CSS Fundamentals

.NET

CSS (Cascading Style Sheets) is a language that describes the style that the browser will give to HTML elements on screen. CSS rulesets are stored in a .css file called a stylesheet.

# Preparation

- Create a directory, HtmlAndCssPractice, in your personal repo.
- Right-Click the directory to open it in VS Code.
- Create a .html file, HtmlPractice.html.
- In VS Code, download the extension 'Live Server' by Ritwick Dey
- Reload your VS Code window with CTRL + Shift + P (to open Command Palette) and Type 'reload window'.
- Type 'doc' + tab to auto-fill the .html page with the HTML template.
- Add <link rel="stylesheet" type="text/css" href="CssPractice.css" /> to the <head> section of your .html just below the <title>.
- Create a file, CssPractice.css, in the same folder as your HtmlPractice.html file.
- Right-Click on the HtmlPractice.html file in the explorer window and click 'Open with Live Server'. This will open the .html file in your browser and automatically update the browser any time your HtmlPractice.html file is saved.

CSS – Why?

- *HTML* is not intended to be responsible for style and formatting.
- Style Formatting *tags* were added with HTML 3.2.
- Development of large websites became a laborious process with fonts and color information added to every page.
- The *World Wide Web Consortium (W3C)* created CSS to enable *separation of concerns* between structure and presentation of documents (HTML + CSS).
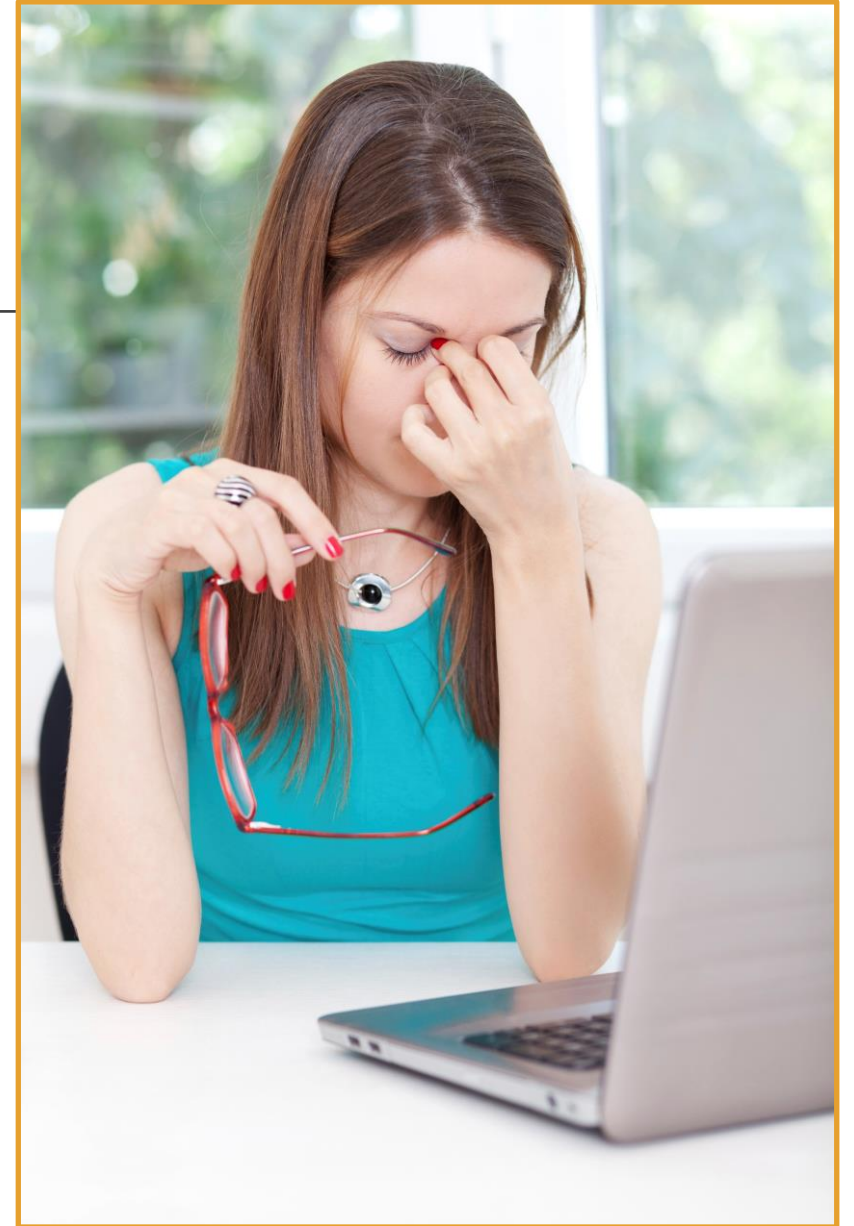
CSS - Adding Styling (1/3) - External

