SOFTWARE DEVELOPER

Fondamenti di UX/UI Design e HTML CSS

Docente: Loredana Frontino

Titolo argomento: SASS

Cos’è?

Cos’è SASS (Syntactically Awesome Style Sheets)

Un CSS con i superpoteri!
È un preprocessore CSS che estende funzionalità del CSS standard e poi viene compilato in CSS standard.

Sass offre funzionalità aggiuntive al CSS , come l'annidamento, i mixin, l'ereditarietà, funzioni, operatori e altre utili funzionalità che aiutano a scrivere CSS robusti e gestibili.

Perché utilizzarlo

Migliore organizzazione e manutenibilità del codice: Le funzionalità come nesting e partials aiutano a strutturare meglio il CSS, rendendolo più facile da leggere e gestire, soprattutto in progetti di grandi dimensioni.

Riutilizzabilità del codice: Variabili e mixin permettono di evitare la duplicazione di stili, rendendo il codice più DRY (Don't Repeat Yourself).

Maggiore flessibilità ed estendibilità: Le funzioni, gli operatori e le direttive offrono maggiore potenza espressiva rispetto al CSS standard.

Flussi di lavoro più efficienti: L'uso di preprocessori può integrarsi bene con strumenti di build e automatizzare compiti come la compilazione e la minificazione del CSS.

Come integrarlo e utilizzarlo

Si può utlizzare il file con estensione .scss

Install directly

1. Download Sass from the Repository based on your operating system
2. Add it to your PATH

Install with npm from Node.js

npm install -g sass

Run Sass

sass input.scss output.css

OR

sass --watch input.scss output.css

Scopriamo la sintassi

Commenti

Come per il normale css vengono settati con
// per i single line
/* per i multiline */

```
// SCSS comment in one line
.class {}

/* SCSS comment
Multinline */
.class {}
```

Variabili in CSS

Nel CSS vengono definite attraverso la :root

```
/* define once */
:root {
  --primary-color: #007bff;
}
/* reuse many times */
.button {
  background-color: var(--primary-color);
}
```

Variabili in SCSS

Si definiscono con $ e permettono di definire le proprietà condivise principali per la definizione di un'interfaccia.

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

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

Nesting

Scrivendo HTML esiste una gerarchia nidificata e con il normale CSS questa non viene riprodotta, ma viene sfruttata invce in SASS dove è possibile annidare anche il CSS.

.SCSS

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

.CSS

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

Nesting - Parents

Per gestire le relazioni tra gli elementi dello stesso livello si applica & per agganciare il parametro. Come se fosse il parametro div.warning in css

.SCSS

```scss
.warning {
    background-color: red;
    &:hover {
      background-color: orange;
    }
    &--urgent {
      color: purple;
    }
    #footer & {
      // a warning in the footer looks different
      background-color: plum;
    }
    & > & {
      // a warning in a warning
      border: 1px dotted black;
    }
}
```

```html
<div class="warning">careful</div>
<div class="warning--urgent">please be
 careful</div>
<div class="warning">
 <span>some error caused</span><span
 class="warning">another error</span>
</div>
<div id="footer">
 <div>some footer text</div>
 <div class="warning">footer warning</div>
</div>
```

Nesting - Parents

Per gestire le relazioni tra gli elementi dello stesso livello si applica & per agganciare il parametro. Come se fosse il parametro div.warning in css

.SCSS

```scss
.warning {
    background-color: red;
    &:hover {
      background-color: orange;
    }
    &--urgent {
      color: purple;
    }
    #footer & {
      // a warning in the footer looks different
      background-color: plum;
    }
    & > & {
      // a warning in a warning
      border: 1px dotted black;
    }
}
```

.CSS

```css
.warning {
  background-color: red;
}
.warning:hover {
  background-color: orange;
}
.warning--urgent {
  color: purple;
}
#footer .warning {
  background-color: plum;
}
.warning > .warning {
  border: 1px dotted black;
}
```

Ereditarietà - @extend

@extend permette di condividere proprietà definite in un selettore in altri selettori, estendendo le proprietà definite per il selettore iniziale.

.scss

```scss
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.message {
  @extend %message-shared;
}
.success {
  @extend %message-shared;
  border-color: green;
}
```

.css

```css
.warning, .error, .success, .message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}
```

Funzioni - @mixin

@mixin permettono di creare un gruppo di dichiarazioni CSS, con all'interno anche elementi variabili che possono essere inseriti per rendere ancora più compatto il codice.

