CSS-2

Nesli Erdogmus

neslierdogmus@iyte.edu.tr

CSS Syntax

- The CSS rule opens with a **selector**.
- We then have a set of **curly braces**.
- Inside the braces will be one or more **declarations**, which take the form of **property** and **value** pairs.
    - In our example, in the second declaration for the level one headings (selected by `h1`), we have the `color` property, which can take various color values (`red` being one of them).

```
h1 {
      background-color: yellow;
      color: red;
}
```

CSS Comments

- CSS comments and starts with `/*` and ends with `*/`.
  - Differently in HTML, you can add comments by using the <!--...--> syntax.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      /* This is a single-line comment */
      p {
        color: red; /* Set text color to red */
      }
      /* This is
      a multi-line
      comment */
      h1 {
        font-size: 5em;
      }
    </style>
  </head>
  <body>
    <h1>Hello</h1>
    <p>It's me...</p>
  </body>
</html>
```

CSS Selectors

- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- We can divide CSS selectors into five categories:
  - Simple selectors (select elements based on name, id, class) ✔
  - Combinator selectors (select elements based on a specific relationship between them)
  - Pseudo-class selectors (select elements based on a certain state)
  - Pseudo-elements selectors (select and style a part of an element)
  - Attribute selectors (select elements based on an attribute or attribute value)

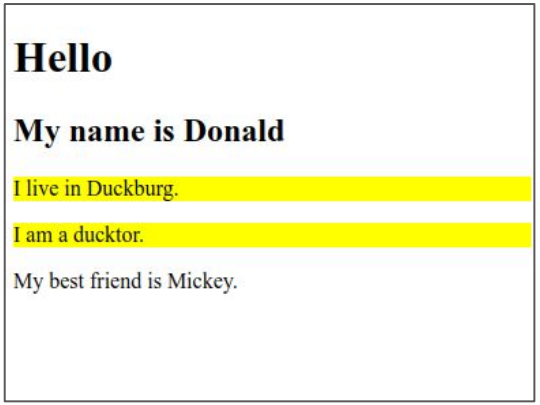
Combinator selectors

- A combinator is something that explains the relationship between the selectors.
  - A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.

Selector	Example	Example description
element element	div p	Selects all <p> elements inside elements
element>element	div > p	Selects all <p> elements where the parent is a element
element+element	div + p	Selects the first <p> element that are placed immediately after elements
element1~element2	p ~ ul	Selects every element that are preceded by a <p> element

element element Selector

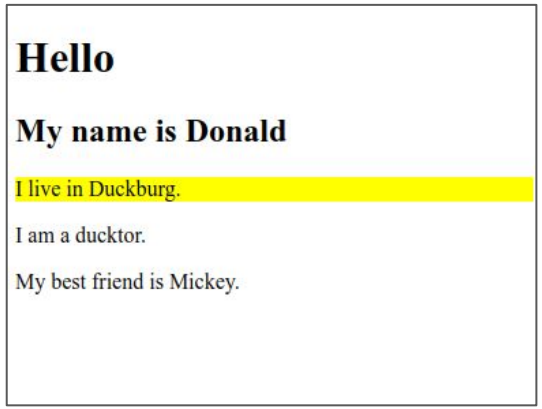
- Also called **descendant** selector, this selector matches all elements that are descendants of a specified element.




```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div p {
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <h1>Hello</h1>
    <div>
      <h2>My name is Donald</h2>
      <p>I live in Duckburg.</p>
      <section>
        <p>I am a ducktor.</p>
      </section>
    </div>
    <p>My best friend is Mickey.</p>
</body>
</html>
```

element>element Selector

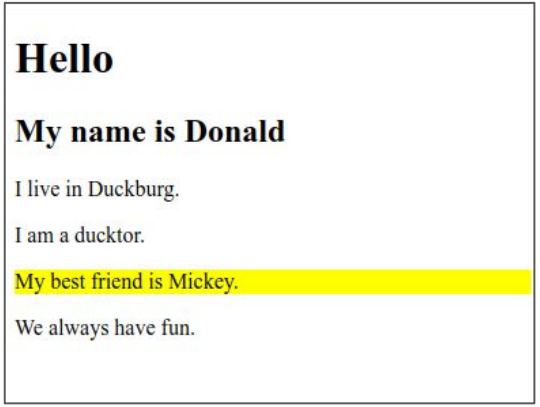
- Also called **child** selector, this selector selects all elements that are the children of a specified element.




