**Preprocesadores CSS (UT3)**

|  |  |
| --- | --- |
| **El concepto de preprocesador** | Un preprocesador de CSS es una herramienta que nos permite escribir pseudo-código CSS que luego será convertido a CSS real. Ese pseudo-código se conforma de variables, condiciones, bucles o funciones. Podríamos decir que tenemos un lenguaje de programación que genera CSS. El objetivo de estos preprocesadores es tener un código más sencillo de mantener y editar. |
| **Preprocesadores más extendidos** | Sass: <http://sass-lang.com/>  Less: <http://lesscss.org/>  Stylus: <http://stylus-lang.com/>  PostCSS: <http://postcss.org/> |
| **El preprocesador Sass** | En esta Unidad de Trabajo nos centraremos en el preprocesador SASS. Actualmente, SASS es el preprocesador de CSS que más cuota de uso tiene. |
| **Instalación de Sass** | Existen varias formas de instalar Sass que se detallan aquí: <http://sass-lang.com/install>  Nos vamos a centrar en Dart Sass, que es la implementación principal de Sass y la primera en recibir todas las características. Tiene un repositorio oficial en Github: <https://github.com/sass/dart-sass>  Para realizar la instalación tenemos varias opciones:   1. Instalar Dart Sass desde el repositorio de GitHub: <https://github.com/sass/dart-sass/releases/tag/1.15.2>  * Descargamos la versión que deseemos (es multiplataforma) y descomprimimos el paquete en el directorio que queramos. * Añadimos al PATH el directorio del paquete que acabamos de descargar. Cómo añadir al PATH el directorio: <https://katiek2.github.io/path-doc/>  1. Instalar Dart Sass mediante el instalador (versión Windows) <http://www.gekorm.com/dart-windows/>   **Mediante Node.js**  Instalación en node y posterior instalación de Sass en su versión de pura de JavaScript (es más lenta, pero la interfaz es igual que en el resto):   1. Instalar node: <https://nodejs.org/en/> 2. Abrimos una terminal en Visual Studio Code. 3. Ejecutamos este comando: npm install -g sass   Npm es el gestor de paquetes de paquetes de Node.js  Una vez que lo tengamos instalado, podemos lanzar desde una terminal (PowerShell o dentro del propio Visual Studio Code) el siguiente comando para verificar la versión actual de Sass instalada en nuestro sistema: sass –version  **Nota importante: Ruby Sass se encuentra en proceso de desaparición.** |
| **Primeros pasos en Sass** | Los ficheros en Sass pueden tener **sintaxis scss**, que es la más accesible para comenzar a trabajar con preprocesadores ya que es similar a CSS y supone un menor reto para introducirse en esta tecnología.  Para comenzar, crearemos un fichero “primerficherosass.scss”.  Escribiremos una serie de reglas respetando la sintaxis scss y compilaremos el fichero:  div {  color: #00ff00;  width: 97%;  height:auto;    .cajaroja{  background-color: #ff0000;  color: #000000;  width:50%;  height:400px;  p{  color:yellow;  }  }  }  Si compilamos lo anterior, obtendremos este código css:  div {  color: #00ff00;  width: 97%;  height: auto;  }  div .cajaroja {  background-color: #ff0000;  color: #000000;  width: 50%;  height: 400px;  }  div .cajaroja p {  color: yellow;  }  /\*# sourceMappingURL=primerficherosass.css.map \*/  **Compilaremos el fichero scss para obtener el fichero de salida css:**  sass primerficherosass.scss primerficherosass.css  **También podemos decidir que el fichero se autocompile cada que le demos a guardar:**  sass --watch primerficherosass.scss primerficherosass.css  **Compilación de una carpeta completa**  sass –watch directorioorigen:directoriodestino  Ejemplo:  sass –watch ./carpetaSCSS:./carpetaSCSS  **Importante**: los ficheros que comiencen por \_ no generarán ficheros de salida al ser compilados. |
