

# CSS *Avanzado* (Incompleto)

**BEM**

# BEM

En proyectos grandes, pueden haber miles de clases CSS desordenadas

- Incluso colisiones entre nombres habituales (.button, .aside, etc.)

BEM es una **estándar** para **dar nombre** a las clases CSS

# BEM

Describe tres tipos de entidades

- Bloques
- Elementos
- Modificadores

# BEM - Bloques

Un **bloque** hace referencia a una entidad **que tiene sentido en sí misma**

```
.block-name
```

Ejemplos:

```
.sidebar
```

```
.card
```

```
.section
```

```
.menu
```

# BEM - Elementos

Los **elementos** son partes de un **bloque** que **no tienen sentido por sí mismas**

```
.block-name
```

```
.block-name__element-name
```

Su clase incluye el nombre de su **bloque** conectado por **\_\_**

Ejemplos:

```
.sidebar__button
```

```
.card__img-container
```

```
.section__title
```

```
.menu__list
```

# BEM - Elementos

Bloques:

```
.menu
```

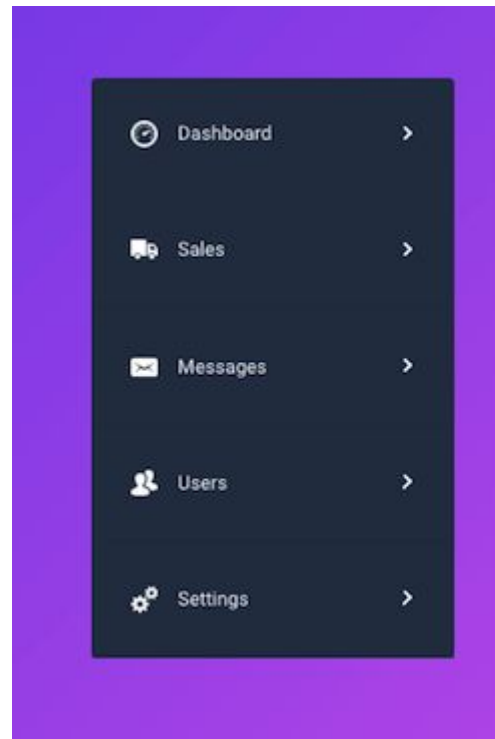
```
.option
```

Elementos:

```
.option__icon
```

```
.option__text
```

```
.option__arrow
```



# BEM - Modificadores

Los **modificadores** son **versiones alteradas** de un **bloque** o **elemento**

```
.block-name  
.block-name__element-name  
.block-name--modifier-name  
.block-name__element-name--modifier-name
```

Su clase incluye el nombre de su **bloque** o **elemento** conectado por --

Ejemplos:

```
.sidebar__button--small  
.section__title--dark-theme  
.menu__list--disabled
```



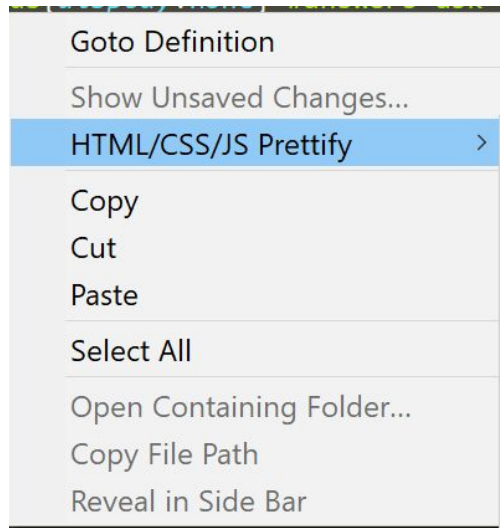
# BEM - Modificadores

Modificadores:

```
.option--unselectable
```

```
.option--expandable
```

```
.option--bordered
```



A partir de ahora  
utilizaremos BEM



**Sass**

# Sass

Sass: Syntactically awesome style syntax

Sass es un lenguaje de **hojas de estilo**

- **Transpila** a CSS (**preprocesador** de CSS)
- Permite escribir código CSS más elegante

