Essential CSS

Contents

1. CSS terminology
2. Defining simple selectors
3. Defining relational selectors
4. Defining attribute selectors
5. Chaining and grouping selectors
6. Pseudo-classes and pseudo-elements

Demos folder:
Demos\A-CssEssentials

1. CSS Terminology

- Style rules
- Declarations
- Selectors

Style Rules

- A CSS style sheet is a collection of style rules
  - For example, this style sheet comprises two style rules:

```
div {
  color: red;
}

p {
  color: blue;
}
```

Declarations

- Each style rule has a declaration block (i.e. the {} curly brackets)
  - The declaration block contains a series of declarations
  - Each declaration is terminated by a semi-colon

- Each declaration defines:
  - A property:value pair
  - Properties are things like `color`, `font-family`, and `width`
  - Values are things like `red`, `Tahoma`, and `30%`

```
div {
  color: red;
  font-family: Tahoma, Arial, sans-serif;
  width: 30%;
}
```

Selectors

- Each style rule specifies a selector
  - The selector defines which parts of the HTML document will be affected by the declarations
- There are several types of selector:
  - Element selectors
  - Class selectors
  - ID selectors

2. Defining Simple Selectors

- Selecting all elements
- Element selectors
- Class selectors
- Id selectors
- Combining selectors

Selecting All Elements

- You can define a rule that selects all elements

```
* {
  color: red;
}
```


Applies to all elements

Element Selectors

- Element selectors are the simplest kind of selector
  - Also known as tag selectors
  - Select all elements in the document that have a particular tag name

- The example at the start of this chapter used element selectors
  - See `ElementSelectors.html`

```
div {
  color: red;
}
```

← Applies to all elements

```
p {
  color: blue;
}
```

← Applies to all <p> elements

Class Selectors

- Class selectors apply to all elements that have a specified CSS class
  - Useful for applying the same style to different kinds of element

- Example
  - See `ClassSelectors.html`

```
.optional {
  background-color: #eeeeff;
}
```

← Applies to all elements whose class is "optional"

```
.required {
  background-color: #ffe0e0;
}
```

← Applies to all elements whose class is "required"

ID Selectors

- ID selectors apply to an element that has a specified id attribute
  - Useful for targeting a specific , for example

- Example
  - See `IdSelectors.html`

```
#mainContent {
  color: blue;
  background-color: #eeeeff;
}
```

← Applies to the element whose id is "mainContent"

```
#additionalContent {
  color: red;
  background-color: #ffeeee;
}
```

← Applies to the element whose id is "additionalContent"

Combining Selectors

- You can combine element/class/id selectors
  - Use this syntax:

```
anElementName#anId.aClassName {
  declarations…
}
```

- Example
  - See `CombiningSelectors.html`

```
.standout {
  color: orange;
  font-size: 18pt;
  font-weight: bold
}
```

← Applies to general elements whose class is "standout"

```
h1.standout {
  font-size: 36pt;
}
```

← Applies to <h1> elements whose class is "standout"

3. Defining Positional Selectors

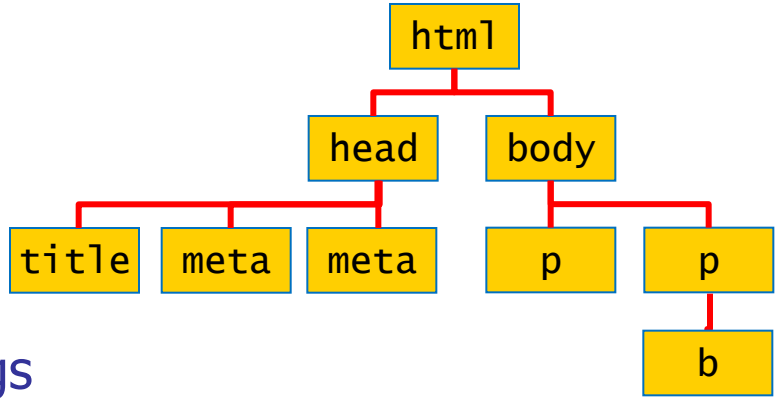
- HTML document structure
- Descendent selectors
- Child selectors
- Adjacent sibling selectors

HTML Document Structure

- Consider this simple HTML document

```
<html>
    <head>
        <title>This is my simple page</title>
        <meta name="author" content="John Smith"/>
        <meta name="description" content="Simple example"/>
    </head>
    <body>
        <p id="intro">This is the intro to my page</p>
        <p id="main">This is the <b>main</b> content in my document</p>
    </body>
</html>
```

- The browser parses this HTML and creates a "DOM" tree
  - You can define CSS rules that target descendants, child elements, and adjacent siblings



Descendant Selectors

- You can define a style rule that targets elements that are a descendant of an element
  - Also known as contextual selectors
  - Use this syntax:

```
outerElement descendantElement {
  declarations…
}
```