```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div>p {
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <h1>Hello</h1>
    <div>
      <h2>My name is Donald</h2>
      <p>I live in Duckburg.</p>
      <section>
        <p>I am a ducktor.</p>
      </section>
    </div>
    <p>My best friend is Mickey.</p>
</body>
</html>
```

element+element Selector

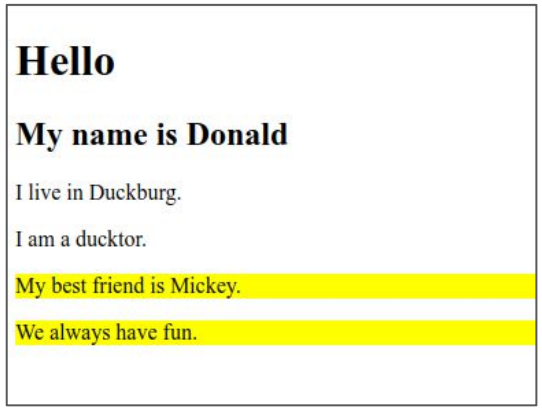
- Also called **adjacent sibling** selector, this selector is used to select an element that is directly after another specific element.




```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div+p {
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <h1>Hello</h1>
    <div>
      <h2>My name is Donald</h2>
      <p>I live in Duckburg.</p>
      <section>
        <p>I am a ducktor.</p>
      </section>
    </div>
    <p>My best friend is Mickey.</p>
    <p>We always have fun.</p>
</body>
</html>
```

element~element Selector

- Also called **general sibling** selector, this selector selects all elements that are next siblings of a specified element.




```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div~p {
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <h1>Hello</h1>
    <div>
      <h2>My name is Donald</h2>
      <p>I live in Duckburg.</p>
      <section>
        <p>I am a ducktor.</p>
      </section>
    </div>
    <p>My best friend is Mickey.</p>
    <p>We always have fun.</p>
</body>
</html>
```

CSS Selectors

- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- We can divide CSS selectors into five categories:
  - Simple selectors (select elements based on name, id, class) ✔
  - Combinator selectors (select elements based on a specific relationship between them) ✔
  - Pseudo-class selectors (select elements based on a certain state)
  - Pseudo-elements selectors (select and style a part of an element)
  - Attribute selectors (select elements based on an attribute or attribute value)

Pseudo-class selectors

- A pseudo-class is used to define a special state of an element. For example, it can be used to:
  - Style an element when a user mouses over it
  - Style visited and unvisited links differently
  - Style an element when it gets focus
- Let’s look at the examples at: https://www.w3schools.com/css/css_pseudo_classes.asp

CSS Selectors

- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- We can divide CSS selectors into five categories:
  - Simple selectors (select elements based on name, id, class) ✔
  - Combinator selectors (select elements based on a specific relationship between them) ✔
  - Pseudo-class selectors (select elements based on a certain state) ✔
  - Pseudo-elements selectors (select and style a part of an element)
  - Attribute selectors (select elements based on an attribute or attribute value)

Pseudo-element selectors

- A pseudo-element is used to style specified parts of an element. For example, it can be used to:
  - Style the first letter, or line, of an element
  - Insert content before, or after, the content of an element
- Let’s look at the examples at: https://www.w3schools.com/css/css_pseudo_elements.asp

CSS Selectors

- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- We can divide CSS selectors into five categories:
  - Simple selectors (select elements based on name, id, class) ✔
  - Combinator selectors (select elements based on a specific relationship between them) ✔
  - Pseudo-class selectors (select elements based on a certain state) ✔
  - Pseudo-elements selectors (select and style a part of an element) ✔
  - Attribute selectors (select elements based on an attribute or attribute value)

Attribute selectors

- It is possible to style HTML elements that have specific attributes or attribute values.
- These selectors enable the selection of an element based on the presence of an attribute alone (for example href), or on various different matches against the value of the attribute.

Attribute selectors

Selector	Example	Description
[*attr*]	a[title]	Matches elements with an *attr* attribute (whose name is the value in square brackets).
[attr=value]	a[href="https://example.com"]	Matches elements with an *attr* attribute whose value is exactly *value* — the string inside the quotes.
[attr~=value]	p[class~="special"]	Matches elements with an *attr* attribute whose value is exactly *value*, or contains *value* in its (space separated) list of values.
[attr|=value]	div[lang|="zh"]	Matches elements with an *attr* attribute whose value is exactly *value* or begins with *value* immediately followed by a hyphen.

Cascading Order

from last week's lecture:

- What style will be used when there is more than one style specified for an HTML element?

- All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:
  1. Inline style (inside an HTML element)
  2. External and internal style sheets (in the head section)
  3. Browser default
- So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

Multiple Styles

from last week's lecture:

- What happens if some properties have been defined for the same selector (element) in different style sheets?
- The value from the last read style sheet will be used.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      h1   {color: blue;}
      h1   {color: red;}
    </style>
  </head>
  <body>
    <h1>This is a heading</h1>
  </body>
</html>
```