# Sass - Scss

Sass permite dos tipos de sintaxis

- **SCSS (superset CSS)**
- SASS

## SASS Syntax

.sass

```
body
  font: 100% $font-stack
  color: $primary-color
```

.SCSS

```
body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

# Sass - Instalación

npm nos permite instalar Sass fácilmente

```
npm install -g sass
```

# Sass - Uso

Sass viene con una CLI para **transpilar** ficheros scss

```
sass --watch input.scss output.css
```

También se pueden transpilar **todos los ficheros** scss de una carpeta

```
sass --watch ruta/origen:ruta/destino
```

# Sass - Uso

Alternativas más cómodas

- Webpack
- Plugin IDE



# Sass - Variables

## Creación de variables

```
$variable-name: value;
```

SCSS    Sass

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

CSS

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

# Sass - Nesting

## Anidado de propiedades

SCSS    Sass

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  
  li { display: inline-block; }  
  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
  }  
}
```

CSS

```
nav ul {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
nav li {  
  display: inline-block;  
}  
nav a {  
  display: block;  
  padding: 6px 12px;  
  text-decoration: none;  
}
```

# Sass - Nesting

Podemos usar un **parent selector** (&) para copiar el valor del selector padre

```
#node {  
  &:hover{  
    color: #0000FF;  
  }  
}
```



#node:hover

# Sass - Nesting

Podemos usar el símbolo **&** para incluir **pseudoclases**, **pseudoelementos**, etc. en el anidamiento

```
#node {  
  color: #222222;  
  
  &:hover{  
    color: #0000FF;  
  }  
  
  &:before{  
    content: "->";  
  }  
}
```

# Sass - Nesting

Podemos usar el símbolo **&** para incluir los **elementos** de un **bloque** en sintaxis **BEM**

```
.block{
  &__elem1{
    color: #999999;
    &--modifier{
      color: #FF000;
    }
  }

  &__elem2{
    color: #333333;
    &--modifier{
      color: #FF000;
    }
  }
}
```

# Ejercicio Nesting

```
.container{  
  position: relative;  
  width: 400px;  
  height: 600px;  
}
```

```
.container p{  
  margin-top: 20px;  
}
```

```
.container img{  
  margin-top: 20px;  
}
```

```
.container p:hover{  
  color: teal;  
}
```

# Ejercicio Nesting (BEM)

```
.aside {  
  width: 30%;  
}  
  
.menu{ /* This goes inside the aside */  
  width: 100%;  
}  
  
.menu-li {  
  cursor: pointer;  
}  
  
.menu-li-icon{  
  width: 32px;  
  height: 32px;  
}
```

```
.menu-li-text{  
  font-weight: 400;  
}  
  
.menu-li-activeText{  
  background-color: #f1f1f1;  
}  
  
.menu-header{  
  font-weight: 800;  
}
```

# Sass - Modules

## Importar módulos CSS con @use

```
@use 'path'; // without file extension
```

SCSS    Sass

```
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

CSS

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}

.inverse {
  background-color: #333;
  color: white;
}
```



# Ejercicio Modules

- Crea un nuevo fichero reset.css
- Importa reset.css desde el fichero principal de estilos
- Comprueba que funciona (body debe tener 0 margin)

# Sass - Mixins

Los **mixins** nos permiten crear **funciones** en CSS

Reciben un **parámetro**, lo aplican y devuelven el código resultante

SCSS    Sass

```
@mixin transform($property) {  
  -webkit-transform: $property;  
  -ms-transform: $property;  
  transform: $property;  
}  
.box { @include transform(rotate(30deg)); }
```

CSS

```
.box {  
  -webkit-transform: rotate(30deg);  
  -ms-transform: rotate(30deg);  
  transform: rotate(30deg);  
}
```

# Sass - Mixins

@if nos permite crear mixins más flexibles

SCSS

Sass

```
@mixin avatar($size, $circle: false) {  
  width: $size;  
  height: $size;  
  
  @if $circle {  
    border-radius: $size / 2;  
  }  
}  
  
.square-av { @include avatar(100px, $circle: false); }  
.circle-av { @include avatar(100px, $circle: true); }
```

