HTML y CSS

Class	Finalizado
Reviewed	☑
Certificado	☑

Introducción a HTML, etiquetas y atributos

Un html es un documento de texto, es un lnguaje de marcado de hipertexto. Es un estandar definido por W3C. Tienen unas etiquetas que las cambian por contenido embebido,las marcas por contenido enriquecido. Lo invento `Tim Berners-Lee`, además es fundador de W3C. Esta en la versión 5, desde los 2014/2015 aproximadamente. Hay un validador ⇒ validator.w3.org , es un lenguaje interpretado, google se fija en el codigo html para poder indexar los buscadores, es importante para poder realizar estrategias de SEO. No es un lenguaje de programacion, porque no tiene bucles, vectores, funciones, etc. Una pagina web normalmente se sirve desde un servidor web, desde un https, hay que instalar un certificado de seguridad, esta peticion va a estos nodos que cambian a una ip, dependiendo de los lenguajes de programacion, interactua normalmente con BD, procesan la lógica de esa peticion, devuelve casi siempre un documento web, lo recoge el navegador y lo interpreta.

IDE ⇒ editor de código.

Documento html, metadatos(head), informacion de la pagina(body y el title que esta en el head).

Metadatos

Otro metadato es ⇒ `<meta description=”La descripcion de nuestra página”/>`

```
<link rel="author" href="http://www.twitter.com/xxx" />
<link rel="help" href="http://aaa.com/ayuda" />
<link rel="search" href="http://aaa.twitter.com/busqueda" />
<link rel="license" href="http://aaa.com/licencia" />
```

Favicons:

```
<link rel="icon" href="/favicon.png" /> <!--HTML5 -->
```

Styles:

```
<link rel="stylesheet" href="index.css" type="text/cs"/>
```

Facebook open graph:

```
<meta property="og:type" content="article" />
<meta property="og:title" content="titulo del post" />
<meta property="og:image" content="http://pagina.com/img/imagen32.jpg" />
<meta property="og:description" content="Descripción de la página" />
```

Twitter:

```
<meta property="twitter:card" content="sumary_large_image" />
<meta property="twitter:creator" content="@Manz" />
<meta property="twitter:title" content="Titulo" />
<meta property="twitter:description" content="Description" />
<meta property="twitter:image:src" content="URL_img.jpg" />
```

Estructura-etiqueta:

```
<strong id="dato">Contenido</strong>
```

Atributos:

```
<div id="pagina">
    <div class="anuncio primero">Aqui ira un anuncio</div>
    <div id="anuncio primero"></div>
    <div class="anuncio primero">
    <a href="google.com" title="Ir a google"/>
</div>
```

span-translate

```
<p>
  <span translate="no"></span>
</p>
<p dir="rtl">
  <span translate="no"></span>
</p>
```

Formularios y tablas

Etiquetas para formularios

La etiqueta `form` ⇒ Lo que hace es agrupar una serie de elementos como inputs o checkbox que lo que hace que el usuario pueda introducir sus datos en este formulario. `Label` nos permite poner una etiqueta a nuestra entrada de datos, nuestro `inputs`. El `for` hace referencia a un `input` que tengamos. El atributo `action` hace referencia a la url que nosotros queremos enviar en este formulario. Cuando usamos formularios tenemos dos métodos `method` ⇒ `Get` (), `Post` (envia la info a resultados esperando que haga algo con esa info).

`textarea` ⇒ Para ingresar texto.

```
<form action="resultados.html" method="POST">
    <div>
        <label for="nombreusuario">Nombre de usuario</label>
        <input name="nombreusuario" id="nombreusuario" required>
    </div>
    <div>
        <label for="password">Contraseña</label>
        <input name="password" id="password" required/>
    </div>
    <div>
        <label for="email">Cual es tu email?</label>
        <input type="email" name="email" id="email"/>
    </div>
    <div>
        <label for="edad">Cual es tu edad?</label>
        <input type="number" name="edad" id="edad" min="0" max="200" stop="5"/>
    </div>
    <div>
```

```
        <label for="nacimiento">Cual es tu fecha de nacimiento</label>
        <input type="date" name="nacimiento" id="nacimiento"/>
    </div>
    <div>
          <label for="comentarios">Comentarios</label>
          <textarea name="comentarios" id="comentarios">
          </textarea>
    </div>
    <button>Enviar</button>
</form>
```

Botones

Los mas utilizados son el tipo `submit` ⇒ Que lo que hace es cuando uno le da enter lo envia, con el button `reset` ⇒ Resetea todo el formulario.