| **Variables en Sass** | Mediante estas variables podemos almacenar colores, fuentes… cualquier valor en CSS que queramos reutilizar. Las variables se declaran con el símbolo $ delante del identificador que deseemos.  Ejemplos:  $fuente1: Helvetica, sans-serif;  $colorprincipal: rgb(187, 57, 57);  /\*En este ejemplo vamos a aprender una nueva propiedad: font. Sirve para establecer una familia de fuente y un tamaño de texto\*/  body{  font: 1em $fuente1;  color: $colorprincipal;  } |
| **Anidaciones** | Mediante las anidaciones de reglas en Sass se puede conseguir un código mucho más claro que con CSS3 puro.  Ejemplos de anidaciones de reglas:  Ejemplo 1 con selectores hijos:  .flexcontenedor{  display:flex;  background-color:dodgerblue;  flex-wrap:wrap;  justify-content: center;  > div{  background-color: #f1f1f1;  width: 140px;  margin: 10px;  padding: 20px;  font-size: 30px;  line-height: 75px;  text-align: center;  }  }  Ejemplo 2 con selectores descendentes:  .contenedorflex{  display:flex;  background-color: dodgerblue;  width:100%;  div{  background-color: #f1f1f1;  margin: 10px;  padding: 20px;  font-size: 30px;  }  }  Otro ejemplo con selectores descendentes:  /\*En este ejercicio aprenderemos un nuevo elemento en HTML: nav. Se utiliza para crear un conjunto de enlaces orientados a la navegación dentro del sitio web\*/  nav {  ul {  margin: 0;  padding: 0;  list-style: none;  }  li { display: inline-block; }  a {  display: block;  padding: 6px 12px;  text-decoration: none;  }  } |
| **Importaciones** | Mediante Sass es posible importar otros ficheros scss más pequeños y más fáciles de mantener. Aquí entra en juego el concepto de “parciales” que pueden ser importados en otros ficheros Sass y que permiten modularizar nuestro proyecto.  **Importante:**  Los ficheros parciales que comiencen por \_ no generarán salida al ser compilados por un comando que compile automáticamente todos los ficheros de una carpeta.  Ejemplo:  Creamos un fichero llamado ‘\_sassimportar.scss’ con el siguiente contenido:  html,  body,  ul,  ol {  margin: 0;  padding: 0;  letter-spacing: 1em;  }  Posteriormente, en otro fichero llamado ‘proyectosass.scss’ importamos el fichero ‘\_sassimportar.scss’  @import 'sassaimportar.scss';  $fuente1: Helvetica, sans-serif;  $colorprincipal: rgb(187, 57, 57);  /\*En este ejemplo vamos a aprender una nueva propiedad: font\*/  body{  font: 1em $fuente1;  color: $colorprincipal;  } |
| **Operadores** | Realizar operaciones matemáticas en CSS puede resultar muy útil.  Operadores aritméticos:  +, -, \*, / y %  Hay que tener cuidado porque estos operadores aritméticos sólo funcionan con unidades compatibles.  La mejor forma de comprobar todo esto que vamos a hacer es echar un vistazo al css generado mediante la compilación del fichero scss.  Ejemplos:  h2{  font-size: 5px + 2em; /\*Aquí hay un error. Unidades incompatibles\*/  font-size: 5px + 2; /\*El resultado sería 7px\*/  }  h3{  font-size:5px \* 2px; /\*Esto arrojaría un error. No se pueden multiplicar dos números que tengan la misma unidad\*/  font-size: 5px \* 2;  }  Otro ejemplo con operadores aritméticos y roles. En este caso vamos a realizar una rejilla fluida con dos elementos dentro de un contenedor y tomando como base 960px de ancho.  Más información sobre los roles en el fichero Roles-ARIA que se encuentra en esta misma carpeta.  Para ampliar esta información, podemos visitar en los siguientes [enlace](https://www.w3.org/TR/html-aria/) y [enlace.](http://nosolousabilidad.com/articulos/wai_aria.htm)  En el fichero scss:  .container {  width: 100%;  }  article[role="main"] {  float: left;  width: 600px / 960px \* 100%;  }  aside[role="complementary"] {  float: right;  width: 300px / 960px \* 100%;  }  En el html:  <div class="container">  <article role="main">  A la izquierda  </article>  <aside role="complementary">  A la derecha  </aside>  </div> |
