**Apuntes de SASS**

Intalar SASS Desde consola:

|  |  |
| --- | --- |
| sudo dnf install nodejs | Instalar node |
| sudo npm install -g sass | instalar sass globalmente |

Iniciar SASS desde el proyecto vsc:

|  |  |
| --- | --- |
| Npm init | Crear recursos |
| npm install sass | instalar sass |

Primeros pasos:

-Crear el index.html

-Crear una carpeta sass, con un archivo styles.scss

|  |  |
| --- | --- |
| Sass –-watch sass:styles | -crea el segimiento de los cambios en tiempo real  sass:styles  Indica que seguirá la carpeta sass y compilara en la carpeta styles (si no existe la crea). |

Para dejar el seguimiento ctrl+c

**Tipo de variables**

|  |  |
| --- | --- |
| $var1: red; | Color |
| $var: 1;  $var: 12px; | Número  Número (con unidad). |
| $var: ‘hola’;  $var: hola; | String  String (pero este no lleva comillas ojo). |
| $var: false;  $var: true; | Booleano |
| $var: 12 13 14;  $var: 12, 13, 14;  $var: (12)(13)(14);  $var:[12 13 14];  $var:[]; | Lista  lista vacia |
| $var: (‘color’:black, ‘tam’:12 ); | mapa |

Para ver que tipo de variable es usamos

|  |
| --- |
| @use ‘sass:meta’;  @debug meta.tipe-of($var); |

**Interpolación y Manejo de strings**

Se refiere a sacar las comillas de un string

|  |
| --- |
| $var: ‘seccion1’;  .#{$var}{  …..  } |

Ahora en caso de querer usar un string como propiedad debemos usar una función del modulo string

|  |
| --- |
| $var: ‘center’;  h1{  justify-content: string.unquote(%var);  } |

Asi mismo para poner comillas usarmos la función quote.

|  |  |
| --- | --- |
| $var:head;  .container{  grid-area: string.quote($var);  } | Variable string sin comillas. **Pero estas no pueden tener espacios. Ej: $var: head head head** |
| $var:head head head;  .container{  grid-area: ‘#{$var}’;  } | Solucion  grid-area: ‘head head head’; |

Para concluir usar interpolacion para los selectores y usar string para las propiedades . Ver cada caso.

**Dividir en sass**

Un error común es cuando queremos poner una variable para las grid $var: 1/100 ; es interpretado como división.

para evitar errores usamos división con la función

|  |
| --- |
| @use ‘sass:math’;  @default math.div(12,2); |

**Error al multiplicar números**

Al multiplicar números con unidad hay un error muy común de elevar la unidad al cuadrado ej: **10px \* 10px = 100px² .** Lo correcto seria **12 \* 12px =100px**

**Listas propiedades etc.**

Las listas no pueden ser modificadas una vez creadas.

|  |  |
| --- | --- |
| @use ‘sass:list’;  $varlista:12 13 14; | **Nota importante en sass todo comienza desde 1 a +** |
| @debug list.nth($varlista, 1) | Devuelve el valor en el index 1 (ej 12) |
| $nuevalista: lista.append($varlista, 15);  $varlista:lista.append($varlista,15);  $var: lista.append($var1,var2); | Crea una nueva lista y añade un elemento al final de la lista  Tambien se le puede asignar la nueva lista a la misma variable.(hack)  Tambien sirve para unir dos listas |
| @debug lista.index($varlista,14); | Devuelve el index en que se encuentra el valor, null sin no lo encuentra (ej 3) |

**Mapas**

|  |  |
| --- | --- |
| @use ‘sass:map’;  $varMap:(  ‘selector’: ‘seccion1’,  ‘colorPrimario’: red,  ‘tam’: 12px  ); | Declaración |
| .#{map.get($map1,'selector')}{  background-color: map.get($map1,'color');  } | Uso |
| $nuevoMapa: map.set($viejoMapa, “key”, valor); | Sirve para agregar o modificar |
| $nuevoMapa: map.merge($mapa1, $mapa2); | Juntar dos mapas |

**NESTING (ANIDAMIENTO)**

|  |  |
| --- | --- |
| **SASS** | **CSS** |
| Nesting  h1{  p{}  h2{}  } | H1 p{}  h1 h2{} |
| H1>{  p{}  h2{}  }  …  H1{  >p{}  h2{}  } | H1>p{}  h1>h2{}  H1>p{}  h1 h2{} |
| H1{  ~{  p{}  h2{}  }  h3{}  } | h1~p{}  h1~h2{}  h1 h3{} |

Los media queris que son estan dentro, hacen referencia al padre**tambien con @keyframes**

**MODULARIZACIÓN**

-Se crea una carpeta nueva para guardar los modulos.

