HOJAS DE ESTILO EN CASCADA AVANZADO

Unidad II: SASS y LESS.

Semana 5: Preprocesadores CSS.

Docente: Carlos R. P. Tovar

Universidad Tecnológica del Perú

Inicio

¿Tienen alguna consulta o duda sobre la anterior unidad?

Logro de la Sesión

Al **finalizar** la sesión el estudiante entiende la importancia de un preprocesador y aplica el preprocesador Stylus en el diseño de páginas web.

Universidad Tecnológica del Perú

Utilidad

¿Que conocen sobre Preprocesadores? y CSS?

Transformación
¿Qué es un Preprocesador CSS?

Son herramientas que extienden las capacidades del CSS tradicional.

Permiten usar variables, funciones, anidamiento y más.

Al final, el código se compila a CSS puro para que el navegador lo interprete.

Principales Preprocesadores

Sass

- Un preprocesador CSS ampliamente utilizado, conocido por su sintaxis flexible y poderosa.

Less

- Un preprocesador CSS de sintaxis similar a CSS, fácil de aprender y usar.

Stylus

- Un preprocesador CSS con una sintaxis expresiva y minimalista, que ofrece flexibilidad y control.

¿Por qué usar Preprocesadores?

Reutilización de Código	Las variables y los mixins permiten reutilizar estilos, ahorrando tiempo y esfuerzo.
Mantenimiento Simplificado	El código más organizado facilita la comprensión y el mantenimiento de los estilos.
Mejor Eficiencia	La organización y reutilización del código conducen a una mayor eficiencia en el desarrollo.
Mayor Flexibilidad	Las funciones y la lógica permiten la creación de estilos complejos y adaptables.

Instalación y configuración de un preprocesador

Elección del Preprocesador

Seleccione el preprocesador que se adapte a sus necesidades y preferencias (Sass, Less, Stylus).

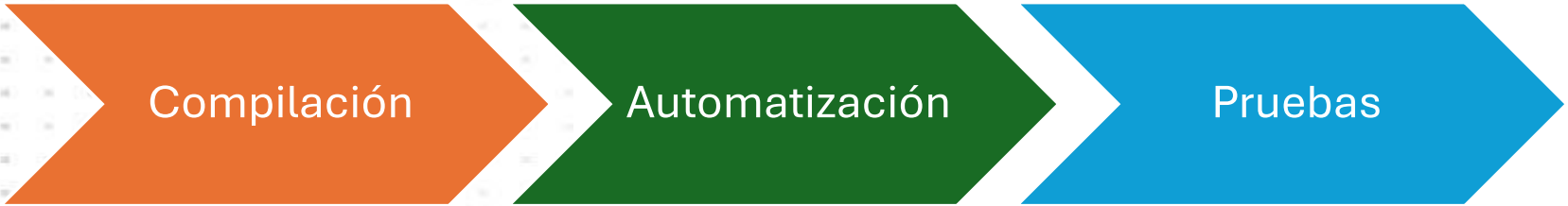
Instalación

Instale el preprocesador utilizando herramientas de gestión de paquetes como npm o yarn.

Configuración

Configure el preprocesador para que funcione correctamente en su entorno de desarrollo.

Compilación y automatización del flujo de trabajo



- Utilizar herramientas como Sass o Less para compilar el código del preprocesador en CSS.
- Integrar la compilación en el flujo de trabajo con herramientas como Gulp o Webpack.
- Verificar el código CSS generado y asegurarse de que funciona correctamente.

Sintaxis básica de un preprocesador

Comentarios

```
// comentario
   de una línea
/* comentario
   de varias
   líneas */
```

Variables

```
$variable:
   valor;
```

Selectores

```
.selector {
   propiedad:
   valor;}
```

Mixins y funciones en preprocesadores

Mixins

- Permiten encapsular un bloque de estilos con un nombre, facilitando la reutilización y la creación de componentes

Funciones

- Permiten encapsular lógica y cálculos, mejorando la modularidad y la legibilidad del código.

