

# FinTech Software Developer

Programmazione WEB - HTML | CSS | Javascript

Docente: Shadi Lahham

# CSS Rules

Presentation & look

Shadi Lahham - Web development

# Intro to CSS

# Anatomy of a Website

## **Content**

Text, Media

## **HTML**

Structure

## **CSS**

Presentation

## **Javascript**

Logic/Interactivity

# What is CSS?

## Cascading Style Sheets

- CSS is a "style sheet language" that lets you style the elements on your page
- CSS works in conjunction with HTML, but is not part of HTML itself

# Anatomy of CSS

- CSS consists of style rules
- A block of CSS code is a **rule**
- Each style rule consists of a selector and declarations of property-value pairs
- A property-value pair is a **declaration**

```
selector {  
  property: value;  
  property: value;  
}
```

Example:

```
body {  
  color: yellow;  
  background-color: black;  
}
```

# Applying CSS to HTML

There are 3 ways to apply CSS styles

- Inline
- Embedded
- External

# Inline CSS

```
<body>  
  <p style="color:red;text-align:center;">Hello</p>  
  <p style="color:orange;text-align:center;">Nice to meet you</p>  
</body>
```

- no separation of concerns
- no reusability since it applies to a single element only
- limited caching, larger HTML file and slower load times
- no selectors or media queries
- hard to read and maintain code

**Never use inline CSS**



# Embedded CSS

```
<head>
  <style>
    p {
      color: red;
      text-align: center;
    }
  </style>
</head>
```

- no separation of concerns
- limited reusability; single HTML file only
- limited caching, larger HTML file and slower load times
- hard to read and maintain code

**Never use embedded CSS**

# External CSS

**index.html**

```
<head>  
  <link rel="stylesheet" href="./css/style.css">  
</head>
```

**style.css**

```
p {  
  color: red;  
}
```

- good separation of concerns
- reusable and modular
- browser caching benefits and faster load times
- easy to maintain and collaborate

**Always use an external CSS**

# Selectors

# Selectors

The selector is used to select which elements in the HTML page will be given the styles inside the curly braces

```
selector {  
  property: value;  
  property: value;  
}
```

# Selector: Element

*/\* Selects all paragraph elements. \*/*

```
p {  
  property: value;  
}
```

*/\* Selects all image elements. \*/*

```
img {  
  property: value;  
}
```

# Selector: Relational

```
/* Selects all em elements that are within a paragraph. */
```

```
p em {  
  color: yellow;  
}
```

```
<!-- This would be selected -->
```

```
<p>This is <em>important.</em></p>
```

```
<!-- This would not! -->
```

```
<h1>This is <em>important.</em></h1>
```

- Position selectors are more specific
- They look for elements inside other elements
- We separate nested elements with a space

# Selector: Relational

```
/* the css */  
ul li a strong {  
  color: purple;  
}
```

```
<!-- the html -->  
<ul>  
  <li>  
    <a href="programs.html">Our <strong>program</strong></a>  
  </li>  
</ul>
```

# Reusing Code

Don't Repeat Yourself (DRY) principle:

“every piece of knowledge must have a single, unambiguous, authoritative representation within a system”

Recognizing duplication and eliminating it through abstraction produces cleaner code than unnecessary repetition (copy paste)

To reuse CSS, we use IDs and classes



# IDs vs. Classes

## ID

- Should only apply to one element on a page
- For example, a page has has one footer
- Uses the symbol #

## Class

- Many elements can have the same class
- There can be many warnings on one webpage
- Uses the symbol .