Poniendo el `required` en el `input` nos obliga a rellenar el formulario.

```
<button type="reset">Reset</button>
<buttion type="submit">Enviar</button>
```

Tablas

Con la etiqueta `table`. Las tablas van por filas para agregar se pone `tr`

```
<div>
    <h3>horas para aprender HTML</h3>
    <table>
        <tr>
          <th>Lunes</th>
          <th>Martes</th>
          <th>Miercoles</th>
          <th>Jueves</th>
          <th>Viernes</th>
          <th>Sabado</th>
          <th>Domingo</th>
        </tr>
        <tr>
          <td>Maria</td>
          <td>9-13</td>
          <td>9-13</td>
          <td>9-13</td>
          <td>9-13</td>
          <td>Descanso</td>
          <td>Descanso</td>
        </tr>
        <tr>
          <td>Iñigo</td>
          <td>10-17</td>
          <td>10-17</td>
          <td>Descanso</td>
          <td>10-17</td>
          <td>Descanso</td>
          <td>Descanso</td>
        </tr>
    </table>
</div>
```

Multimedia

Imágenes

`src` ⇒ Que vamos tipo de imágen vamos renderizar

`alt` ⇒ Mensaje que se va a mostrar en caso que no se pueda mostrar la foto.

```
<div>
    <img src="logo1.webp" alt="Logo HTML5" width="300" height="300"/>
</div>
```

Videos y audio

Hay tres tipos de code de videos que soporta hml5, que es el webm, mp4, egg.

controls ⇒ Muestra los botones para controlar el video.

autoplay ⇒ Automaticamente se reproduce apenas se cargue la pagina

muted ⇒ Desactiva el sonido del video

loop ⇒ Continuamente reproduce el video

```
<video width="300" height="300" controls autoplay muted loop>
    <source src="Intro.mp4" type="video/mp4">
</video>
```

Tres tipos de audio ⇒ mp3, wav, ogg.

```
<audio autoplay controls loop>
  <source src="gnr-welcome.mp3" type="audio/mp3" />
</audio>
```

Introducción al CSS

Introducción a las hojas de estilo

Selectores en CSS

De etiqueta, de clase, por id…etc

Las tres formas de insertar estilos

Colores

Fondos de colores e imágenes

Background position: left bottom.

Estilos de altura,anchura, padding y margin

Fuentes en Css

font-size

font-weight

font-style

mejor importar las fuentes con import@

Disposición de elementos y para multimedia

Disposiciones y alineaciones

Disposiciones por defecto dentro del html.

Los div por defecto `son block.`

Los span por defecto `son inline.`

Las imagenes por defecto `son de inline-block.`

Con la propiedad `float` podemos cambiar de lugar los elementos `inline ⇒ right, left, bottom,`

Posicionamiento

Donde disponemos los elementos dentro del html.

`Static ⇒` Viene por defecto, no se puede editar.

`Relative ⇒` Nos permite posicionarlo al posicionamiento original.

`Fixed ⇒` Se suele utilizar en barras de navegación. Se calcula en funcion del modelo absoluto, esta fijado con respecto a alguna referencia. No colapsa con otros margenes. Siempre se va a mantener fijos.

`Absolute ⇒` Nos permite posicionarlo igual que fixed pero en relacion a su elemento padre. Siempre tiene que tener top o bottom.

`z-index:` Nos dice como es su posicionamiento, si va a estar debajo o encima del elemento.

`Sticky ⇒` Utiliza una mezcla entre el relative y el fixed

FlexBox

Caja Flexible, utilizado en modos responsivos.

```
<body>
  <div class="flex">
    <div class="contenido-1"></div>
    <div class="contenido-2"></div>
    <div class="contenido-3"></div>
  </div>
</body>
```

Nos posiciona todo dentro del div .flex, contiene todo dentro de ese div y lo posiciona de una manera ordena, por defecto lo hace en una row o fila. En flex para posicionar se usa `justify-content:center/start/end/space-between/space-around;` en el eje de las X.