https://www.w3schools.com/css/css_howto.asp

External .css file (Recommended): Include a reference to the relative location of the .css file inside a <link> element in the <head> section.

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

Benefits of External CSS.

- Separation of concerns - HTML and CSS are in separate documents, you shouldn't mix them in the same file.
- Reusability - the same stylesheet can be used to style many .html files.
- Central location - Change styling in just one place.
- Readability - HTML becomes less cluttered, and CSS styling is easier to understand.

CSS - Adding Styling (2/3) - Internal

Internal CSS :

- defined inside the <style> element inside the <head> section.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
<body>
```

CSS - Adding Styling (3/3) - Inline

Inline CSS :

- used to apply a unique style to a single *element*,
- add a *style* attribute inside the relevant *elements* opening tag.

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>

</body>
</html>
```

CSS – Syntax

https://www.w3schools.com/css/css_syntax.asp

A *rule* or "**rule set**" is a statement that tells browsers how to render particular elements on an HTML page.

The *selector* points to the HTML element you want to style.

The *declaration* block is bounded by '{ }', contains *declarations* separated by ';'.

Selector

Declaration

Declaration

h1

{ color:blue; font-size:12px;}

Property

Value

Property

Value

Each declaration includes a CSS *property* name and a *value*, separated by a colon.

CSS – Selector

Selector Type	Example RuleSet	Description
element Selector	div { declarations }	Selects 1 or more HTML elements based on the tag name.
id Selector	#id1 { declarations }	Selects HTML elements based on the id attribute name.
class Selector	.center { declarations }	Selects HTML elements based on the class attribute name.
Universal Selector	* { declarations }	Selects all HTML elements on the page.
Selector List	h1, h2, p { declarations }	Selects all indicated HTML tags in the document.
Concatenation	p.center { declarations }	Selects all center class elements inside a <p> element.

```css
p {
  text-align: center;
  color: red;
}
```

```css
#para1 {
    text-align: center;
    color: red;
}
```

```css
.center {
  text-align: center;
  color: red;
}
```

```css
p.center {
    text-align: center;
    color: red;
}
```

```css
h1, h2, p {
    text-align: center;
    color: red;
}
```

```css
* {
    text-align: center;
    color: blue;
}
```

CSS – Pseudo-Selector

There are two types of *Pseudo-Selectors*.

Pseudo-Class Selectors	*Pseudo-Element* Selectors
Used to define a special state of an *element*, like displaying a visited link differently or changing an *elements* color when the mouse *hovers* over it.	A CSS *pseudo-element* is used to style specified parts of an *element*, like the first letter (or line) of an element.

CSS – Pseudo Class Selectors

A ***pseudo-class*** applies styling only when an element is in a certain ***state***. It is a colon and keyword appended to a ***selector***:

- when the mouse hovers over the element
- when a checkbox is checked.

Use a colon (**:**) in front of the ***pseudo-class*** name. These are just a few of the over 30 ***pseudo-classes*** available.

- p.myClass:active
- a.myClass:visited
- a:checked
- a.myClass:disabled
- div:first
- a.myClass:nth-child(n)
- a.myClass:hover

CSS –
Pseudo Class Examples

This is a link

This link was clicked

The mouse is over this link

This link is selected

```
<head>
<style>
/* unvisited link */
a:link {
  color: red;
}

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

/* mouse over link */
a:hover {
  color: hotpink;
}

/* selected link */
a:active {
  color: blue;
}
</style>
</head>
<body>
```

CSS – Pseudo Classes with HTML Classes

Pseudo Classes can be placed on HTML elements to be more specific as to which elements you want the style.

```
<!DOCTYPE html>
<html>
<head>
<style>
a.highlight:hover {
    color: #ff0000;
}
</style>
</head>
<body>

<p><a class="highlight" href="css_syntax.asp">CSS Syntax</a></p>
<p><a href="default.asp">CSS Tutorial</a></p>

</body>
</html>
```

CSS Syntax

CSS Tutorial

CSS – Pseudo Elements

https://www.w3schools.com/css/css_pseudo_elements.asp

A CSS *pseudo-element* is used to style specified parts of an *element*.