CSS

```
.square-av {  
  width: 100px;  
  height: 100px;  
}  
  
.circle-av {  
  width: 100px;  
  height: 100px;  
  border-radius: 50px;  
}
```

# Sass - Mixins

## @else

SCSS

Sass

```
$light-background: #f2ece4;
$light-text: #036;
$dark-background: #6b717f;
$dark-text: #d2e1dd;

@mixin theme-colors($light-theme: true) {
  @if $light-theme {
    background-color: $light-background;
    color: $light-text;
  } @else {
    background-color: $dark-background;
    color: $dark-text;
  }
}

.banner {
  @include theme-colors($light-theme: true);
  body.dark & {
    @include theme-colors($light-theme: false);
  }
}
```

CSS

```
.banner {
  background-color: #f2ece4;
  color: #036;
}
body.dark .banner {
  background-color: #6b717f;
  color: #d2e1dd;
}
```

# Sass - Mixins

## @else if

SCSS

Sass

CSS

```
@mixin triangle($size, $color, $direction) {  
  height: 0;  
  width: 0;  
  
  border-color: transparent;  
  border-style: solid;  
  border-width: $size / 2;  
  
  @if $direction == up {  
    border-bottom-color: $color;  
  } @else if $direction == right {  
    border-left-color: $color;  
  } @else if $direction == down {  
    border-top-color: $color;  
  } @else if $direction == left {  
    border-right-color: $color;  
  } @else {  
    @error "Unknown direction #{$direction}.";  
  }  
}  
  
.next {  
  @include triangle(5px, black, right);  
}
```

```
.next {  
  height: 0;  
  width: 0;  
  border-color: transparent;  
  border-style: solid;  
  border-width: 2.5px;  
  border-left-color: black;  
}
```

# Ejercicio Mixins

- Crea un mixin que centre verticalmente utilizando transform

# Ejercicio Mixins

- Crea un mixin que centre verticalmente y/o horizontalmente utilizando transform
- El mixin debe poder recibir por parámetro en qué direcciones hay que centrar

# Sass - Operators

SCSS permite expresar valores en forma de **operaciones matemáticas**

SCSS Sass

```
.container {  
  width: 100%;  
}  
  
article[role="main"] {  
  float: left;  
  width: 600px / 960px * 100%;  
}  
  
aside[role="complementary"] {  
  float: right;  
  width: 300px / 960px * 100%;  
}
```

CSS

```
.container {  
  width: 100%;  
}  
  
article[role="main"] {  
  float: left;  
  width: 62.5%;  
}  
  
aside[role="complementary"] {  
  float: right;  
  width: 31.25%;  
}
```



# Sass - Extend

Habitualmente, los **modificadores** incluyen estilos de la clase general

```
.error {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
  
.error--serious {  
  border-width: 3px;  
}
```

Copiarlos es tedioso y rompe el concepto **DRY**

# Sass - Extend

Scss permite implementar herencia usando **@extend**

SCSS    Sass

```
.error {  
  border: 1px #f00;  
  background-color: #fdd;  
  
  &--serious {  
    @extend .error;  
    border-width: 3px;  
  }  
}
```

CSS

```
.error, .error--serious {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
  
.error--serious {  
  border-width: 3px;  
}
```

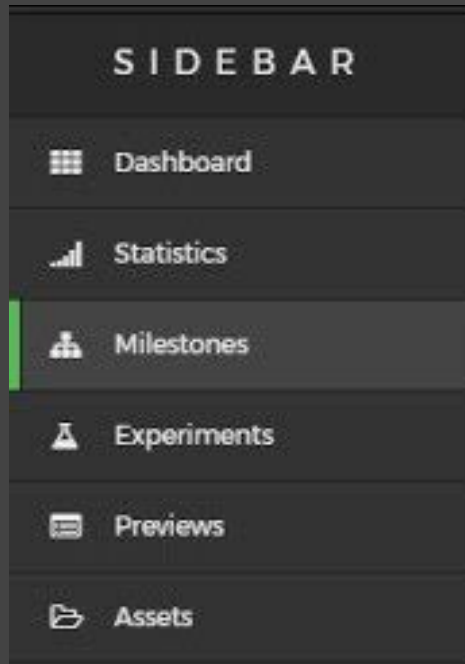
# Ejercicio extend

```
.centered-figure {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
  background-color: #222;  
}
```

```
.centered-image {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
  width: 200px;  
  height: 200px;  
}
```

```
.centered-caption {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
  font-size: 12px;  
}
```