Hay dos ejes uno que es el principal X(horizontal) e Y, y segun como le digamos que es su eje principal los va alineando.

`flex-wrap: nowrap/wrap;` redimenciona el elemento a medida que la pantalla es mas pequeña.

`align-items: strech/center/flex-start/flex-end` alineacion en funcion del contenido.

Con `flex-direction: row/row-reverse/column/column` Si lo ponemos en una column, al cambiar los ejes el alig-items y el justify-content cambian.

Si tenemos diferentes filas podemos utilizar la propiedad `align-content` es una prop mas avzanda, divide las filas que se van creando en base a su wrap.

```
.flex{
  display: flex;
  justify-content: center;
  flex-direction: column;
  background-color: yellow;
  heigth: 300px;
  flex-wrap: nowrap;
  align-items: center;
}
.contenido-1{
  background-color: white;
  height: 200px;
```

```
  width: 100px;
  border: 1px solid blue;
}
.contenido-2{
  background-color: white;
  height: 200px;
  width: 100px;
  border: 1px solid blue;
}
.contenido-3{
  background-color: white;
  height: 200px;
  width: 100px;
  border: 1px solid blue;
}
```

Overflox

Desbordamiento,pasa si tenemos un contenedor de tamaño fijo, y tenemos texto como queremos que sea el comportamiento del texto para que no se desborde.

Tiene cuatro valores posibles

hidden ⇒ Todo lo que sobresale se oculte

scroll ⇒ Se crea una barra de scroll que permite scrollear el contenido.

visible ⇒

auto ⇒ Su contenedor es automatica, nos optimiza para el viewport donde estamos.

Podemos ponerle para que lado de la coordenada queremos que lo haga, en el X o en Y.

```
.contenedor {
    backgroun-color: bisque;
    width: 300px;
    height: 200px;
    paddin: 15px;
    overflow: hidden;
}
.contenedor-1 {
    backgroun-color: bisque;
    width: 300px;
    height: 200px;
    paddin: 15px;
    overflow-x: hidden;
    overflow-y: scroll;
}
```

Trabajando la opacidad y galerías de imágenes

opacidad ⇒ Se mide entre 0 y 1. Se utiliza en el hover, cuando se pasa por encima se puede activar este estado.

```
*{
  margin: 0;
  paddin: 0;
}
.imagen-principal{
    width: 100vw;
    opacity: 0.5;
}
.imagen-principal:hover{
    opacity: 1;
}
```

Reproductores de vídeo

```
<video class="video" playsinline autoplay muted loop>
  source src="img/video.mp4" type="video/mp4"
</video>
<div class="">
  <h1>Invita San sebastian</h1>
  <button>Llamada a la acción</button>
</div>
```

CSS:

```
*{
  margin: 0;
  padding: 0;
}
html{
  font-family: Helvetica;
  color: white;
}
.video{
  width: 100vw;
  height: 100vw;
  object-fit: cover;
  position: fixed;
}
.header{
  height: 100vw;
  position: relative;
  text-align: center;
  display: flex;
  flex-direction: column;
  gap: 10px;
  justify-content: center;
  align-items: center;
}
```

Anidación de selectores

Anidación de selectores

```
<div class="contenedor">
  <p>Esto es un parrafo</p>
  <p>Esto es un parrafo</p>
  <p>Esto es un parrafo</p>
  <p>Esto es un parrafo</p>
  <button id="boton-contenedor">Boton</button>
</div>
<p>Esto es un parrafo</p>
```

```
.contenedor p {color:green}
```

```
.contenedor #boton-contenedor{color: yellow}
```

```
body section .main .contenedor p {color: red}
```

Pseudoclases

Se utiliza dentro de nuestro codigo css, para hacer referencia a diferentes elementos dentro de nuestro html.