To use it, place two colons (::) between the *element* and the *pseudo element*.

Pseudo-Elements allow you to style just the first letter (or line) of an element or insert other content before or after the content of an element

- ::before
- ::after
- ::first-line
- ::first-letter

This example formats the first line of the text in all `<p>` elements:

```
p::first-line {
    color: #ff0000;
    font-variant: small-caps;
}
```

<p>You can use the ::first-line pseudo-element to add a special effect to the first line of a text. Some more text. And even more, and more, and more, etc...</p>

YOU CAN USE THE ::FIRST-LINE PSEUDO-ELEMENT TO ADD A SPECIAL effect to the first line of a text. Some more text. And even more, and more, and more, and more, and more, and more, and more, and more, and more, and more, and more, and more.

CSS – Chaining Pseudo Elements

https://www.w3schools.com/css/css_pseudo_elements.asp

You can combine a *pseudo class* and a *pseudo element* to combine the two effects.

If you wanted to you could chain the :first-child and ::first-line selectors together.

```
article p:first-child::first-line {
 font-size: 120%;
 font-weight: bold;
}
```

The above code will make the first line of the first paragraph bold

CSS – Combinators

https://www.w3schools.com/css/css_combinators.asp

The *selector* part of a CSS *RuleSet* can contain more than one *selector*.

Between the *selectors*, we can a *combinator*. There are four different *combinators*. A *combinator* describes the relationship between two **selectors**.

Selector	Symbol	Description
descendant selector	(space) p .myClass{ ruleset}	Matches any descendant element (nested inside) of a specified element. This includes grandchildren, etc.
child selector	>	Selects only elements that are the immediate descendants (children) of an element.
Adjacent (next) sibling selector	+	Selects the next (sibling) element only if the sibling occurs immediately after this element.
general sibling selector	~	Selects all sibling elements that follow this element.

CSS Combinator - Descendent

Descendent means any <p> element nested inside a <div> to unlimited depth.

```
<style>
div p {
    background-color: yellow;
}
</style>
</head>
<body>

<div>
  <p>Paragraph 1 in the div.</p>
  <p>Paragraph 2 in the div.</p>
  <section><p>Paragraph 3 in the div.</p></section>
</div>

<p>Paragraph 4. Not in a div.</p>
<p>Paragraph 5. Not in a div.</p>
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3 in the div.

Paragraph 4. Not in a div.

Paragraph 5. Not in a div.

CSS Combinators Example - Child

Child means any **\<p>** element inside the **\<div>** but only one level deep. This means grandchild elements are not styled.

```html
<head>
<style>
div > p {
  background-color: yellow;
}
</style>
</head>
<body>

<div>
  <p>Paragraph 1 in the div.</p>
  <p>Paragraph 2 in the div.</p>
  <section><p>Paragraph 3 in the div.</p></section>
  <p>Paragraph 4 in the div.</p>
</div>

<p>Paragraph 5. Not in a div.</p>
<p>Paragraph 6. Not in a div.</p>

</body>
```

```
<!-- not Child but Descendant -->
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3 in the div.

Paragraph 4 in the div.

Paragraph 5. Not in a div.

Paragraph 6. Not in a div.

CSS Combinators – Adjacent Sibling

Adjacent Sibling
means the first <p>
sibling immediately
following a div.

```
<head>
<style>
div + p {
    background-color: yellow;
}
</style>
</head>
<body>

<div>
    <p>Paragraph 1 in the div.</p>
    <p>Paragraph 2 in the div.</p>
</div>

<p>Paragraph 3. Not in a div.</p>
<p>Paragraph 4. Not in a div.</p>

</body>
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3. Not in a div.

Paragraph 4. Not in a div.

CSS Combinators Example – General Sibling

In this example, *General Sibling* means all <p> siblings occurring <u>after</u> the div.