# Selector: ID

```
/* Selects the one element on the page with an id of site-footer */  
#site-footer {  
  property: value;  
}
```

```
<!-- the html -->  
<p id="site-footer">Copyright message</p>
```

# Selector: Class

```
/* Selects all elements with a class of warning. */  
.warning {  
  color: red;  
}
```

```
<!-- the html -->  
<p class="warning">Run away!</p>
```

```
<div class="warning">  
  this is also a warning  
</div>
```

```
<ul>  
  <li>  
    <p class="warning">Danger</p>  
  </li>  
</ul>
```

# Grouping Selectors

```
h3, .message, #notificationArea {  
    color: Maroon;  
}
```

*/\* or \*/*

```
h3,  
.message,  
#notificationArea {  
    color: Maroon;  
}
```

CSS properties

# Property: Color

```
/* The color property changes the color of the text */  
p {  
  color: red;  
  color: #ff0000;  
  color: rgb(255, 0, 0);  
}
```

# Property: Background-color

```
/* The background-color property changes the color of the background */  
p {  
  background-color: black;  
  background-color: #000000;  
  background-color: rgb(0, 0, 0);  
}
```

# CSS Color Values

Browsers can accept colors in many different ways

Color name	red
Hexadecimal value	#FF0000 #FF0000FF
RGB value	rgb(255, 0, 0) rgba(255, 0, 0,1)
HSL value	hsl(0, 100%, 50%) hsla(0, 100%, 50%,1)

[HTML Color Picker](#)

[147 CSS Color Names](#)

[Chrome devtools color-picker](#)

[216 Web Safe Colors](#)



# Property: Width

- Sets the width of a block-level element or `img`
- Doesn't work for inline elements (unless their `display` property is changed)
- Accepts a variety of length units

```
#sidebar {  
  width: 200px;  
  width: 20em; /* relative to font size */  
  width: 20%; /* relative to containing element width */  
  width: 20vw; /* relative to viewport: 1vw = 1% viewport width */  
}
```

A list of all CSS length units

[The Lengths of CSS](#)

[CSS Units](#)

The most used are: `px`, `rem`, `em`, `vw`, `vh`, `%` (percentage)

# Property: Font-family

```
p {  
  /* Specific font name */  
  font-family: "Times New Roman";  
  
  /* Generic name */  
  font-family: serif;  
  
  /* Comma-separated list */  
  font-family: "Arial", sans-serif;  
}
```

The font-family property defines which font is used  
When listing multiple fonts, always list a generic name last  
[Fallback Font Stacks](#)

Web-safe fonts are pre-installed by many operating systems  
Not all systems have the same fonts, but web-safe font stacks contain fonts that look similar  
[CSS Web Safe Fonts](#)

# Custom fonts: @font-face

```
@font-face {  
  font-family: 'MyFontName';  
  src: url('fontFile.eot'); /* IE9 */  
  src: url('fontFile.eot?#iefix') format('embedded-opentype'), /* IE6-IE8 */  
        url('fontFile.woff2') format('woff2'), /* Very Modern Browsers */  
        url('fontFile.woff') format('woff'), /* Modern Browsers */  
        url('fontFile.ttf') format('truetype'), /* Safari, Android, iOS */  
        url('fontFile.svg#svgFontName') format('svg'); /* Old iOS */  
}
```

```
body {  
  font-family: 'MyFontName', sans-serif;  
}
```

**Careful:** using custom fonts makes your page slower

[@font-face | MDN](#)

[CSS @font-face Rule](#)

# Google web fonts

```
<!-- the html -->
<head>
  <!-- rest of head -->
  <link href="https://fonts.googleapis.com/css?family=Trade+Winds&display=swap" rel="stylesheet">
  <!-- rest of head -->
</head>

/* the css */
p {
  font-family: 'Trade Winds';
}
```

**Careful:** using webfonts, such as google fonts, makes your page slower  
Use with moderation

# Property: Font-size

*/\* The font-size property specifies the size of the font. \*/*

```
p {  
  /* Pixels */  
  font-size: 12px;  
  
  /* em */  
  font-size: 1.5em;  
  
  /* Percentage */  
  font-size: 100%;  
}
```

# Property: Fonts - a shorthand property

```
p {  
  font-style: italic;  
  font-weight: bold;  
  font-size: 10px;  
  font-family: sans-serif;  
}
```

*/\* or using a shorthand property set the values of multiple CSS properties \*/*

```
p {  
  font: italic bold 10px sans-serif;  
}
```