# Ejercicio Sass & BEM



Sass - Concatenation (pendiente)

CSS

# Responsiveness

# CSS Responsive

Decimos que una página es **responsive** cuando esta se adapta a distintos tamaños de pantalla

La implementación de una página responsive requiere un esfuerzo extra por parte del desarrollador



# Demo

Chrome device toolbar

¿Cómo conseguimos que una página sea responsive?

# Media queries

Permiten aplicar propiedades CSS solamente si se **cumple una condición**

```
@media only screen and (min-width: 600px) {  
  .className {  
    background-color: lightblue;  
  }  
}
```

```
//Solo en pantallas más anchas de 600px
```

```
@media only screen and (max-height: 800px) {  
  .className {  
    background-color: lightblue;  
  }  
}
```

```
//Solo en pantallas menos altas que 800px
```

[Ejemplo](#)

# Media queries

Podemos utilizar **operadores lógicos** en las condiciones de una media query

- AND: and
- OR: ,

```
/* When the width is between 600px AND 900px OR above 1100px - change the appearance of <div> */
@media screen and (max-width: 900px) and (min-width: 600px), (min-width: 1100px) {
  div.example {
    font-size: 50px;
    padding: 50px;
    border: 8px solid black;
    background: yellow;
  }
}
```

# Mediaqueries

Generalmente son necesarias para implementar una página responsive

[Ejemplo](#)

# Ejercicio Mediaqueries

Partiendo del ejercicio anterior

- Si el ancho de la pantalla supera los 1600px el menú debe convertirse en una navbar vertical con este aspecto



# Ejercicio Responsiveness

- Refactoriza el ejercicio portfolio para que sea responsive (pantalla de iphone 8)
- Aplica metodología BEM y sintaxis SASS

# CSS Responsive

Las unidades en % también son imprescindibles para un diseño responsive, ya que se adaptan al tamaño de la pantalla

```
/* No se adapta a pantallas móviles de > 800px */  
div.example {  
  width: 800px;  
}
```

```
/* Se adapta a pantallas móviles > 800px */  
div.example {  
  width: 90%;  
}
```



# CSS Responsive

En este caso concreto seguimos pudiendo limitar el ancho a 800px en pantallas grandes

```
/* Se adapta a pantallas móviles > 800px */  
/* No supera los 800px de ancho */  
div.example {  
  width: 90%;  
  max-width: 800px;  
}
```

# Flexbox model

# Flexbox

El sistema antiguo de CSS para crear filas y columnas responsive no es muy elegante

Flexbox es un modelo que se añade en una **versión moderna** de CSS (IE11+) para mejorar esa situación

# Flexbox container

# Flexbox

Para utilizar flexbox, primero hay que crear un **contenedor flexbox**

```
.className {  
  display: flex;  
}
```

# Flexbox

Disponemos de 6 propiedades para configurar ese contenedor

`flex-direction`

`flex-wrap`

`flex-flow`

`justify-content`

`align-items`

`align-content`

# flex-direction

**flex-direction** determina en qué **dirección** se ordenan los elementos

```
.className {  
  display: flex;  
  flex-direction: value;  
}
```

Posibles valores

- [column](#)
- [column-reverse](#)
- [row](#)
- [row-reverse](#)

# flex-wrap

**flex-wrap** determina si los contenidos **saltan de línea** cuando se acaba el espacio

```
.className {  
  display: flex;  
  flex-wrap: value;  
}
```

## Posibles valores

- [wrap](#)
- [nowrap](#)
- [wrap-reverse](#)



# flex-flow

**flex-flow** es un simple atajo para escribir las dos propiedades anteriores en una

```
.className {  
  display: flex;  
  flex-flow: flex-direction-value flex-wrap-value;  
}
```