```
<head>
<style>
div ~ p {
    background-color: yellow;
}
</style>
</head>
<body>

<p>Paragraph 1.</p>

<div>
    <p>Paragraph 2.</p>
</div>

<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>

</body>
```

Paragraph 1.

Paragraph 2.

Paragraph 3.

Some code.

Paragraph 4.

CSS – Units of measurement

There are two ways to give measurement in CSS. Both *Absolute* and *Relative* measurements have appropriate uses.

Absolute Measurements	
px	pixels
mm	millimeters
cm	centimeters
in	inches
pt	points(1/72 of an inch)
pc	picas (12 points)

Relative Measurements	
%	percentage relative to parent element's width
em	1em is the same as the font size(scales with it) of the current element
rem	stands for "root em" and works the same except refers to the base font size
vh, vw	these are 1/100th of the height and width of the viewport, respectively

Most common

CSS – Colors

There are four ways color can be set in CSS.

How to Define Colors	
keywords	'red', 'blue', 'aquamarine', etc (Ex. Color:blue;)
hex value	The # sign followed by six hex numbers (0-F) with each pair representing rgb values for up to 256 different values of each. (Ex. Color:#ff0000;)
rgb(r,g,b)	This function specifies values from 0 to 255 for red (r), green (g), and blue (b). Ex. Color:rgb(224,176,255);. There is an optional alpha (transparency) value, but this isn't used much.
hsl(h,s,l)	This function to specifies hue, saturation, and lightness values, 0 to 255, to define a color. (Ex. Color:hsl(240,100%,50%);). There is an optional alpha (transparency) value, but this isn't used much. (Color:hsl(240,100%,50%, .4);)

CSS - Rule Set Conflicts

What happens if two CSS rulesets conflict with each other?

The order that *selectors* are chosen for styling depends on three things:

- Importance
- Specificity
- Source order

What is this cascade?

By combining importance, origin, specificity, and the source order of the style concerned, the CSS cascade assigns a weight to each declaration. This weight is used to determine exactly, and without conflict, which style declarations should be applied to a specific element: the declaration with the highest weight takes precedence.

http://reference.sitepoint.com/css/cascade

7

TMW Code Club

CSS – Specificity (1/4)

Specificity is the means by which browsers decide which CSS *property* values are chosen to style an *element*.

Specificity is a weight given to each *RuleSet*. When multiple *RuleSets* have the same weight, the last occurring *RuleSet* wins.

A *Ruleset* with a more specific selector combination wins over a less specific combination.

Specificity only comes into play when the same *element* is targeted by multiple declarations.

Directly targeted *elements* will always take precedence over inherited styling.

```css
div#test span
{ color: green; }

div span
{ color: blue; }

span
{ color: red; }
```

```html
<div id="test">
    <span>
        Text
    </span>
</div>
```

Although, the rule for blue is more specific than the rule for red, the rule for green is the most specific and will be applied. *RuleSet* order does not matter.

CSS – Specificity of Selectors (2/4)

Selectors by order of increasing specificity:

- Type selectors (e.g., h1) and pseudo-elements (e.g., ::before).
- Class selectors (e.g., .example), attributes selectors (e.g., type="radio") and pseudo-classes (e.g., :hover).
- ID selectors (e.g., #example).

The following CSS styles, Which rule will be applied?

- div { color: red; }   /* least specific - won't be applied */
- #yellow { color: yellow; } /* most specific - will be applied */
- .green { color: green; }   /* in the middle - won't be applied */

```css
#specificityId {
    background-color: red;
}


div {
    background-color: whitesmoke;
}


.specificityClass {
    background-color: brown;
}
```

```html
<div>
  <p class="specificityClass"
        id="specificityId"
        style="background-color:lightgreen;"
        >This is a specificity example</p>
</div>
```

CSS – Specificity and Source Order (3/4)

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

If two selectors have equal importance level and specificity, then the latter rule wins over earlier one.

For example:

- div { color: blue; } /* This comes first, will lose */
- div { color: red; }   /* comes last, so is applied */

Internal styling wins over external styling.