# Typography for programmers

## Font Stack

A list of fonts, ordered for browser use. Include fallback fonts for compatibility

## Font Size

Determines text character height, in px, em, rem for scalability and accessibility

## Line Height

Vertical space between text lines (leading), crucial for readability

Recommended to be 1.5 to 1.6 times font size

## Font Weight

Thickness or boldness of characters

normal, bold, or numeric values (100 to 900)

# Typography for programmers

## Font Style

Specifies text as `italic` or `normal`

## Text Alignment

Horizontal alignment of text within its element

`left`, `right`, `center`, or `justified`

## Color

Specifies text color

color names, hexadecimal codes, RGB, or HSL values

## Text Decoration

Adds decorations like `underline`, `overline`, or `line-through` to text



# Typography for programmers

```
.main {  
  font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;  
  font-size: 16px;  
  line-height: 1.5;  
  font-weight: bold;  
  font-style: italic;  
  text-align: center;  
  color: #48c06c;  
  text-decoration: underline;  
}
```

# More CSS Properties

Many CSS properties have self-explanatory names:

- background-color
- font-family
- font-size
- color
- width
- height

Most common CSS properties

[CSS Properties Reference](#)

Complete reference

[CSS reference](#)

Check browser compatibility before using properties

[Can I use...](#)

# CSS Cascade

# CSS Cascade

```
p {  
  color: orange;  
  font-family: sans-serif;  
}
```

```
.info-paragraph {  
  color: blue;  
  background-color: orange;  
}
```

```
#main-paragraph {  
  font-weight: bold;  
  color: green;  
}
```

```
<p>Paragraph</p>
```

```
<p class="info-paragraph">Paragraph</p>
```

```
<p class="info-paragraph" id="main-paragraph">Paragraph</p>
```

# Cascading priority: Importance

The browser assigns different priorities to CSS depending on the type of selector

1. Inline CSS - Most Important
2. ID selector
3. Class selector
4. Element selector - Least Important

# Cascading priority: Specificity

Your browser also assigns priority based on the specificity of the selection  
More specific selectors have higher priority

```
/* Most specific */  
.main .sale .clearance p {  
  color: red;  
}
```

```
.header .title p {  
  color: green;  
}
```

```
/* Least specific */  
.footer p {  
  color: blue;  
}
```

# Cascading priority: Source order

The tie-breaker is rule order

Rules lower in the file overwrite rules higher in the file

```
a {  
  background-color: yellow;  
}
```

```
a {  
  background-color: teal;  
}
```

```
/* This rule wins */
```

```
a {  
  background-color: black;  
}
```

# Cascading priority: Specificity example

```
<!-- the html -->
<div class="main">
  <p>What color am I?</p>
  <div class="sale">
    <p>What color am I?</p>
    <div class="clearance">
      <p>What color am I?</p>
    </div>
  </div>
</div>
```

```
/* the css */
.main .sale .clearance p {
  color: red;
}

.main .sale p {
  color: orange;
}

.main p {
  color: lime;
}
```



# Cascading priority: !important

The `!important` declaration overrides any other declarations

Using it is a very **bad practice** because it makes debugging more difficult by breaking the natural cascading in stylesheets

Only use `!important` when:

- You need to override foreign CSS (e.g. from a library)
- You need to override inline styles

```
<!-- the html -->  
<div class="simple" style="color: red;">What color am I?</div>
```

```
/* the css */  
.simple {  
  color: blue !important;  
}
```

# Cascading priority: !important is dangerous

```
<!-- the html -->
<div class="main">
  <p>What color am I?</p>
  <div class="sale">
    <p>What color am I?</p>
    <div class="clearance">
      <p>What color am I?</p>
    </div>
  </div>
</div>
```

```
/* the css */

p {
  color: pink!important;
}

.main .sale .clearance p {
  color: red;
}

.main .sale p {
  color: orange;
}

.main p {
  color: lime;
}
```