Hover

```
.hover{
  width: 300px;
  height: 100px;
  background-color: bisque;
}
```

```
.hover:hover {
  background-color: red;
}
```

First-Child

Hace referencia al primer hijo independientemente del elemento

```
.listado {
  padding: 15px;
  background-color: red;
}

.listado p:fist-child{
  color: while;
}
```

Link, Visited, Active

Hace referencias a links no visitados y visitados.

```
a:link {
  color: blue;
}
a:visited{
  color: grimson;
}
a:active{
  color: green;
}
```

Pseudoelementos

Todos los elementos tienen un after(despues) y un before(antes)

```
.contenedor{
  width: 300px;
  height: 200px;
  background-color: green;
}
span.frase{
  font-style: italic;
}
span.after::before{
  content:"<<";
  font-size: xx-small;
}
span.after::after{
  content:">>";
  font-size: xx-small;
}
```

Pseudoelementos para las letras

```
.libro{
  background-color: bisque;
  padding: 15px;
  font-family: Arial;
}
.libro p::first-letter{
  font-size: xx-large;
}
.libro p::fist-line{
  color: white;
}
```

Placeholder afecta a los inputs del formulario, selection hace referencia cuando yo selecciono algo

```
.libro p::selection{
  background-color: chocolate;
  color: lime;
}
```

Afecta a las listas ordenadas y desordenadas:

```
.listado ul li::marker{
  color: red;
}
```

Especificidad

El selector de clase tiene mas selectividad que el selector de etiqueta. Hay 3 niveles de especificidad. Tiene mas especificidad si lo ponemos en el html al estilo, pero aún tiene más especificidad si le ponemos !important. De todas maneras hay que evitarlo.

#p > .clase > etiqueta

Creacion de estilos para formularios

Select ⇒ deplegable con varias opciones

Html:

```
<form action="" method="">
  <input class="input" type="text" name="name"/>
  <input class="input" type="password" name="password"/>
  <input class="input-moderno" type="text" name="name" placeholder="Tu Nombre"/>
  <input class="input-background" type="text" name="name" placeholder="Tu Nombre"/>
  <input class="input-search" type="text" name="name" placeholder="Tu Nombre"/>
  <textarea name="description"></textarea>
  <select>
    <option>Opcion 1</option>
    <option>Opcion 2</option>
    <option>Opcion 3</option>
  </select>
<form>
```

Css:

```
.input{
  with: 100%;
  height: 5px 10px;
  border: 2px solid green;
  border-radius: 10px;
  margin-bottom: 10px;
}

.input[type="password"]{
  color: red;
}
.input-moderno{
  border: none;
  border-bottom: 1px solid #c2c2c2;
  outline: none;
  padding: 5px;
  color: #5f5f5f;
}
.input-moderno::placeholder{
  color: #c2c2c2;
}
.input-moderno:focus{
```

```
  border-bottom: 1px solid #5f5f5f;
}
.input-background{
  background-color: aquamarine;
  border: none;
  border-radius: 5px;
  padding: 5px 10px;
  color: white;
  box-shadow: 0 0 10px #ccc;
}
.input-background::placeholder{
  color: #ccc;
}
.input-search{
  padding: 10px;
  padding-left: 30px;
  border: none;
  background-color: white;
  border-radius: 8px;
  margin: 10px;
  background-image: url("../img/search-icon.png");
  background-repeat: no-repeat;
  background-position: 5px;
}
textarea{
  resize: both/horizontal/vertical/none;
}
select{
  with: 100%;
  padding: 15px;
  background-color: #ccc;
}
```

Creando un formualario moderno

Bootstrap

Incluyendo Bootstrap en nuestro proyecto

Nos da estilos para que no tengamos que hacer nada en css, nos da componentes, estilos, libreria de open source, codigo abierto

Utilizado por la NASA, por muchas empresas.