| **Mixins** | Los mixins permiten ahorrar mucho código cuando se utilizan continuamente las mismas propiedades para los elementos CSS.  Su funcionamiento es similar a las subrutinas de los lenguajes de programación y pueden recibir parámetros.  **Su función fundamental es permitir la reutilización de código de una forma muy sencilla.**  En el siguiente ejemplo se van a utilizar una serie de propiedades específicas para cada navegador que normalmente requieren un gran trabajo de código. Mediante los mixins podemos ahorrar mucho trabajo:  @mixin border-radius() {  -webkit-border-radius: 3px;  -moz-border-radius: 3px;  -ms-border-radius: 3px;  border-radius: 3px;  }  .caja1{  @include border-radius;  border: 1px solid green;  width:33%;  float:left;  box-sizing:border-box;  }  .caja2{  @include border-radius;  border: 1px solid blue;  width:33%;  float:left;  box-sizing:border-box;  }  .caja3{  @include border-radius;  border: 1px solid purple;  width:33%;  float:left;  box-sizing:border-box;  }  Los mixins también se pueden utilizar con **parámetros. En este ejemplo vamos a ahorrar aún más código perfeccionándolo.**  @mixin border-radius($radius) {  -webkit-border-radius: $radius;  -moz-border-radius: $radius;  -ms-border-radius: $radius;  border-radius: $radius;  width:33%;  float:left;  box-sizing:border-box;  }  .caja1{  @include border-radius(3px);  border: 1px solid red;  }  .caja2{  @include border-radius(3px);  border: 1px solid red;  }  .caja3{  @include border-radius(3px);  border: 1px solid red;  } |
| **Referencia al selector padre** | Para ahorrar código en las anidaciones, Sass permite hacer referencia al selector padre de la anidación con el caracter &  Ejemplo: & sería equivalente a escribir a.dinamico  a.dinamico {  color: red;  &:hover {  background-color: #ff6;  }  }  Código CSS equivalente:  a.dinamico {  color: red;  }  a.dinamico:hover {  background-color: #ff6;  }  Como podemos observar, se puede ahorrar bastante código.  Podríamos hacer referencia a la metodología BEM, pero no disponemos de suficiente tiempo en el módulo para ello. Más información sobre la metodología BEM [aquí.](http://getbem.com/) |
| **Herencia** | Es un mecanismo por el cual un selector puede recibir estilos CSS que han sido declarados con anterioridad.  Para utilizar la herencia se va a hacer uso de las clases placeholder. Estas clases no van a aparecer en el código CSS resultante de la compilación si no se utilizan.  Ejemplo de una **clase placeholder** (comprobamos que no aparece en el código CSS generado hasta que no se utiliza).  %encabezado{  background-color: blanchedalmond;  color: brown;  font-family: 'Times New Roman', Times, serif;  }  Para utilizar esta clase placeholder se usa @extend  Ejemplo:  h1 {  @extend %encabezado;  font-size: 2em;  }  h2 {  @extend %encabezado;  font-size: 1.5em;  }  **¡Importante!**  **Como podemos ver, la funcionalidad es similar a utilizar mixins, pero en este caso es más sencillo. Debemos centrarnos en los mixins únicamente cuando necesitamos parámetros o realizar cálculos más complejos. Además, @extend introduce un código más optimizado para estos casos, ya que las reglas de estilo no se repiten dos veces en cada selector.**  ¡FUNDAMENTAL!  Ejemplo de la potencia que tiene esta característica @extend. Es muy importante que observemos el código CSS generado.  Ejemplo:  %caja {  padding: 10px;  font-size: 1.2em;  }  %caja-borde {  border: 1px solid #ddd;  }  %caja-fondo {  background-color: #f0f0f0;  }  %caja-espaciadoextra {  padding: 20px;  }  .caja {  @extend %caja;    &-borde {  @extend %caja;  @extend %caja-borde;  }    &-fondo {  @extend %caja;  @extend %caja-fondo;  }  &-espaciadoextra {  @extend %caja;  @extend %caja-espaciadoextra;  }  &-combinada {  @extend %caja;  @extend %caja-borde;  @extend %caja-fondo;  @extend %caja-espaciadoextra;  }  }  Con esto nos ahorramos muchas clases a la hora de escribir el código HTML y evitamos situaciones de este tipo:  <div class="caja caja-borde caja-fondo caja-espaciadoextra">  Este código es muy engorroso  </div>  En su lugar obtendremos esto:  <div class="caja-combinada"> ... </div> |