# Custom properties

# CSS Custom Properties

enhance maintainability and flexibility in styling

- reusable: define once, use across multiple elements
- dynamic: changes properties with JavaScript at runtime
- fallback: value to use when a custom properties is not defined
- cascade and inheritance: follow CSS rules

**syntax:**

use `--` followed by a name

```
:root {  
  --primary-color: #007bff;  
}
```

# Reusable & Fallback

```
/* define once */
:root {
  --primary-color: #007bff;
}

/* reuse many times */
.button {
  background-color: var(--primary-color);
}

.special {
  color: var(--primary-color, orange); /* fallback if custom prop not defined */
}
```

# Dynamic

*// change a custom property value using Javascript*

```
document.documentElement.style.setProperty('--primary-color', '#ff0000');
```

*// example, change primary color after 2 seconds*

```
setTimeout(() => {
```

```
  document.documentElement.style.setProperty('--primary-color', '#ffaadd');
```

```
}, 2000);
```

# Cascade and inheritance

index.html

```
<div class="container">
  <div class="box"></div>
</div>
```

style.css

```
:root {
  --primary-color: #7cc22d;
}

.container {
  --primary-color: #f1c120;
}

.box {
  background-color: var(--primary-color); /* inherits color from .container */
}
```

# :root

The **:root** pseudo-class selector represents the highest-level "root" element of the document, typically the `<html>` element. It is often used to define CSS custom properties, providing consistency and flexibility in styling

## Consistency:

- abstracts away specific HTML elements
- a convention followed by developers and frameworks
- ensures uniformity across projects regardless of HTML structure or naming conventions
- enhances readability

## Flexibility:

- sometimes the root element might differ from `<html>`
- e.g. XML-based documents or shadow DOM



# Pseudo-classes



# Pseudo-classes

- recognized by the single colon prefix (:)
- work like regular HTML classes but aren't written in the source code
- change based on what users do or how the document is structured
- style elements based on their current state, position within the document or their content

```
selector:pseudo-class {  
  property: value;  
}
```

## Example

```
a:hover {  
  color: pink;  
  text-decoration: none;  
}
```

Complete list here:

[Pseudo-classes - CSS: Cascading Style Sheets](#)

# Pseudo-classes

```
/* unvisited link */  
a:link {  
  color: #ff0000;  
}
```

```
/* visited link */  
a:visited {  
  color: green;  
}
```

```
/* moused over */  
a:hover {  
  color: purple;  
}
```

To be effective, `a:hover` **must** come after `a:link` and `a:visited`

```
/* selected with keyboard*/  
a:focus {  
  color: purple;  
}
```

```
/* activated link */  
a:active {  
  color: blue;  
}
```

To be effective, `a:active` **must** come after `a:hover`

# CSS reset & normalize

# Why CSS resets are needed

- Each browser varies in how it displays web pages
- Browsers define different default styles, so you never start from the same blank slate
- CSS reset style sheets are used to normalize the default CSS across browsers

There are two main approaches:

- Reset
- Normalize

# CSS reset

- Removes every default style.
- Remove all built-in browser styling
- Standard elements like H1-6, p, strong, em, etc. end will look exactly similar without any styling
- The developer is supposed to add any styling from scratch

[CSS Tools: Reset CSS](#)

# CSS reset

## HTML5 Test Page

This is a test page filled with common HTML elements to be used to provide visual feedback whilst building CSS systems and frameworks.

[Text](#)

[Headings](#)

[Paragraphs](#)

[Blockquotes](#)

[Text](#)

[Headings](#)

[Heading 1](#)

[Heading 2](#)

[Heading 3](#)

[Heading 4](#)

[Heading 5](#)

[Heading 6](#)

[\[Top\]](#)

[Paragraphs](#)

A paragraph (from the Greek paragraphos, “to write beside” or “written beside”) is a self-contained unit of a discourse in writing dealing with a particular point or idea. A paragraph consists of one or more sentences. Though not required by the syntax of any language, paragraphs are usually an expected part of formal writing, used to organize longer prose.