.scss

```scss
@mixin theme($theme: DarkGray) {
  background: $theme;
  box-shadow: 0 0 1px rgba($theme, .25);
  color: #fff;
}

.info {
  @include theme;
}
.alert {
  @include theme($theme: DarkRed);
}
```

.css

```css
.info {
  background: DarkGray;
  box-shadow: 0 0 1px rgba(169, 169, 169, 0.25);
  color: #fff;
}

.alert {
  background: DarkRed;
  box-shadow: 0 0 1px rgba(139, 0, 0, 0.25);
  color: #fff;
}
```

@mixin o @extend ?

@mixin

1. Il codice CSS compilato non è DRY;
2. lo stesso CSS viene ripetuto per ogni classe
3. Il file CSS generato è più grande
4. Flessibile: accettano argomenti variabili

@extend

1. Non flessibile: non accetta argomenti
2. Codice compilato DRY
3. Crea relazioni semantiche tra i selettori
4. Accoppia i selettori insieme

Operators

Fare calcoli matematici nel CSS è molto utile.
Sass ha una serie di operatori matematici standard come +, -, *, math.div(), e %.

.SCSS

```scss
@use "sass:math";

.container { display: flex; }

article[role="main"] {
  width: math.div(600px, 960px) * 100%;
}

aside[role="complementary"] {
  width: math.div(300px, 960px) * 100%;
  margin-left: auto;
}
```

.CSS

```css
.container { display: flex; }

article[role=main] {
  width: 62.5%;
}

aside[role=complementary] {
  width: 31.25%;
  margin-left: auto;
}
```

Moduli @use

la regola @use introduce un sistema di moduli per importare e gestire fogli di stile in modo più strutturato ed efficiente rispetto al tradizionale @import.

.scss

```scss
// _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

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

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

Esercitazioni

Esercizio 1

Creare una pagina HTML con una serie di 6 bottoni e 5 input di testo posizionati in serie a cui si vuole dare uno stile per definire un design system HTML

Definire le variabili: - 6 variabili colore (primary, secondary, tertiary, success, error, dark)

- 1 variabile per il font
- 2 variabili per i border-radius dei bottoni e per quelli degli input

Creare 2 mixin: uno che gestisca le proprietà comuni dei bottoni (che si differenzieranno per colore testo, background color e border color) e l'altro che gestisca le proprietà comuni degli input (che si differenzieranno per spessore dei bordi e colore dei testi)

Inserire anche le possibili variazioni che ci possono essere quando con il pulsante si va in hover e per l'input quando è in stato focus

ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Esercitazioni

Esercizio 1

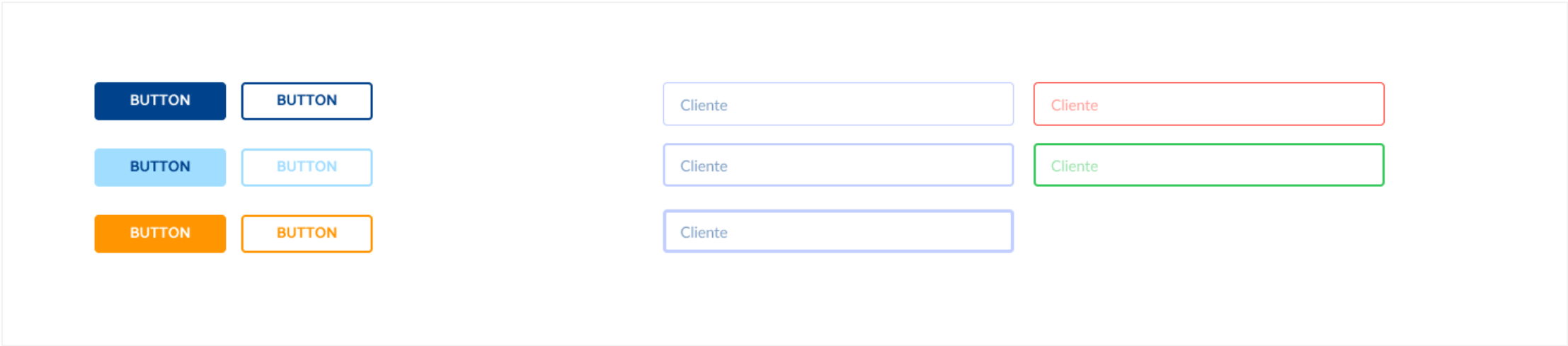
Associare ai primi tre bottoni creati delle classi che applichino color-primary al primo bottone, color secondary al secondo e color tertiary al terzo bottone

Associare ai primi tre input delle classi che applichino uno spessore di 1px al primo e colore dark del testo, spessore 2px al secondo e colore dark del testo, spessore 3px al terzo e colore primay del testo

ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Esercitazioni

Esercizio 1



Flow control rules

Sass fornisce diverse regole di flusso che consentono di controllare se gli stili vengono emessi o di emetterli più volte con piccole variazioni.

Possono anche essere utilizzate in mixin e funzioni per scrivere piccoli algoritmi che semplificano la scrittura di Sass.

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Questa coppia di regole controlla se il blocco di codice scritto verrà valutato o meno, inserendo una condizione che restituisce true o false.