Importar el css:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-0evHe/X+R7YkI
```

Importar el Js:

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pprn3073KE6tl6bjs2QrFaJG
```

Estilos y animaciones de Bootstrap

Sistema grid que tiene, esta basada en 8 columnas, breakpoints son puntos de ruptura, que es el layout que es como se comporta la pantalla:

- Extra small ⇒ < 576px
- Small ⇒ ≥ 576px
- Medium ⇒ ≥ 768px
- Large ⇒ ≥ 992px

Extra Large ⇒ ≥ 1200px

Extra extra largue ⇒ ≥ 1400px

Acepta diseño responsive, generalmente lo hariamos como media queries. Podemos editarlo:

```
$grid-breakpoints: (
  xs: 0,
  sm: 576px,
  md: 768px,
  lg: 992px,
  xl: 1200px,
  xxl: 1400px
);
```

Containers:

Containers

Containers are the most basic layout element in Bootstrap and are required when using our default grid system. Containers are used to contain, pad, and (sometimes) center the content within them. While containers can be nested, most layouts do not

https://getbootstrap.com/docs/5.2/layout/containers/

Grid divide el layout en 12 columnas, pero a la vez esas columnas dentro hay filas.

Animaciones y transiciones

Hay dos formas de animan en css, las tranformaciones y transiciones(por defecto esta `all`, cuando uso el all `se transicionan todas las animaciones`, si quiero que se le aplica a la propiedad le llamo `transform,` le podemos poner un `delay`).

```
.padre {
  width: 400px;
  height: 400px;
  backgroun-color: blue;
}
.hijo{
  width: 50%;
  height: 50%;
  background-color: white;
  /*transition: transform 1s linear/ease-in/ease-in-out/ 2s*/;
  animation: traslacion 4s ease-in-out infinite forwards alternate;
}
.padre:hover .hijo {
  /*transform: translateX(100%);
  background-color: blueviolet;*/
  animation-play-state: paused;
}
@keyframes traslacion {
  0%{
      background-color: brown;
      transform: traslateX(0%) translateY(0%);
    }
  25%{
      transform: traslateX(100%) translateY(100%);
    }
  75%{
      transform: traslateX(100%) translateY(100%);
    }
  100%{
      transform: traslateX(0%) translateY(0%);
      background-color: beige;
    }
}
```

Tooltips

Es una pista de que es lo que hace ese elemento.

Html:

```
<button class="btn-tooltip">Buscar
  <span class="tooltip">hace una busqueda<span>
<button>
<div class="triangulo"></div>
```

Css:

```
body{
  padding: 0;
  margin: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
}
.btn-tooltip{
  padding: 10px 20px;
  border-radius: 5px;
  border: none;
  background-color: green;
  color: white;
  border-bottom: 2px solid darkgreen;
  cursor: pointer;
  transition: background-color 150ms;
}
.btn-tooltip:hover{
  background-color: rgb(0,100,0);
}
/*.btn-tooltip:hover::before{
  content: "Esto es un tooltip"
}*/
.tooltip{
  position: relative;
  opacity: 0;
  top: -50%;
  left: -100%;
  transform: translateX(25%);
  color: white;
  background-color: #333;
  padding: 5px 10px;
  border-radius: 5px;
  transition: opacity 150ms
}
.btn-tooltip:hover{
  background-color: rgb(0,100,0);
}
.btn-tooltip:hover .tooltip{
  opacity: 1;
}
.tooltip::before{
  content: ""
  border: solid 110px transparent;
  border-top-color: #333;
  position: absolute;
  top: 40px;
  left: 50%;
  transform: tranlateX(-50%);
}
/*.triangulo{
  border: solid 110px transparent;
  border-top-color: #333;
}*/
```

Introducción al diseño responsivo

Unidades fijas y relativas

El diseño responsivo no es mas que adaptar nuestro diseño a diferentes medidas, segun la pantalla.

Hay 3 tipos de unidades:

- Fijas ⇒ Pixeles
- Unidades relativas ⇒ porcentajes o vw y vh

```
body{
  margin: 20px;
  padding: 0;
}
.escenario{
  margin: 10px;
  border: 1px solid #ccc;
  padding: 10px;
}
.escenario-1{
  width: 100%;
}
.escenario-2{
  width: 50%;
}
.px{
  background-color: green;
  width: 200px;/*tamaño fijo*/
}
.porcentaje{
  background-color:red;
  width: 50%; /*tamaño de su padre*/
}
.view{
  background-color: yellow;
  width: 50vw; /*tamaño de la pantalla*/
}
```

Responsive y unidades em y rem

EL viewport no es mas que el display. No solo hay que basarse en px y vw.

El diseño responsive es más que sea amigable para diferentes tipos de pantalla, no solo consiste en adaptar, sino crear una experiencia única para esa pantalla.