-Dentro de la carpeta se guardan los modulos con nombre de **\_nombre.scss.** El guion evita que se cree un css en la carpeta styles

|  |  |
| --- | --- |
| Importar | |
| @use ‘nombre’;  @use ’\_nombre’;  @use ‘\_nombre.scss’ | Es lo mismo |
| @use ‘modulos/nombre’; | En caso de tenerlo en carpeta |

|  |  |  |
| --- | --- | --- |
| Styles | \_modulo |  |
| @use ‘\_modulo’;  @debug modulo.suma(12); | @function suma($var){  @return $var +12;  } | Demostración para llamar a una variable, función o mixin de un modulo |

Importar modulo con variable modificadas

|  |  |  |
| --- | --- | --- |
| Styles | \_modulo | !default permite ser modificado al importar  Para modificar varias variables son separadas por ‘,’  y no lleva ; tampoco nada al final |
| @use ‘variables.scss’ with (  $varcolor:green  ); | $varcolor:red !default; |

**Scoope de las variables en CSS vs SASS**

|  |  |
| --- | --- |
| **Css** | **SASS** |
| En css las variables locales (custom properties ) están disponibles para el selector y sus hijos | En sass solo esta disponible para ese bloque de código |

Esto puede ser solucionado con el anidamiento:

|  |
| --- |
| .body{  $varColor:red;  h1{  color:$varColor;  }  } |

Cambiar el valor de una variable global de manera local:

|  |  |
| --- | --- |
| $varColor:red;  body{  $varColor:blue !global;  } | Variable global  Se modifica globalmente con !global |

Uso de media queris y anidamiento:

las media queris hacen referencia al padre también aplica a los **@keyframes**

Podemos forzar la referencia al padre con & por lo tanto tenemos buenas practicas:

|  |  |
| --- | --- |
| h1{  :hover  &:hover{}  } | h1 :hover (incorrecto)  h1:hover (correcto) |

BEM con anidamiento

|  |
| --- |
| .card{  &\_\_title{...}  &\_\_paragraph{...}  } |

**CONDICIONAES IF**

Al igual que javaScript toma solamente como condicion falsa a (false o null)

lo demás es verdadero o trully

|  |
| --- |
| @if{  [}@else](mailto:}@else)if{  ...  [}@else](mailto:}@else){  ...  } |

**CICLOS FOR, EACH**

En sass se comienza desde 1 a infinito

|  |  |
| --- | --- |
| @for $var from 1 to 10{  selector#(var){  }  } | $var toma valores del 1 al 9 (no incluye al 10) |
| @for $var from 1 through 10{} | $var toma valores del 1 al 10 (si incluye al 10) |

Recorrer una lista con each

|  |  |
| --- | --- |
| $varLista: 1 2 3 4 6;  @each $i in $varLista{} | 5 iteraciones y $i toma valores 1 2 3 4 6 |

Recorrer una matriz con each

|  |  |
| --- | --- |
| $var: 'hero' red 12px, 'nav' green 11px;  @each $selector, $color, $tam in $var{  .#{$selector}{  color:$color;  font-size: $tam;  }  } | Matriz de ejemplo  ejemplo de uso |

Recorrer un mapa con each

|  |  |
| --- | --- |
| @each $clave,$valor in $varMap{  } | $clave se cargara con las claves  $valor se cargara con los valores |

|  |  |
| --- | --- |
| recorrer maps  **$varLista:'nav' red 20px, 'header' blue 20px;**  @each $selector, $color, $tam in $varLista{  .#{$selector}{  background-color: $color;  width: $tam;  }  } | va de 1 a 10  bluce de 5 reps y $i toma 1 2 3 4 6 en cada iteracion  listas anidadas (matriz) |
| Mixin (procesos)  @mixin centrarConFlex{  display: flex;  justify-content: center;  align-items: center;  }  h1{  @include centrarConFlex;  }  **con parametros**  @mixin crearGrid($rows, $cols, $gap){  grid-template-rows: repeat($rows, 1fr);  grid-template-columns:repeat($cols,1fr);  gap: $gap;  }  h1{  @include crearGrid(2,3,20px);  }  parametros por default  @mixin crearGrid($rows:1, $cols:1){...} | Declaracion del maxin  incluir el pedazo de codigo: |
| Shorthands inverso  h1{  margin: {  top:0px;  left:2px;  }  } | H1{  margin-top:0px;  margin-left:2px;  } |
| @mixin ejemplo($color , $tam , $selector...) {  @each $i in $selector {  .#{$i}{  background-color:$color;  font-size: $tam;  }  }  }  @include ejemplo(red, 12px, head, nav);  mandar bloques de codigo a un mixin---  algo como mandar parametros pero codigo  @mixin ejemplo($color , $tam , $selector...) {  @each $i in $selector {  .#{$i}{  background-color:$color;  font-size: $tam;  &:hover{  @content;  }  }  }  }  @include ejemplo(red, 12px, header, nav){  background-color: blue;  font-size: 2rem;  } | Para metros infinitos  1- se tiene que poner los tres puntos en el ultimo argumento, este será tomado como una lista por eso la recorremos  al final agregamos parametros que deseamos  aca en content ira el codigo injectado  codigo que mandamos {….} |
| @mixin ejemplo($color , $tam , $selector...) {  @each $i in $selector {  .#{$i}{  background-color:$color;  font-size: $tam;  &:hover{  @content($color);  }  }  }  }  @include ejemplo(red, 12px, header, nav) using($color){  background-color: $color;  font-size: 2rem;  } | En el ejemplo anterior no se podian acceder a las variables del mixin..  para poder hacerlo agreagamos:  la variable que vamos usar  definir con using las que usaremos  la usamos de manera normal |
| Funcioness  @use 'sass:math';  @function pixelesToEm($pixeles){  $aux: math.div($pixeles, 1px);  @return math.div($aux,16);  } | Declaracion de la funcion con parametros  retorno de la funcion y fin de la funcion. Es obligatorio |
| Pasar argumentos sin importat el orden ej:  @mixin proceso($color:red, $tam:16px){  background-color: $color;  font-size:$tam;  }  h1{  @include proceso($tam:18px);  }  para llamar un modulo con una modificacion previa  ….modulo..  $color: red !default;  ...principal..  @use ‘mudulo’ with ($color:black); | (primero el color, despues el tamaño)  aca obiamos el color y nos dirigimos directamente al tamaño con su nombre de referencia y su valor  default indica que se puede cambiar en caso de ser necesario  aca se importa con la modificacion, usar (,) para agregar otra variable a modificar |
| Variables, mixin, funciones privadas  -colores: blue, red, black; | Ahora solo es accesible desde el modulo en que se encuentre. |