```css
#specificityId {
    background-color: ■red;
}


div {
    background-color: ■whitesmoke;
}


.specificityClass {
    background-color: ■brown;
}
```

```html
<div>
  <p class="specificityClass"
     id="specificityId"
     style="background-color:lightgreen;">
     This is a specificity example</p>
</div>
```

CSS – Specificity and !important (4/4)

The *!important* flag can be used to raise a *RuleSet*'s specificity. This ensures that a style always gets applied.

div { color: red !important; } <!- - always applied - ->

The only way to override an *!important* flag is to apply another *!important* flag on a selector with the same specificity later in the document or on a selector with greater specificity.

*!important is not best practice. Use Specificity instead.

CSS –*Inherit* Keyword

```
<!DOCTYPE html>
<html>
<head>
<style>
span {
  color: blue;
  border: 1px solid black;
}

.extra span {
  color: inherit;
}
</style>
</head>
<body>

<div>
  Here is <span>a span element</span> which is blue, as span
elements are set to be.
</div>

<div class="extra" style="color:green">
  Here is <span>a span element</span> which is green, because
it inherits from its parent.
</div>

<div style="color:red">
  Here is <span>a span element</span> which is blue, as span
elements are set to be.
</div>

</body>
</html>
```

Here is a span element which is blue, as span elements are set to be.
Here is a span element which is green, because it inherits from its parent.
Here is a span element which is blue, as span elements are set to be.

The width, margin, padding, and border of an *element* is not automatically inherited from its parent. Some *elements* inherit but others do not.

The *inherit* keyword specifies that a property shall *inherit* its styling from its parent element.

The *inherit* keyword can be used for any CSS property, and on any HTML *element*.

CSS – display Property

The **display** property defines the display type of an **element**. This overrides whatever default display values the property has.

display: none removes the element from the page completely.

visibility: hidden means that the tag is still given space on the page but isn't seen.

- *inline* - Display on the same line as other elements.
- *block* - Start on a new line.
- *flex* - Make the element a flexbox.
- *none* - Do not display the element at all.

```
p.ex1 {display: none;}
p.ex2 {display: inline;}
p.ex3 {display: block;}
p.ex4 {display: inline-block;}
```

CSS – *position* Property (1/2)

The *position* property is an important property related to the layout of a webpage. It governs the flow of **elements**. You can even take **elements** out of the normal flow of the page to appear where you want.

Important position property values:

- *static* – (default) the element is placed in the normal document flow
- *relative* - the element occurs in its normal place, but can be moved around with *top*, *bottom*, *left*, and *right* properties
- *absolute* - removes the element from the normal flow of the document, fixes it in place relative to the html element or nearest positioned ancestor
- *fixed* - fixes element in place relative to browser window.
- *sticky* - hybrid between *relative* and *fixed*, allowing relative positioning until scrolled to a 'threshold' point

CSS – *position* Property (2/2)

```
<!DOCTYPE html>
<html>
<head>
<style>
h2.pos_left {
  position: relative;
  left: -30px;
}

h2.pos_right {
  position: relative;
  left: 50px;
}
</style>
</head>
<body>

<h1>The position Property</h1>

<p>Relative positioning moves an element RELATIVE to its original position.</p>
<p>The style "left: -30px;" subtracts 30 pixels from the element's original left
position.</p>
<p>The style "left: 50px;" adds 50 pixels to the element's original left position.</p>

<h2 class="pos_left">This heading is moved left according to its normal position</h2>

<h2 class="pos_right">This heading is moved right according to its normal position</h2>