Las unidades que mas se suelen utilizar ⇒ em (hace referencia al tamaño donde esta contenido el elemento) y rem (hace referencia al tamaño fuente de su contenedor). Estas relacionadas con el tamaño de fuente.

```
.escenario{
  border: 1px solid #333;
  padding: 10px;
}
.escenario-1{
  font-size: 32px;
}
.escenario-2{
  font-size: 16px;
}
.em{
  font-size: 2em;
}
.rem{
  font-size: 2rem;
}
```

Medias queries

En paginas responsivas, lo más importante es el width, la anchura, nos determina si estamos en un dispositivo u otro.

Cuando escriba @media ya viene por defecto el all que afecta a todos los dispositivos. A screen caundo estamos pintando la pagina web en la pantalla. Con sprint accedemos a una impresora. Se suele usar contactenado con una condicion, con el max-width(anchura maxima).

Podemos decirle que nos calcule en landcape(mas ancho que alto), portraid (es mas alto que ancho)

```
h1{
  color:red;
  fon-size: 4rem;
}
p{
  color: black;
  font-size: 2rem;
}

@media (min-width: 576px;){
    h1{
        color: blue;
    }
}
@media (min-width: 995px;){
    h1{
        color: green;
    }
}
@media (min-height: 360px;), (min-width: 1100px;){
    p{
      color: lemon;
    }
}
@media(orientacion: landscape) and (min-height:400px;){
  p{
  color: brown;
  }
}
```

Uso avanzado de Flexbox

Hay dos posiciones, flex y grid. Hay dos elementos claves, el contenedor y dentro del contenedor tenemos los items.

```
<div class="contenedor">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
  <div class="item item4">Item 4</div>
</div>
```

CSS:

```
.contenedor{
  background-color: yellow;
  padding: 0.5rem;
  display:flex;
  gap: 1rem;
  flex-direction: column;
  min-height: 5rem;
  flex-wrap: wrap/nowrap;
}
.item{
  background-color: salmon;
  padding: 0.25rem;
  width: 10rem;
}
.item2{
  align-self: stretch;
}
```

Hay dos ejes principales dentro de los contenedores ⇒ eje principal(horizontal), y el eje secundario(vertical), por efecto esta en `row`, para darlo vuelta los ejes le pongo al `flex-direction: column;`

El eje principal utiliza ⇒ `justify-content: center/start/end/space-between/space-around/space-evenly`

El eje secundario utiliza ⇒ `align-items: center/baseline/strech(por defecto)/end`

Para los items ⇒

`justify-self: center`

`align-self: stretch`

Viewport, Grid y templates

En grid, necesitamos un condenedor o elemento padre.

```
<div class="contenedor">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
  <div class="item item4">Item 4</div>
  <div class="item item5">Item 5</div>
  <div class="item item6">Item 6</div>
  <div class="item item7">Item 7</div>
  <div class="item item8">Item 8</div>
  <div class="item item9">Item 9</div>
</div>
```

CSS:

```
.contenedor{
  display: grid;
  gap: 0.5rem;
  column-gap: 0.1rem;
  row-gap: 0.1rem;
  grid-template-columns: auto auto auto;
  grid-template-columns: repeat(3,auto);
  grid-template-columns: 20px 3rem 25vw;
  grid-template-columns: 1fr 1fr 2fr;
  grid-template-rows: 2fr 1fr 1fr;
  grid-template-areas: "head head head"
                       "main main main"
                       "footer footer footer";
}
.item{
  background-color: aquamarine;
  padding: 0.25rem;
  font-size: 2rem;
  border: solid 1px blue;
}
.item1{
  grid-column: 2/2;
  grid-area: head;
}
.item2{
  grid-area: 2/1/4/3;
}
.item3{
  grid-area: footer;
}
```

Debemos decirle como distribuir los items. Tambien podemos de trabajar con areas, con el `grid-template-areas`. Con el `grid-column: 2 / 3`, podemos decirle de donde queremos que comience y donde finalice. Con el `grid-row: 2 / 2`

Variables reutilizables en CSS

Se definen de la siguiente manera:

```
:root{
  --color-principal: rgb(1,187,255);
  --ancho-principal: 250px;
}
.titulo{
  color: var(--color-principal);
}
```

Sistema Grid de Bootstrap

Ver en la parte de Available Breakspoint

Usando pre-procesadores CSS

Introduccion a SASS

Un preprocesador lo que hace, es que escribimos en un lenguaje, debemos convertirlo en otro lenguaje que nuestro nevegador sea capaz de leerlo. Necesitamos un compilador.