**Buenas practicas**

nombrar al scss principal **styles.scss**

una forma de ordenar los módulos es ponerlos en una carpeta ej: módulos.

Para que saturar el archivo principal se crea un archivo intermedio este contiene todos los modulos que vamos a usar en el proyecto. Obligatoriamente tiene el nombre **\_index.scss**  es un modulo que contiene a los otros módulos.

|  |
| --- |
| @forward ‘colores.scss’; |

Con use importamos normalmente para usar variables, funciones o mixins. Forward sirve para que poder acceder desde la hoja principal (styles.scss) al modulo en si.

|  |
| --- |
| @use ‘modulos/index.css’  @debug index.$colorPrincipal; |

Alias c

on fordward

si tenemos dos variables con el mismo nombre en distintos módulos y evitar el error es necesario poner alias a los modulos:

|  |
| --- |
| @forward ‘colores’ as col-\*;  @forward ‘tamanios’ as tam-\*; |

El guion es opcional pero queda mas entendible.

Ahora para llamar a una variable , funcion o mixin desde el archivo principal seria:

|  |  |
| --- | --- |
| @use ‘index’;  @debug index.$col-colorPrincipal;  @debug index.col-funcSumar(20px,50px);  [h1{@include](mailto:h1{@include) index.col-proceso(red);} | Para variables  funciones  mixin |

**HIDE:**Poner en privado una variable, funcion, mixin desde el fordwar

si bien se puede poner desde el propio modulo con el (\_ o -) tambien se puede desde el index.scc

|  |
| --- |
| @fordwar ‘modulo’ hide $variable, funcSumar, procGrid; |

**SHOW:** funciona radicalmente diferente, con show todas los VFP son privadas menos las que se agregan:

|  |
| --- |
| @fordwar ‘modulo’ show $variable, funcSumar, procGrid; |

**WITH** :cambiarle el valor por defecto desde el fordwar

|  |  |
| --- | --- |
| En el index.scss | En el módulo |
| @fordward ‘modulo’ with($color:red) | $color:green !default; |

Nota: es posible hacerlo desde el archivo principal (styles.scss) pero no es buena practica.

Extend: es algo similar a la herencia. Se pueden heredar de cualquier etiqueta

Mejora la legibilidad

|  |  |
| --- | --- |
| .h1{  color:red;  background-color:blue;  }  h2{  **@extend .h1;**  font-size:12px;  } | .h1, .h2{  color:red;  background-color:blue;  }  .h2{  font-size:12px;  } |

Cuidado con los anidamientos

|  |  |
| --- | --- |
| @extend .h1 !optional; | Quiere decir que si no no encuentra la etiqueta/clase no agreaga nada. Y no tira error. |

Para heredar de un modulo o index, a diferencia de las VFM, no hace falta poner **@extend .modulo.h1**

solamente

**@extend .h1;**

**Placeholders:** pone en privado el selector, no son compiladas solamente son para usar en extend. Tiene un comportamiento similar a los mixins:

-Cuando usar mixin: bloques de codigo dinamicos.

-Cuano usar @extend con placeholder: bloques de codigo esataticos

|  |  |
| --- | --- |
| %nombreASD{  …..  }  h1{  @extend %nombre !optional;  } | Declaración del placeholder %  Porción de código  hereda las propiedades. !optional para la buena practica |
| %\_nombreASD{  …..  } | En caso de estar en un modulo este se vuelve esta etiqueta se vuelve privada, y solo se puede llamar dentro del modulo |

No se pueden agragar extend a selectores copuestos ej: **h1>h2** solamente a **.h1**