## Ejemplo

```
.className {  
  display: flex;  
  flex-flow: row wrap;  
}
```

# justify-content

**justify-content** determina como se alinean **horizontalmente** los contenidos

```
.className {  
  display: flex;  
  justify-content: value;  
}
```

Posibles valores

- [center](#)
- [flex-start](#)
- [flex-end](#)
- [space-around](#)
- [space-between](#)

# align-items

**align-items** determina cómo se alinean **verticalmente** los contenidos **dentro de la fila**

```
.className {  
  display: flex;  
  align-items: value;  
}
```

Posibles valores

- [center](#)
- [flex-start](#)
- [flex-end](#)
- [stretch](#)
- [baseline](#)

# align-content

**align-content** determina cómo se alinean **verticalmente** las **filas**

```
.className {  
  display: flex;  
  align-content: value;  
}
```

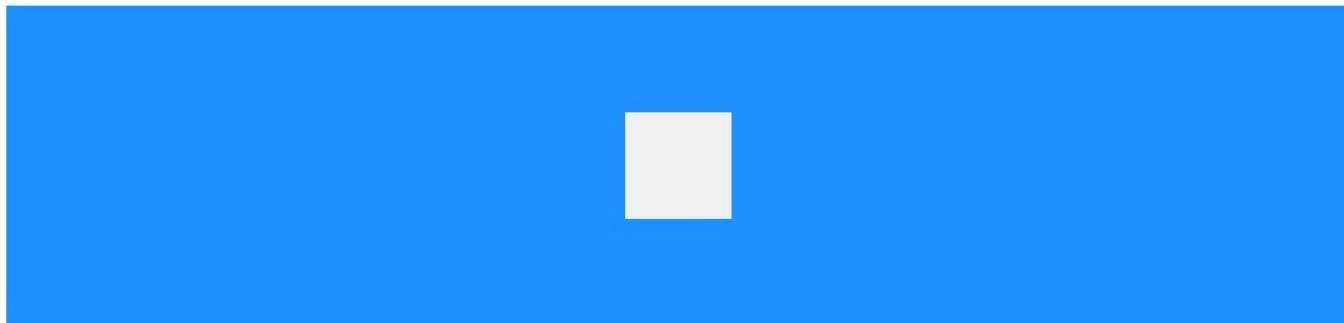
Posibles valores

- [space-between](#)
- [space-around](#)
- [center](#)
- [flex-start](#)
- [flex-end](#)
- [stretch](#)
- [baseline](#)

# Centrado perfecto con flexbox

Centrar un elemento en un contenedor es muy fácil con flexbox

```
.parent-container {  
  justify-content: center;  
  align-items: center;  
}
```



# Ejercicio Flexbox

- Avanza hasta el nivel 13 en [Flexbox-froggy](#)

Flexbox children

# align-content

A los hijos de un contenedor flex les llamamos **flex items**

Disponemos de 6 propiedades para configurar los atributos de esos items

- `order`
- `flex-grow`
- `flex-basis`
- `flex`
- `align-self`



# order

order determina el orden en el que se posicionan los **ítems**

```
#item4 {  
  order: 1;  
}
```

```
#item2 {  
  order: 2;  
}
```

```
#item1 {  
  order: 3;  
}
```

```
#item3 {  
  order: 4;  
}
```



Ejemplo

# flex-grow

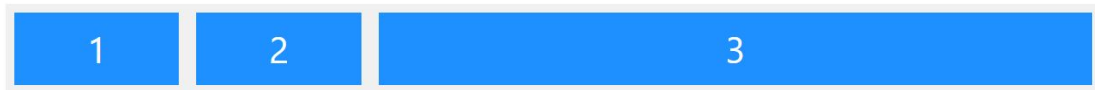
**flex-grow** determina cuánto crecerá un elemento **relativo a los demás**

- **flex-grow: 0** evita que el ítem crezca

```
#item1 {  
  flex-grow: 1;  
}
```

```
#item2 {  
  flex-grow: 1;  
}
```

```
#item3 {  
  flex-grow: 8;  
}
```



[Ejemplo](#)