- Example
  - See `DescendantSelectors.html`
  - Discuss the rules, and describe the outcome

Child Selectors

- You can define a style rule that targets elements that are a direct child of an element
  - Use this syntax:

```
parentElement > childElement {
  declarations…
}
```

- Example
  - See `ChildSelectors.html`
  - Discuss the rules again, and describe the outcome

Adjacent Sibling Selectors

- You can define a style rule that targets elements that are siblings of an element (i.e. same parent)
  - Use this syntax:

```
firstElement + adjacentSiblingElement {
  declarations…
}
```

- Example
  - See `AdjacentSiblingsSelectors.html`
  - Discuss the rules again, and describe the outcome

4. Defining Attribute Selectors

- Recap of attributes
- Overview of attribute selectors
- Testing for attribute presence
- Testing for an attribute value

Recap of Attributes

- Some HTML elements can have attributes
  - For example:

```
<table>
  <tr>
    <td colspan="3">
    …

<input type="text" size="20" … />
<input type="submit" value="submit" … />
```

Overview of Attribute Selectors

- You can define style rules that target elements based on attributes on the elements
  - Use [] to denote the attribute of interest

- You can select elements based on:
  - The presence of an attribute
  - The value of an attribute

Testing for Attribute Presence

- You can define a style rule that targets elements that have a particular attribute (regardless of its value)
  - Use this syntax:

```
anElementName[anAttributeName] {
  declarations…
}
```

- Example
  - See `AttributePresenceSelectors.html`

```
a[name] {
  color: blue;
}
```

← Applies to bookmarks

```
a[href] {
  color: orange;
}
```

← Applies to hyperlinks

Testing for an Attribute Value

- You can define a style rule that targets elements that have a particular attribute value
  - Use = to test the equality
  - Use ^= to test for "starts with of equals" (buggy)

- Example
  - See `AttributeValueSelectors.html`

```
img[src="euroImages/Wales.png"] {
  width: 200px;
}
```

← Applies to specific image

```
img[src^="nonEuroImages/"] {
  width: 100px;
}
```

← Applies to all images in a subfolder

5. Chaining and Grouping Selectors

- Chaining selectors
- Grouping selectors

Chaining Selectors

- You can chain selectors together

- Example
  - What does this rule mean?

```
#mainContent div.narrative h4 + ul > li a[href^="http://acme.com"] {
  font-size: 20pt;
}
```

Grouping Selectors

- You can group selectors together
  - Use a comma separator between selectors

- Example
  - The following are equivalent!

```
h1 { color: orange; background-color: #ffeeee; }
h2 { color: orange; background-color: #ffeeee; }
h3 { color: orange; background-color: #ffeeee; }
h4 { color: orange; background-color: #ffeeee; }
h5 { color: orange; background-color: #ffeeee; }
h6 { color: orange; background-color: #ffeeee; }
```

```
h1, h2, h3, h4, h5, h6 { color: orange; background-color: #ffeeee; }
```

6. Pseudo-Classes and Pseudo-Elements

- Overview of pseudo-classes
- Using pseudo-classes with hyperlinks
- :focus pseudo-class
- Overview of pseudo-elements
- :first-child pseudo-element
- :first-line pseudo-element
- :first-letter pseudo-element
- :before and :after pseudo-elements

Overview of Pseudo-Classes

- CSS supports the concept of pseudo-classes
  - i.e. additional qualifiers that you can place on a selector
  - Allows you to fine-tune when the rule applies

- General syntax for pseudo-classes:

```
aSelector:pseudoClass {
  declarations…
}
```

Using Pseudo-Classes with Hyperlinks

- CSS supports several pseudo-classes for hyperlinks

Pseudo-class	Description
`:link`	Selects all unvisited links
`:visited`	Selects all visited links
`:hover`	Selects links on mouse-over
`:active`	Selects the active link

- Note:
  - `:hover` must come after `:link` and `:visited`
  - `:active` must come after `:hover`

- Example
  - See `LinkPseudoClasses.html`

:focus Pseudo-Class

- The `:focus` pseudo-class selects an element that has input focus

```
aSelector:focus {
  declarations…
}
```

- Example
  - See `FocusPseudoClass.html`

Overview of Pseudo-Elements

- CSS supports the concept of pseudo-elements
  - i.e. additional qualifiers that you can place on a selector
  - Allows you to fine-tune when the rule applies

- General syntax for pseudo-elements:

```
aSelector:pseudoElement {
  declarations…
}
```

The :first-child Pseudo-Element

- The `:first-child` pseudo-element selects an element that is the first child of another element

```
aSelector:first-child {
  declarations…
}
```

- Example
  - See `FirstChildPseudoElement.html`

The :first-line Pseudo-Element

- The `:first-line` pseudo-element selects the first line within an element

```
aSelector:first-line {
  declarations…
}
```