```
$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);
  }
}
```

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Permette di ripetere un blocco di codice CSS per un numero specifico di volte. È utile per generare stili ripetitivi con piccole variazioni, come griglie, animazioni o varianti di colori.

Ci sono due forme principali del ciclo @for:

1. @for $var from <start> through <end>
2. @for $var from <start> to <end>

Flow control rules – regole supportate

@if - @else

@for

@each

@while

@for $var from <start> through <end>

Con questa regola si vanno ad analizzare gli elementi del ciclo a partire da start e chiudendo con end compreso.

```
@for $i from 1 through 5 {
    .col-#{$i} {
        width: (100% / 5) * $i;
    }
}
```

```
.col-1 {
  width: 20%;
}
.col-2 {
  width: 40%;
}
.col-3 {
  width: 60%;
}
.col-4 {
  width: 80%;
}
.col-5 {
  width: 100%;
}
```

Flow control rules – regole supportate

@if - @else

@for

@each

@while

@for $var from <start> to <end>

Con questa regola si vanno ad analizzare gli elementi del ciclo a partire da start e proseguendo fino all'elemento precendente al valore di end.

```
@for $i from 1 to 3 {
    .item-#{$i} {
        margin-left: $i * 10px;
    }
}
```

```
.item-0 {
  margin-left: 0px;
}
.item-1 {
  margin-left: 10px;
}
.item-2 {
  margin-left: 20px;
}
```

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Permette di iterare su elementi all'interno di una lista o di una mappa (associazione chiave-valore), applicando lo stesso blocco di codice a ciascun elemento. È ideale per generare stili basati su una serie di valori predefiniti.

Iterazione su lista:

```
$colors: (primary, secondary, accent);

@each $color in $colors {
  .button--#{$color} {
    // Supponendo che esista una funzione get-color()
    background-color: get-color($color);
    color: white;
  }
}
```

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Permette di iterare su elementi all'interno di una lista o di una mappa (associazione chiave-valore), applicando lo stesso blocco di codice a ciascun elemento. È ideale per generare stili basati su una serie di valori predefiniti.

Iterazione su mappa:

```
$font-sizes: (
  small: 14px,
  medium: 16px,
  large: 18px
);

@each $size-name, $size-value in $font-sizes {
  .text-#{$size-name} {
    font-size: $size-value;
  }
}
```

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Permette di eseguire ripetutamente un blocco di codice CSS finché una determinata condizione rimane vera. A differenza di @for (che itera un numero specifico di volte) e @each (che itera su elementi di una collezione), @while continua finché la sua espressione di condizione restituisce true.

!!! È cruciale assicurarsi che all'interno del blocco di codice ci sia una logica che, ad un certo punto, modifichi le variabili coinvolte nella <condizione> in modo che essa diventi false. Se la condizione rimane sempre true, il ciclo continuerà all'infinito, potenzialmente causando problemi di compilazione o di performance. !!!

Flow control rules – regole supportate

@if - @else

@for

@each

@while

Permette di eseguire ripetutamente un blocco di codice CSS finché una determinata condizione rimane vera.

```
$i: 1;

@while $i <= 5 {
  .margin-#{$i} {
    margin: $i * 5px;
  }
  // incremento variabile per far diventare la condizione falsa
  $i: $i + 1;
}
```

```
.margin-1 {
  margin: 5px;
}
.margin-2 {
  margin: 10px;
}
.margin-3 {
  margin: 15px;
}
.margin-4 {
  margin: 20px;
}
.margin-5 {
  margin: 25px;
}
```

Esercitazioni

Esercizio 2

Proseguendo sull'esercizio 1

Aggiungere al mixin dei bottoni una condizione che permetta di gestire I bottoni quando sono di tipo outline (con background color transparent) e gestire le varie casistiche di hover in relazione a questo fattore.

Provare a gestire l'assegnazione delle classi che gestiscono I colori dei bottoni e quelli degli input con un @each

Link utili

SASS

SASS Flow control

Install sass

Placeholder selectors

Intro to sass, scss and less

LESS