[\[Top\]](#)

[Address](#)

[Contact the Author here](#)

[test@test.com](#)

[\[Top\]](#)

[Blockquotes](#)

A block quotation (also known as a long quotation or extract) is a quotation in a written document, that is set off from the main text as a paragraph, or block of text.

It is typically distinguished visually using indentation and a different typeface or smaller size quotation. It may or may not include a citation, usually placed at the bottom.

[Said no one, ever.](#)

[\[Top\]](#)

[Lists](#)

# CSS normalize

- Aims to make built-in browser styling consistent across browsers
- Elements like H1-6 will appear bold, larger, etc. in a consistent way across browsers
- The developer is supposed to add additional styling where required

[Normalize.css](#)



# CSS normalize

## HTML5 Test Page

This is a test page filled with common HTML elements to be used to provide visual feedback whilst building CSS systems and frameworks.

- [Text](#)
  - [Headings](#)
  - [Paragraphs](#)
  - [Blockquotes](#)

### Text

### Headings

#### Heading 1

#### Heading 2

#### Heading 3

# Reset or normalize?

**Normalize** has some advantages

- Preserves useful defaults
- Corrects common bugs
- Doesn't clutter dev tools
- Modular
- Better documentation

**Answer:** depends on the project. It might need reset, normalize or parts of both

There are also other approaches such as [Destyle.css](#)

Your turn

# 1.Simple styling

- Create an HTML file with some headings, paragraphs, lists and other elements
- Create three folders called: inline, embedded, external
- In each folder copy the HTML file that you created
- For the first folder use inline styling, for the second embedded, and use an external css file for the third
- Use at least the following style changes
  - Change the size of a text
  - Change the color
  - Change the background color of one or more elements
  - Change the font

## 2.Simple selecting

- Create an HTML file with some headings, paragraphs, lists and other elements
- Style the page using at least the following style changes
  - Change the size of a text
  - Change the color
  - Change the background color of one or more elements
  - Change the font
- In your CSS use at least one example of the following selectors
  - Element selector
  - Relational selector
  - ID selector
  - Class selector

## 3.Font mania

- Create an HTML file with some headings, paragraphs, lists and links
- Style the page using colors and fonts
  - Links not inside lists and paragraphs should be red
  - Links inside lists should have a web-safe font and should not be red
  - Links inside paragraphs should have a google font and should not be red
  - Add a CSS rule to style your links using pseudo-classes
  - Group selectors for your links and other elements **(DRY)**

### **Bonus:**

- Try to use many google fonts in a page and calculate the impact on the page loading time. Present your findings in a Doc file

## 4.The great reset

- Create the following structure
  - Create 3 folders named: *test-reset*, *test-normalize*, *test-destyle*
  - Download *reset.css*, *normalize.css* and *destyle.css* and put them in */style* in each of the folders
  - Create an *index.html* file with the HTML tags that you know, especially headers, paragraphs, images, lists, tables, forms and inputs
  - Copy *index.html* in each folder
  - Write a */style/style.css* file for each folder
  - In style.css apply styling to your HTML using many different properties [CSS Properties Reference](#)

Continues on next page >>>

## 4.The great reset

- Result
  - Each style.css is different, because of the different resets, but
    - the result in the browser should look exactly the same for all 3 folders
    - The result should also look the same in different browsers *Chrome, Firefox, Edge, Safari*
- Report
  - Create a .txt or .doc or .md file in which you explain which method, reset/normalize/destyle, is easier to work with based on:
    - The length of the CSS that you had to write
    - The number of CSS rules that you had to override



# References

Validate your HTML:

[The W3C Markup Validation Service](#)

Validate your CSS:

[The W3C CSS Validation Service](#)

Check browser compatibility:

[Can I use... Support tables for HTML5, CSS3, etc](#)

# References

## **Reset or normalize**

[The Opinionated Decision on CSS Resets](#)

[Normalize CSS or CSS Reset?!](#)

[About normalize.css](#)

## **In-depth reading about CSS resets**

[A tale of CSS Resets and Everything You Need to Know About Them](#)