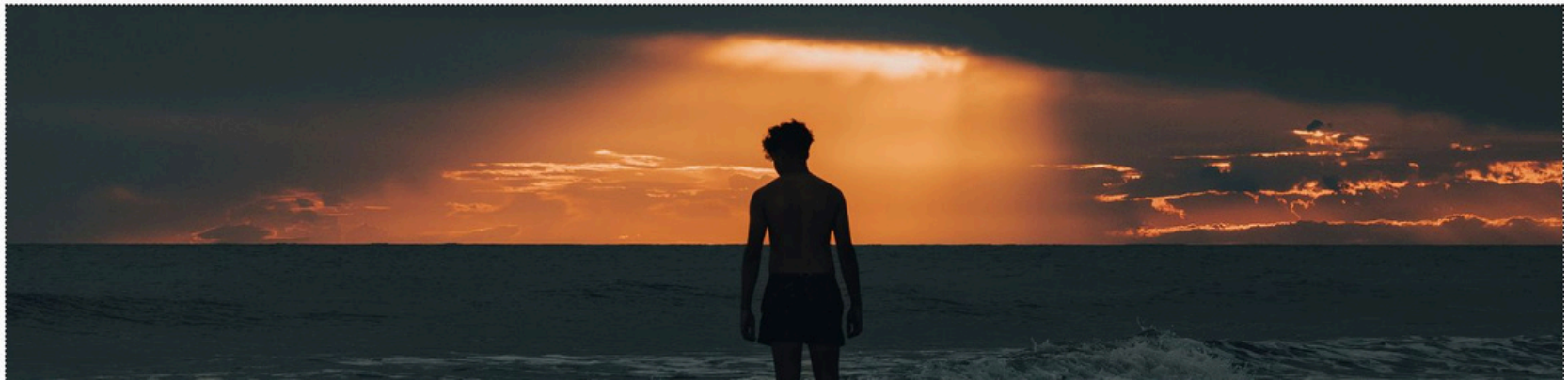
Probemos ahora con el valor **contain**, como se muestra en el ejemplo a la derecha.

La imagen se contendrá en el contenedor, pero no cubrirá toda la superficie del mismo, como puedes ver en la referencia debajo:

```css
img{
    width: 100%;
    height: 100%;
    object-fit: contain;
}
```

Si agregamos ahora la propiedad **object-fit** con el valor **cover**, la imagen cubrirá todo el fondo pero sin deformarse, aunque dejará elementos por fuera del contenedor.

```
img{
    width: 100%;
    height: 100%;
    object-fit: cover;
}
```



Conclusiones

Debemos siempre priorizar nuestro público y el diseño elegido. Si trabajamos correctamente nada debería generar el sacrificar esta línea en pos de lograr una interfaz responsiva.

Entender qué busca, usa y quiere nuestro usuario es el punto de partida, conocer luego las herramientas y cómo utilizarlas desde el maquetado: HTML y CSS completan el camino hacia una verdadera interfaz responsiva.

Nota: recomendamos utilizar search.google.com/test/mobile-friendly para medir cuán óptima es tu interfaz para dispositivos móviles.

it College