# flex-grow

**flex-shrink** determina cuánto decrecerá un elemento **relativo a los demás**

- **flex-shrink: 0** evita que el ítem decrezca

```
#item3 {  
  flex-shrink: 0;  
}
```



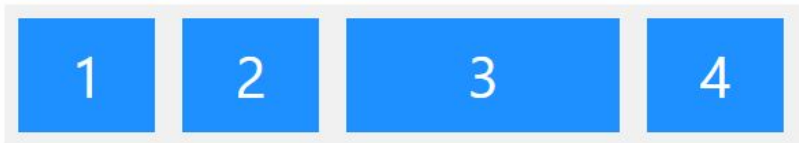
Ejemplo

# flex-basis

**flex-basis** determina el **ancho inicial** de un item (forzando el resto a que se adapten)

- when flex-direction is **column**, flex-basis controls **height**.

```
#item3 {  
  flex-basis: 200px;  
}
```



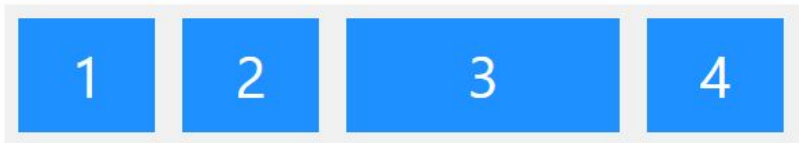
Ejemplo

# flex-basis

**flex-basis** determina el **ancho inicial** de un item (forzando el resto a que se adapten)

- when flex-direction is **column**, flex-basis controls **height**.

```
#item3 {  
  flex-basis: 200px;  
}
```



Ejemplo

# flex (ítem)

**flex** es un atajo para escribir las propiedades flex del ítem en una línea

```
#item {  
  flex: flex-growth-value flex-shrink-value flex-basis-value;  
}
```

## Ejemplo

```
#item1 {  
  flex: 0 0 200px;    // No crecerá ni decrecerá  
}
```

# align-self

Hace **override** de la propiedad **align-items** del **contenedor**

```
#item3 {  
  align-self: center;  
}
```



[Ejemplo](#)

# justify-content

**justify-content** determina como se alinean **horizontalmente** los contenidos

```
.className {  
  display: flex;  
  justify-content: value;  
}
```

Posibles valores

- [center](#)
- [flex-start](#)
- [flex-end](#)
- [space-around](#)
- [space-between](#)



# Ejercicio Flexbox

- Avanza hasta el nivel 20 en [Flexbox-froggy](#)

Calc

# calc

**calc** permite utilizar expresiones matemáticas para asignar valor a una propiedad

Se puede utilizar cualquier unidad en los cálculos (px, %, vh, em, etc.)

```
.className {  
  width: calc(50% - 20px);  
  height: calc(100vh - 40px);  
  margin: calc(10px * 2);  
}
```

Es necesario **dejar un espacio en blanco** entre los operadores (+, -)

# Ejercicio Calc

- Crea un contenedor que ocupe un 100% del ancho excepto 50px que debe dejar de margen a cada lado

# Pseudoelementos

# Pseudo-elements

Los **pseudoelementos** nos permiten dar estilo a una parte concreta de nuestro elemento

```
.className::pseudo-element {  
  property: value;  
  Property: value;  
}
```

Ejemplo:

```
p::first-line {  
  color: #ff0000;  
  font-variant: small-caps;  
}
```

# Pseudo-elements

Hay varios tipos de pseudoelementos ([referencia](#))

Nos centraremos en los más usados:

- Before
- After


# Pseudo-elements

**before** y **after** nos permiten **añadir contenido** antes o después de nuestros elementos respectivamente (sin tener que añadir más HTML)


```
h1::before {  
  content: url(smiley.gif);  
}
```

## Ejemplos:

- [Before](#)
- [After](#)

 **This is a heading**

The ::before pseudo-element inserts content before the content of an element.