Sintaxis de los preprocesadores

Sass	Utiliza una sintaxis similar a CSS con algunas extensiones.
Less	Utiliza una sintaxis más similar a JavaScript, con funciones y expresiones.
Stylus	Utiliza una sintaxis concisa y flexible, con un enfoque en la legibilidad.

Variables en preprocesadores CSS

Declaración

- Se declaran con un nombre y un valor, por ejemplo:
- $color principal: #333;

Uso

- Se utilizan en lugar de valores directos, por ejemplo:
- background color: $color principal;

Ventajas

- Facilitan la modificación de estilos, mejorando la consistencia y la mantenibilidad.

Variables en preprocesadores CSS

Tipo de Variable	Descripción
$variable	Variable global, accesible en todo el código.
@variable	Variable global, accesible en todo el código.

Introducción a Stylus

- Stylus es un preprocesador CSS que permite escribir estilos de forma más flexible, limpia y potente.
- Fue creado con Node.js en 2010 y se integra fácilmente en entornos modernos de desarrollo web.
- Ofrece una sintaxis muy libre: se puede usar con o sin llaves {}, con o sin : y ;.
- Soporta funciones, herencia, mixins, condicionales y bucles.
- Sitio oficial Stylus https://stylus-lang.com/ .

Ventajas de Stylus:

- Sintaxis minimalista y opcional.
- Potente sistema de funciones y mixins.
- Integración fluida con herramientas como Webpack, Gulp, Grunt, etc.

Variables en Stylus

- Almacenan valores para reutilizar en todo el código
- Se definen con el signo =
- No necesitan prefijo como $ (Sass) o @ (Less)

Ventajas:

- Sin caracteres especiales innecesarios
- Fáciles de leer y mantener
- Se pueden redefinir en cualquier momento

```
// Definición
color-primario = #3498db
ancho-base = 960px
fuente-principal = 'Helvetica Neue'

// Uso
.header
  background-color color-primario
  width ancho-base
  font-family fuente-principal
```

Ejemplo básico de Stylus

Stylus

```
$primary-color = #3498db

body
  font-family 'Arial', sans-serif
  background $primary-color

nav
  ul
    margin 0
    padding 0
    list-style none

  li
    display inline-block
    margin-right 20px
```

CSS

```
body {
 font-family: 'Arial', sans-serif;
 background: #3498db;
}
nav ul {
 margin: 0;
 padding: 0;
 list-style: none;
}
nav li {
 display: inline-block;
 margin-right: 20px;
}
```

Mixins en Stylus

¿Qué son los mixins?

- Bloques de código reutilizables
- Pueden aceptar parámetros
- Se llaman como si fueran propiedades CSS

```
// Definición
sombra()
  box-shadow 0 2px 5px rgba(0,0,0,0.3)

// Uso
.card
  sombra()
```

Mixins en Stylus

Mixins con parámetros:

```
// Mixin con parámetros
border-radius(n)
  -webkit-border-radius n
  -moz-border-radius n
  border-radius n

// Uso con valor
.boton
  border-radius(5px)
```

Mixins con parámetros por defecto:

```
sombra(nivel = 1)
  if nivel == 1
    box-shadow 0 1px 3px rgba(0,0,0,0.12)
  else
    box-shadow 0 3px 10px rgba(0,0,0,0.2)
```

Funciones en Stylus

- **Funciones integradas:**

```
// Operaciones con colores
color-claro = lighten(#3498db, 20%)
color-oscuro = darken(#3498db, 20%)

// Operaciones matemáticas
ancho = 100px
doble-ancho = ancho * 2
mitad-ancho = ancho / 2
```

- **Funciones personalizadas:**

```
// Función simple
em(valor, base = 16px)
  (valor / base)em

// Uso
.titulo
  font-size em(24px)  // 1.5em
```

Funciones en Stylus

- **Funciones con lógica condicional:**

```
// Función con condicional
contraste-color(color)
  if (lightness(color) > 50%)
    return #000
  else
    return #fff
```

```
// Uso
.boton
  background-color color-primario
  color contraste-color(color-primario)
```