- Example
  - See `FirstLinePseudoElement.html`

The :first-letter Pseudo-Element

- The `:first-letter` pseudo-element selects the first letter within an element

```
aSelector:first-letter {
  declarations…
}
```

- Example
  - See `FirstLetterPseudoElement.html`

:before and :after Pseudo-Elements (1)

- The `:before` and `:after` pseudo-elements can be used to insert some content before/after the content of an element

```
aSelector:before {
  content: someContent
}
```

```
aSelector:after {
  content: someContent
}
```

:before and :after Pseudo-Elements (2)

- You can generate the following kind of content for the `:before` and `:after` pseudo-elements:
  - A string
  - A URL
  - A counter

- Note:
  - You can reset and increment a counter as follows:

```
aSelector {
  counter-reset: aCounter optValue;
}
```

```
aSelector {
  counter-increment: aCounter optAmt;
}
```

  - To display a counter:

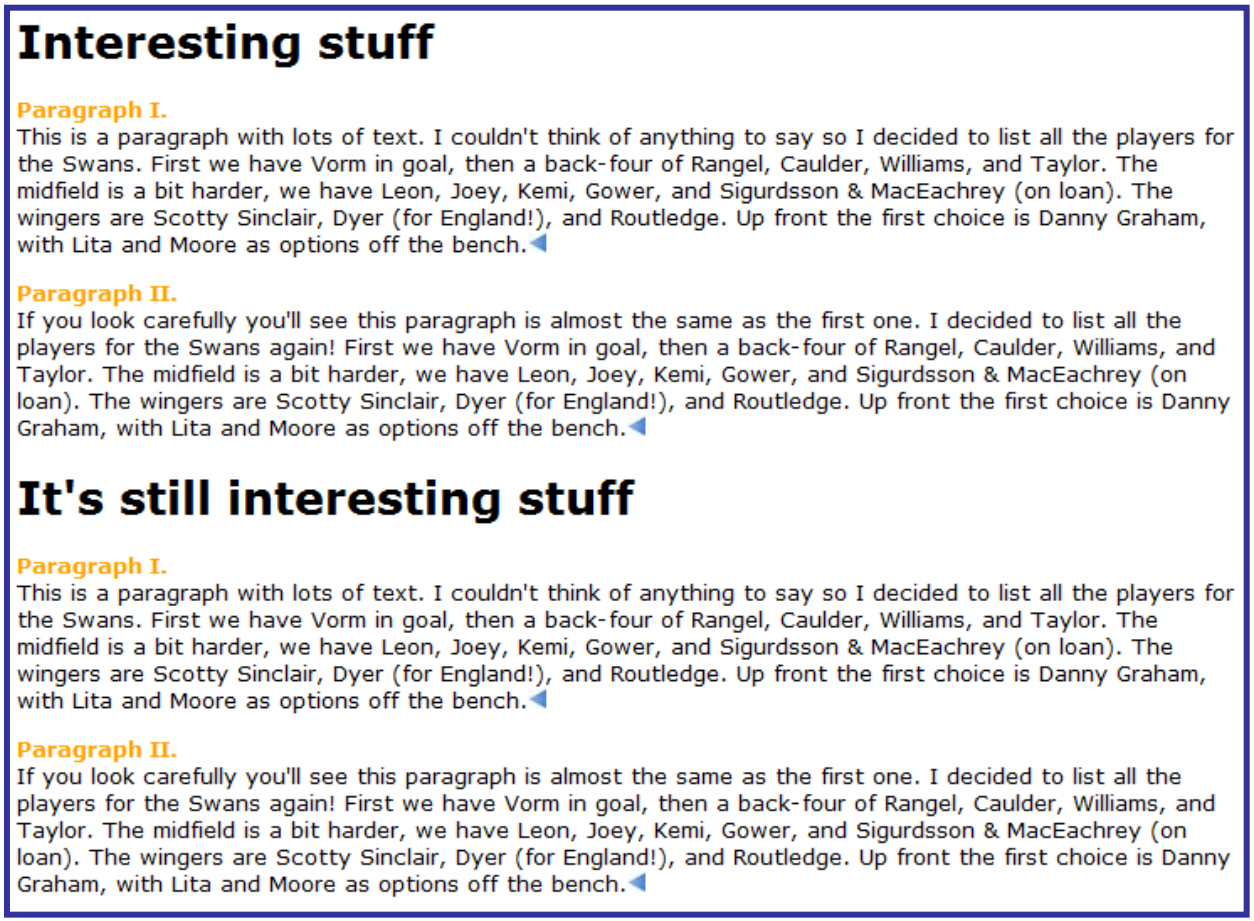
```
aSelector {
  content: counter(aCounter, aListStyle);
}
```

For details of list styles, see:

www.w3.org/TR/CSS2/generate.html#propdef-list-style-type

:before and :after Pseudo-Elements (3)

- Example
  - See `BeforeAndAfterPseudoElements.html`
  - Illustrates various content types and techniques



Any Questions?