Debemos ponerle a la extensión, hay dos tipos de archivos `styles.scss` y `styles.sass`(nos elimina las llaves y las comas). Tenemos la posibilidad de utilizar extensiones, instalar `Live Sass Compiler,` dar click en la parte inferior de `Vs Code`, para interpretar sass

Variables y mixins

Variables: Se definen con el signo del dolar

```
$text-color: #2b2b2b;
```

Existen un elemento dentro del scss que se llaman `mixins`, es una forma de reutilizar código, también acepta variables:

```
@mixin formato-texto($bg-color){
    color: $text-color: #f1f1f1;
    text-transform: uppercase;
    width: 50rem;
    background-color: $bg-color;
}
h1{
    @include formato-texto()
}
.listado{
    width: 10rem;
    background-color: aquamarine;
    padding: 0.5rem;
    ul{
        margin: 0 0.5rem;
        li{
            list-style: none;
            &:hover{
                color: white;
            }
        }
    }
}
```

Importacion de extensiones

Ponerle de nombre al archivo `_componentes.scss` (aquello que tengan el barra baja, el compilador nos los va a leer, debemos importarlo en nuestro css principal, llamado index.scss) entonces lo

diferenciamos del resto:

```
.btn{
  border: none;
  padding: 0.25rem 0.5rem;
  color: #b2b2b2;

  &:hover{
    color:red;
    background-color: black;
  }
}
```

De esta manera podemos importar componentes:

```
@import "sass/componentes";
@import "sass/mixins"
@import "sass/estilos"
@import "sass/pagePrincipal"
```

> Buenas prácticas es aislar cada uno de nuestros componentes, mixins, diferentes páginas y estilos e importarlos en el index.css principal.

Se pueden extender un estilo que ya este definido a otro componente:

```
.btn-2{
  @extend .btn
  margin: 0.5rem;
  border-radius: 0.25rem;
}
```

Instalacion global SASS y Use vs Import

A partir del 2019 empezaron con una nueva version de sass, se empieza a usar `@use y @forward.` Para poder utilizar las nuevas funcionalidades, hay que instalar en la terminal `⇒ npm install -g sass.`

Para compilar en tiempo real, debemos decirle que archivos vamos a utilizar `sass --watch style.scss style.css.`

En vez de utilizar el import, usar el `use`, que nos crea un namespace, a traves del cual podemos acceder, y se llamaria `componentes.$color`, o `mixins.$color`:

```
@use "sass/componentes" as comp;
@use "sass/componentes"
```

Funciones SASS

Funciones ya configuradas, para facilitarnos el trabajo.

`Strings`, el debug va a pasar por la terminal, nos ayuda a ver por pantalla antes de verlo en el css, se utiliza con las tipografias, podemos calcular el tamaño de las strings:

```
@use "sass:string"
@debug "hola";
@debug string.index("Hekvetica Neue","Neue");
$string: "Helvetica";
$newString: string.insert($string, "Bold", 9)

@debug $newString
@debug string.length($newString)
```

```
@debug string.slice($newString, 5, 9)

@debug string.to-upper-case("Roboto Mono")
@debug string.to-lower-case("Roboto Mono")

@debug string.unique-id();
```

Numbers:

```
@use "sass:math"

@debug math.$e;
@debug math.$pi;

@debug math.ceil(4.5);
@debug math.floor(4.6)
@debug math.clamp(0,256.36,10)

@debug math.max(3,4,10,2,235)
@debug math.min(2,5,1,543)
@debug math.round(4.6)
```

Lists:

```
@use "sass:list"

$lista1: 10px 10px 0px 15px

@debug $lista1:
@debug lista.append($lista1, 25px, space);

$border-custom: solid 1px #ccc;
@debug list.index($border-custom, solid)
```