Cascade, specificity, inheritance

- These control how CSS is applied to HTML and how conflicts between style declarations are resolved.
  - **Cascade**: The algorithm that defines how user agents combine property values originating from different sources.
  - **Specificity**: The algorithm used by browsers to determine the CSS declaration that is the most relevant to an element, which in turn, determines the property value to apply to the element.
  - **Inheritance**: The set of rules that controls what happens when no value is specified for a property on an element.

Cascade

- **Cascade**: It defines the origin and layer that takes precedence when declarations in more than one **origin** or **cascade layer** set a value for a property on an element.
  - **Origin**: CSS declarations come from different origin types:
    - User-agent stylesheets
    - User stylesheets
    - Author stylesheets
  - **Cascade layers**: For all origins - user-agent, author, or user - styles can be declared within or outside of named or anonymous layers. (`layer`, `layer()` or `@layer`)
    - Styles declared outside of a layer are treated as being part of an anonymous last declared layer.

https://developer.mozilla.org/en-US/docs/Web/CSS/@layer

Cascading Order

1. **Relevance**: Extract all the rules that apply to a given element.
2. **Origin and importance**: Sort these rules according to their importance:

Order (low to high)	Origin	Importance
1	user-agent (browser)	normal
2	user	normal
3	author (developer)	normal
4	CSS @keyframe animations	
5	author (developer)	!important
6	user	!important
7	user-agent (browser)	!important
8	CSS transitions	

Cascading Order

- Styles can be contained within layers within their origin type.
  - With author styles, there is also the issue of where inline styles land in the cascade order.
- The order in which layers are declared is important in determining precedence.
- Normal styles in a layer take precedence over styles declared in prior layers.
  - Normal styles declared outside of any layer take precedence over normal layered styles regardless of specificity.

Cascading Order

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      @import unlayeredStyles.css;
      @import AStyles.css layer(A);
      @import moreUnlayeredStyles.css;
      @import BStyles.css layer(B);
      @import CStyles.css layer(C);
      p {
        color: red;
        padding: 1em !important;
      }
    </style>

...

<p style="line-height: 1.6em;
text-decoration: overline
!important;">Hello</p>
```

Order (low to high)	Author style	Importance
1	A - first layer	normal
2	B - second layer	normal
3	C - last layer	normal
4	All unlayered styles	normal
5	inline style	normal
6	animations	
7	All unlayered styles	`!important`
8	C - last layer	`!important`
9	B - second layer	`!important`
10	A - first layer	`!important`
11	inline `style`	`!important`
12	transitions	

Cascading Order

Precedence Order (low to high)	Style Origin	Importance
1	user-agent - first declared layer user-agent - last declared layer user-agent - unlayered styles	normal
2	user - first declared layer user - last declared layer user - unlayered styles	normal
3	author - first declared layer author - last declared layer author - unlayered styles inline `style`	normal
4	animations	
5	author - unlayered styles author - last declared layer author - first declared layer inline `style`	`!important`
6	user - unlayered styles user - last declared layer user - first declared layer	`!important`
7	user-agent - unlayered styles user-agent - last declared layer user-agent - first declared layer	`!important`
8	transitions	

Cascading Order

1. **Relevance**: Extract all the rules that apply to a given element.
2. **Origin and importance**: Sort these rules according to their importance:
3. **Specificity**: In case of equality with an origin, the specificity of a rule is considered to choose one value or another.
4. **Order of appearance**: In case of equal origin and specificity, the last declaration in the style order is applied.

Specificity

- The specificity algorithm calculates the weight of a CSS selector to determine which rule from competing CSS declarations gets applied to an element.
- The weight is determined by the number of selectors of each weight category in the selector matching the element (or pseudo-element).
    - If there are two or more declarations providing different property values for the same element, the declaration value in the style block having the matching selector with the greatest algorithmic weight gets applied.
    - The specificity algorithm is basically a three-column value of three categories or weights - ID, CLASS, and TYPE - corresponding to the three types of selectors.
    - The three columns are created by counting the number of selector components for each selector weight category in the selectors that match the element.

Specificity

- The selector weight categories are as the following:
  - **ID column**: Includes only ID selectors, such as `#example`.
  - **CLASS column**: Includes class selectors, such as `.myClass`, attribute selectors like `[type="radio"]` and `[lang|="fr"]`, and pseudo-classes, such as `:hover`, `:nth-of-type(3n)`, and `:required`.
  - **TYPE column**: Includes type selectors, such as `p`, `h1`, and `td`, and pseudo-elements like `::before`, `::placeholder`, and all other selectors with double-colon notation.
  - **No value**: The universal selector (`*`) and the pseudo-class `:where()` and its parameters aren't counted when calculating the weight
- Combinators, such as `+`, `>`, `~`, `" "`, and `||` don't add any value to the specificity weight.
- Some details and exceptions are not mentioned here.

Specificity

- In the following CSS, we have three selectors targeting <input> elements to set a color.
  - If those selectors all target the same input, the input will be red, as the first declaration has the highest value in the ID column.