 **This is a heading**

**Note:** IE8 supports the content property only if a !DOCTYPE is specified.



# Pseudo-elements

**before** y **after** también permiten añadir texto

```
h1::before {  
  content: "Esto aparece antes del h1 -> ";  
}
```

# Pseudo-elements

[Más ejemplos](#)

# Ejercicio pseudoelementos pendiente

# Animaciones CSS

# CSS Animations

Las animaciones CSS nos permiten crear transiciones entre estilos con un alto nivel de flexibilidad

La propiedad keyframes nos permite crear una animación

```
@keyframes example {  
  from {background-color: red;}  
  to {background-color: yellow;}  
}
```

# CSS Animations

También podemos especificar keyframes en forma de porcentajes

```
@keyframes example {  
  0%    {background-color: red;}  
  25%   {background-color: yellow;}  
  50%   {background-color: blue;}  
  100%  {background-color: green;}  
}
```

Ejemplo

# CSS Animations

Podemos modificar el comportamiento de las animaciones mediante varias propiedades

`animation-name`

`animation-duration`

`animation-delay`

`animation-iteration-count`

`animation-direction`

`animation-timing-function`

`animation-fill-mode`

# CSS Animations

**animation-delay** determina un tiempo inicial de espera hasta que la animación empiece

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-delay: 2s;  
}
```



# CSS Animations

**animation-iteration-count** determina el número de veces que se repite la animación antes de volver a su **estado inicial**

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-iteration-count: 3;  
}
```

# CSS Animations

**animation-iteration-count** acepta un valor infinito

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-iteration-count: infinite;  
}
```

# CSS Animations

**animation-direction** determina si la animación se reproduce del derecho o del revés

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-direction: reverse;  
}
```

# CSS Animations

La propiedad **animation-timing-function** nos permite especificar una curva (como en transition) a la animación

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-timing-function: ease-in;  
}
```

# CSS Animations

La propiedad **animation-fill-mode** determina si el elemento mantiene el estado inicial o el estado final de la animación

**forwards** mantiene el estado inicial, **backwards** el final y **both** mantiene el estado inicial antes y el estado final después de la animación

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-fill-mode: forwards;  
}
```

[Ejemplo](#)

# CSS Animations

```
div {  
  animation-name: example;  
  animation-duration: 5s;  
  animation-timing-function: linear;  
  animation-delay: 2s;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
}
```

Podemos condensar las propiedades de animation en una única propiedad

```
div {  
  animation: example 5s linear 2s infinite alternate;  
}
```

# Ejercicio animaciones

<https://s3-us-west-2.amazonaws.com/s.cdpn.io/455884/CSSTransitionsAnimation-s-04.mov.gif>

# Ejercicio animaciones II

<https://s3-us-west-2.amazonaws.com/s.cdpn.io/455884/CSSTransitionsAnimation-s-07.mov.gif>



# Cross-browser compatibility

# Cross-browser compatibility

Los navegadores se encargan de renderizar los contenidos de las páginas

Por tanto tienen que ir **evolucionando** junto con los lenguajes web para adaptarse a las nuevas features

No todos los navegadores evolucionan al mismo ritmo y no todos los usuarios tienen la versión más nueva de su navegador

De ahí surge el problema **cross-browser compatibility**

# Cross-browser compatibility

En CSS algunas propiedades modernas requieren diferentes nombres según el navegador al que estemos dando soporte

```
.example {  
  -webkit-transform: "translateX(20px)";  
  -ms-transform: "translateX(20px)";  
  transform: "translateX(20px)";  
}
```

Los **mixins** de Sass son bastante útiles en esos casos

```
@mixin translate($x, $y) {  
  -webkit-transform: translate($x, $y);  
  -ms-transform: translate($x, $y);  
  transform: translate($x, $y);  
}
```

# Cross-browser compatibility

Puedes comprobar si una propiedad requiere prefijos [aquí](#)

Hemos visto cómo solucionamos la compatibilidad de Javascript mediante

- Babel
- Polyfills

En CSS también disponemos de **autoprefixers** que aplican prefijos de forma automática en función de los navegadores a los que queremos dar soporte

Demo autoprefixer

Iconos

# Iconos

Podemos descargar iconos en forma de imagen en [Flaticon](#)

Una forma más cómoda de gestionar los iconos de nuestra página es utilizar **fuentes iconográficas**

Demo icoMoon