</body>
</html>
```

The position Property

Relative positioning moves an element RELATIVE to its original position.

The style "left: -30px;" subtracts 30 pixels from the element's original left position.

The style "left: 50px;" adds 50 pixels to the element's original left position.

**his heading is moved left according to its normal position**

This heading is moved right according to its normal position

CSS - Responsive Web Design

https://developer.mozilla.org/en-US/docs/Tools/Responsive_Design_Mode

Responsive Web Design is a way of designing web pages so that they render well on any window, screen, or device size. *RWD* has become increasingly important due to the share of internet traffic conducted from cell phones and devices. There are a few ways to design web pages to change based on the size of the screen.

- Fluid grid systems make your element containers sized with *relative* units like percentages instead of absolute units
- Flexible images are sized relatively to prevent rendering outside their container.
- Media queries change CSS based on the window size
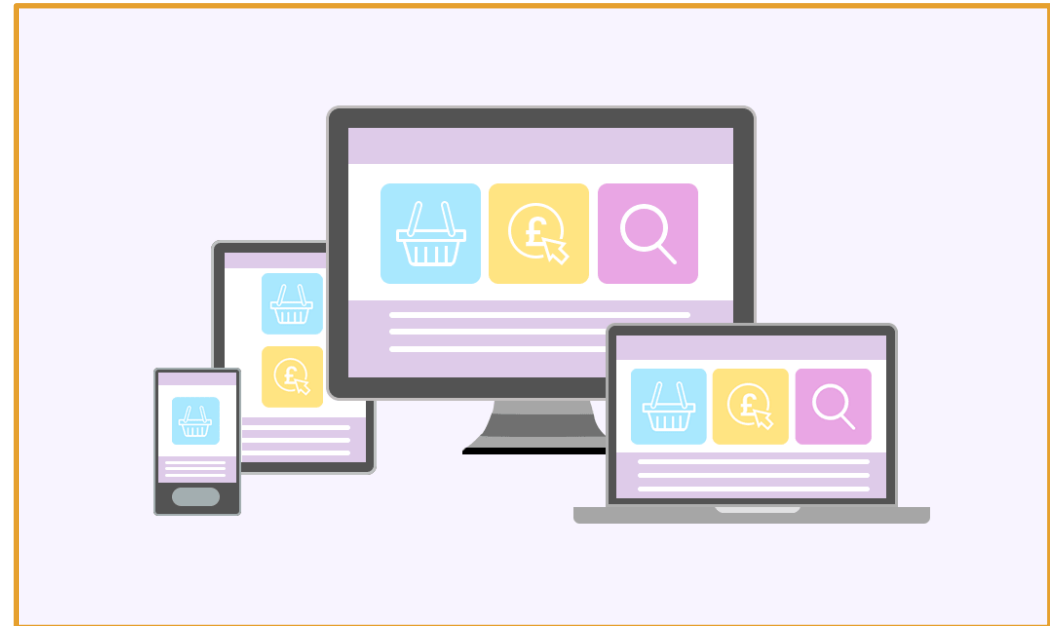
CSS - Media Queries

Media queries are a way of adapting the display of CSS for different screen sizes. They can be used to respond to different media types and features. An example of the syntax is shown below:

A media type can be declared in the *head* of an HTML document using the "*media*" *attribute* inside of a *<link>* element. The value of the "media" *attribute* specifies how the linked document will be displayed on each device.

```css
@media screen and (display-mode: fullscreen) {
  /* Code in here only applies to screens in fullscreen */
}


@media all and (orientation: landscape) {
  /* Code in here only applies in landscape orientation */
}


@media screen and (min-device-width: 500px) {
  /* Code in here only applies to screens equal or greater than 500 pixels wide */
}
```

CSS – 'at-rules'

A *media query* is one type of **@rule**. **@rule**'s tell CSS what to do in some certain situation.

The most important **@rule**'s are:

- **@media** - used for media queries
- **@font-face** - used for defining a custom font
- **@keyframes** - used for animations in CSS

```
@font-face {
  font-family: "Open Sans";
  src: url("/fonts/OpenSans-Regular-webfont.woff2") format("woff2"),
       url("/fonts/OpenSans-Regular-webfont.woff") format("woff");
}
```

Details about CSS3

CSS Level 2 needed 9 years to reach the "Recommendation" status. This was because a few secondary features held back the whole specification.

In order to accelerate the standardization of non-problematic features, the CSS Working Group of the W3C divided CSS in smaller components called *modules*.

Each of these *modules* is now an independent part of the language and moves towards standardization at its own pace.

While some *modules* are already W3C Recommendations, others are still early working drafts. New *modules* are added when new needs are identified.
CSS3 is still evolving but some features are available such as:

- rounded corners, shadows, gradients, transitions and animations, flexbox layouts