```
#myElement input.myClass {
  color: red;
} /* 1-1-1 */
input[type="password"]:required {
  color: blue;
} /* 0-2-1 */
html body main input {
  color: green;
} /* 0-0-4 */
```

Specificity

- In the following CSS, we have three selectors targeting <input> elements to set a color.
  - If we convert the id selector to an attribute selector, the first two selectors would have the same specificity. For multiple declarations with equal specificity, the last declaration found in the CSS is applied to the element. In this case, the color will be blue.

```
[id="myElement"] input.myClass {
  color: red;
} /* 0-2-1 */
input[type="password"]:required {
  color: blue;
} /* 0-2-1 */
html body main input {
  color: green;
} /* 0-0-4 */
```

Inheritance

- CSS properties can be categorized in two types:
  - Inherited properties, which by default are set to the computed value of the parent element
  - Non-inherited properties, which by default are set to initial value of the property

```
p {
  color: green;
}
```

```
<p>This paragraph has <em>emphasized
text</em> in it.</p>
```

This paragraph has *emphasized text* in it.

```
p {
  border: medium solid;
}
```

```
<p>This paragraph has <em>emphasized
text</em> in it.</p>
```

This paragraph has *emphasized text* in it.

Inheritance

```css
.main {
    color: rebeccapurple;
    border: 2px solid #ccc;
    padding: 1em;
}
.special {
    color: black;
    font-weight: bold;
}
```

```html
<ul class="main">
    <li>Item One</li>
    <li>Item Two
        <ul>
            <li>2.1</li>
            <li>2.2</li>
        </ul>
    </li>
    <li>Item Three
        <ul class="special">
            <li>3.1
                <ul>
                    <li>3.1.1</li>
                    <li>3.1.2</li>
                </ul>
            </li>
            <li>3.2</li>
        </ul>
    </li>
</ul>
```

- Item One
- Item Two
  - 2.1
  - 2.2
- Item Three
  - **3.1**
    - **3.1.1**
    - **3.1.2**
  - **3.2**

Inheritance

- The `inherit` keyword allows authors to explicitly specify inheritance.

```
p {
  color: green;
}
```

```
<p>This paragraph has <em>emphasized
text</em> in it.</p>
```

This paragraph has *emphasized text* in it.

```
p {
  border: medium solid;
}
em {
  border: inherit;
}
```

```
<p>This paragraph has <em>emphasized
text</em> in it.</p>
```

This paragraph has *emphasized text* in it.

Inheritance

- The `initial` keyword applies the initial (or default) value of a property to an element.
  - On inherited properties, the initial value may be unexpected. You should consider using the `inherit`, `unset`, or `revert` keywords instead.

```
p {
  color: red;
}

em {
  color: initial;
}
```

```
<p>
  <span>This text is red.</span>
  <em>This text is in the initial
color (typically black).</em>
  <span>This is red again.</span>
</p>
```

This text is red. *This text is in the initial color (typically black).* This is red again.

Inheritance

- The `revert` keyword reverts the cascaded value of the property from its current value to the value the property would have had if no changes had been made by the current style origin to the current element.
  - This keyword removes from the cascade all of the styles that have been overridden until the style being rolled back to is reached.
- The `unset` keyword resets a property to its inherited value if the property naturally inherits from its parent, and to its initial value if not.
  - It behaves like the `inherit` keyword in the first case, when the property is an inherited property, and like the `initial` keyword in the second case, when the property is a non-inherited property.

Inheritance