Operaciones e Interpolación

- **Operaciones matemáticas:**

```
// Cálculos directos
.contenedor
  width (100% - 20px)
  height (600px / 2)

// Con variables
margin-base = 16px
.contenido
  padding margin-base * 2
```

- **Interpolación de variables:**

```
// Interpolación en propiedades
propiedad = width
valor = 100px

selector
  {propiedad} valor

// Interpolación en selectores
prefijo = 'mi'
.{prefijo}-clase
  color red
```

Operaciones e Interpolación

- **Interpolación en strings:**

```
// Strings interpolados
url-path = '/imagenes/'
fondo = 'fondo.jpg'

body
  background-image url(url-path + fondo)
```

Estructuras condicionales:

```
// Condicional if/else
.boton
  if (ancho-base > 1000px)
    padding 20px
  else
    padding 10px

// Operador ternario
.encabezado
  padding (ancho-base > 1000px ? 20px : 10px)
```

Bucles con for/in:

```
// Bucle simple
for i in 1..5
  .col-{i}
    width (20% * i)

// Bucle con condición
margin-values = 0 5 10 15 20
for value in margin-values
  .m-{value}
    margin unit(value, px)
```

Importación y Modularización

Importación de archivos:

```
// main.styl
@import 'variables'
@import 'mixins'
@import 'componentes/botones'
@import 'componentes/forms'

// Importación con globbing (algunas versiones)
@import 'componentes/*'
```

Estructura de archivos:

```
styles/
├── main.styl
├── variables.styl
├── mixins.styl
├── componentes/
│   ├── botones.styl
│   ├── forms.styl
│   └── cards.styl
└── utilidades/
    ├── grid.styl
    └── helpers.styl
```

Organización recomendada:

```
// variables.styl
colores = {
  primario: #3498db,
  secundario: #2ecc71,
  texto: #333
}

// mixins.styl
sombra(nivel = 1)
  // código del mixin

// componentes/botones.styl
.boton
  // estilos de botones
```

Estructura de archivos:

```
styles/
├── main.styl
├── variables.styl
├── mixins.styl
├── componentes/
│   ├── botones.styl
│   ├── forms.styl
│   └── cards.styl
└── utilidades/
    ├── grid.styl
    └── helpers.styl
```

Preprocesamiento (Compilación)

Puedes transformar el archivo .styl en css desde el terminal

// requiere tener Stylus instalado con npm

npm install -g stylus

Compilar un archivo

stylus archivo.styl -o estilo.css

Compilar y minificar

stylus archivo.styl -o estilo.min.css --compress

Compilar con watch

stylus archivo.styl -o estilo.css --watch

Compilar directorio completo

stylus src/ -o dist/ --watch

Plataformas para procesar Stylus online:

1. Stylus Lang Playground (oficial)
   - Editor interactivo oficial de Stylus.
   - Puedes escribir código .styl a la izquierda y ver el CSS compilado a la derecha.
   - Ideal para probar rápidamente sintaxis y ver resultados.
2. CodePen
   - Permite usar Stylus desde las opciones de configuración del CSS.
   - Ve a Settings > CSS > CSS Preprocessor > Stylus.
   - Ideal para pequeños experimentos y compartir ejemplos.
3. JSFiddle (requiere configuración manual)
   - Aunque no tiene soporte nativo para Stylus, puedes embeber el CSS compilado desde otras herramientas.

CIERRE
Recursos Adiconales (VSCODE)

1. **Manta's Stylus Supremacy:** VSCODE plugin para dar formato al archivo Stylus
2. **npm Intellisense:** VSCODE plugin que autocomplete módulos npm y sentencias de importación
3. **Stylus de sysoev :** VSCODE plugin que da Soporte del lenguaje Stylus

Configuraciones en VSCODE

```
{
  "files.associations": {
    "*.styl": "stylus"
  },
  "editor.formatOnSave": true,
  // Configuración específica para Stylus
  "stylusSupremacy.insertNewLineAroundImports": true,
  "stylusSupremacy.insertNewLineAroundBlocks": true,
  "languageStylus.useSeparator": true,
  "languageStylus.useBuiltinFunctions": true
}
```

Universidad
Tecnológica
del Perú