```css
h3 {
  font-weight: normal;
  color: blue;
}
```

```html
<h3 style="font-weight: revert; color: revert;">
  This should have its original font-weight (bold) and color: black
</h3>
<p>Just some text</p>
<h3 style="font-weight: unset; color: unset;">
  This will still have font-weight: normal, but color: black
</h3>
<p>Just some text</p>
```

This should have its original font-weight (bold) and color: black

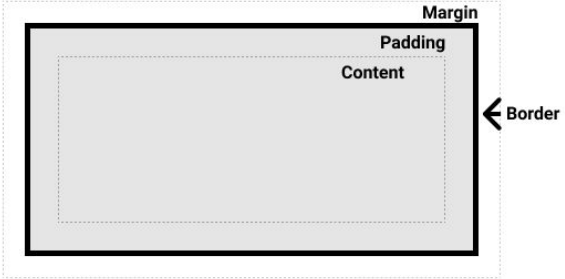
Just some text

This will still have font-weight: normal, but color: black

Just some text

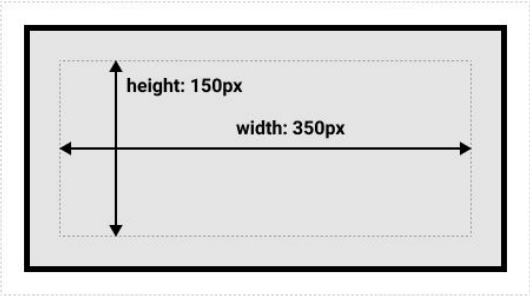
CSS Box Model

- Everything in CSS has a box around it.
- Understanding these boxes is key to
  - being able to create more complex layouts with CSS
  - being able to align items with other items.
- A block box has:
  - Content box
  - Padding box
  - Border box
  - Margin box



CSS Box Model

```css
.box {
  width: 350px;
  height: 150px;
  margin: 10px;
  padding: 25px;
  border: 5px solid black;
}
```




```css
.box {
  box-sizing: border-box;
  width: 350px;
  height: 150px;
  margin: 10px;
  padding: 25px;
  border: 5px solid black;
}
```

In the alternative box model, any width is the width of the visible box on the page.

CSS Box Model

- When there is too much content to fit in a box, **overflow** happens.
  - Wherever possible, CSS does not hide content. This would cause data loss.

```
/* Keyword values */
overflow: visible;
overflow: hidden;
overflow: clip;
overflow: scroll;
overflow: auto;
overflow: hidden visible;

/* Global values */
overflow: inherit;
overflow: initial;
overflow: revert;
overflow: revert-layer;
overflow: unset;
```

CSS Box Model

- There are several types of boxes that have different behaviour in terms of page flow and in relation to other boxes on the page.
- Boxes have an inner display type and an outer display type.
  - Outer display type controls behaviour like if the box will break into a new line, respect width and height properties, etc. (block / inline)
  - Inner display type dictates how elements inside that box are laid out.

CSS Layout

- CSS page layout techniques allow us to take elements contained in a web page and control where they're positioned relative to the following factors:
  - their default position in normal layout flow
  - the other elements around them, their parent container
  - the main viewport/window
- Normal flow: The default way of the browser to lay out HTML pages.
  - The order of elements is respected.
  - The elements are stacked on top of one another.

CSS Layout

- For more complex layouts, you can alter this default behavior using some CSS tools:
  - The `display` property
  - Floats
  - The `position` property
  - Table layout
  - Multi-column layout

CSS Layout

- The `display` CSS property sets whether an element is treated as a block or inline element and the layout used for its children, such as flow layout, grid or flex.

```
/* precomposed values */
display: block;
display: inline;
display: inline-block;
display: flex;
display: inline-flex;
display: grid;
display: inline-grid;
display: flow-root;

/* box generation */
display: none;
display: contents;
```

```
/* multi-keyword syntax */
display: block flow;
display: inline flow;
display: inline flow-root;
display: block flex;
display: inline flex;
display: block grid;
display: inline grid;
display: block flow-root;
```

```
/* other values */
display: table;
display: table-row; /* all table
elements have an equivalent CSS
display value */
display: list-item;

/* Global values */
display: inherit;
display: initial;
display: revert;
display: revert-layer;
display: unset;
```

CSS preprocessor

- A CSS preprocessor is a program that lets you generate CSS from the preprocessor's own unique syntax.
- To use a CSS preprocessor:
  - you must install a CSS compiler on your web server OR
  - you must use the CSS preprocessor to compile on the development environment, and then upload compiled CSS file to the web server.
- Most CSS preprocessors will add some features that don't exist in pure CSS, such as mixin, nesting selector, inheritance selector, and so on. These features make the CSS structure more readable and easier to maintain.

CSS preprocessor

- A few of most popular CSS preprocessors:
  - Sass: https://sass-lang.com/
  - LESS: https://lesscss.org/
  - Stylus: https://stylus-lang.com/
  - PostCSS: https://postcss.org/