

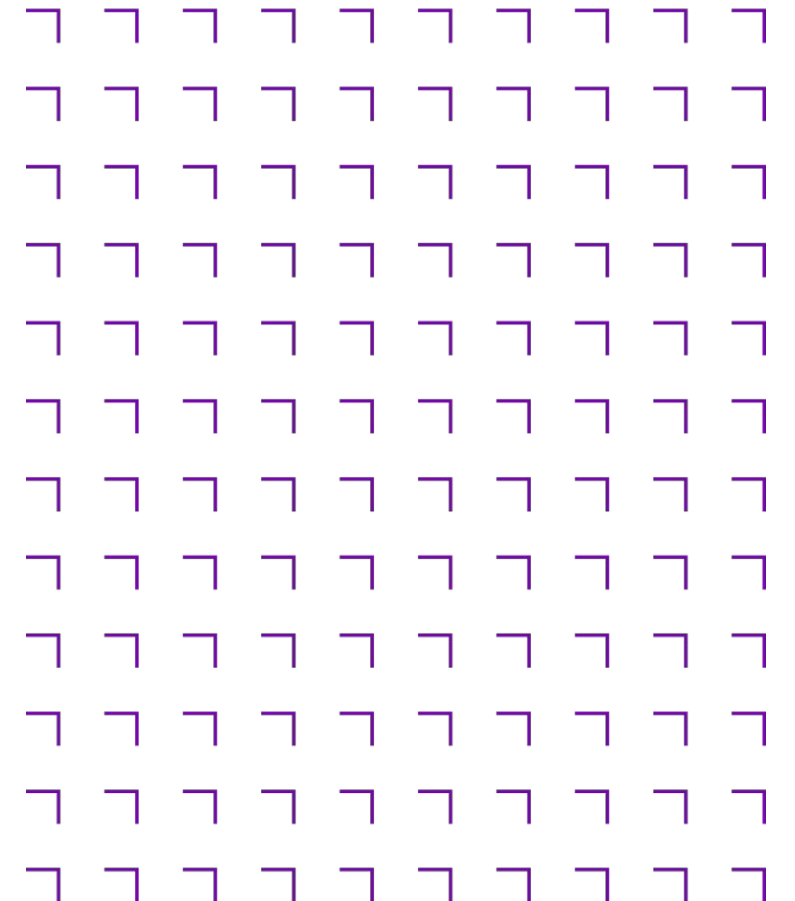
Front-End Web Development

Unit 7: CSS - Advanced

Course Outline



1. Getting Started
2. HTML - Structuring the Web
3. CSS - Styling the Web
4. JavaScript - Dynamic client-side scripting
5. CSS - Making Layouts
6. Introduction to Websites/Web Applications
7. **CSS - Advanced**
8. JavaScript - Modifying the Document Object Model (DOM)
9. Dynamic HTML
10. Web Forms - Working with user data
11. JavaScript - Advanced
12. Building a Web Application with JavaScript
13. Introduction to CSS Frameworks – Bootstrap
14. Building a Web Application with Svelte
15. SEO, Web security, Performance
16. Walkthrough project



Course Learning Outcomes



- Competently write HTML and CSS code
- Create web page layouts according to requirements using styles
- Add interactivity to a web page with JavaScript
- Access and display third-party data on the web page
- Leverage Bootstrap and Static Site Generator



- Final Project - 100% of the grade

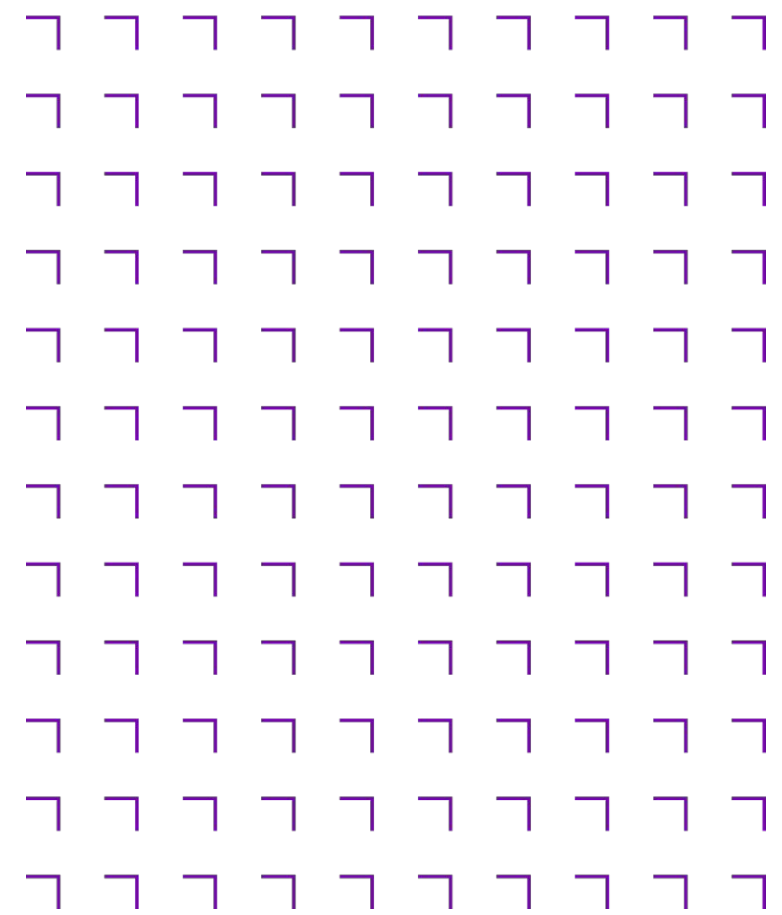
- Design and Build functioning Website using HTML5, CSS (including Bootstrap), JavaScript (browser only)

- ✓ Code will be managed in GitHub
- ✓ Website will be deployed to GitHub Pages
- ✓ All code to follow best practice and be documented

- Details and How-To-Guide are available on the course page under the section called Assessments

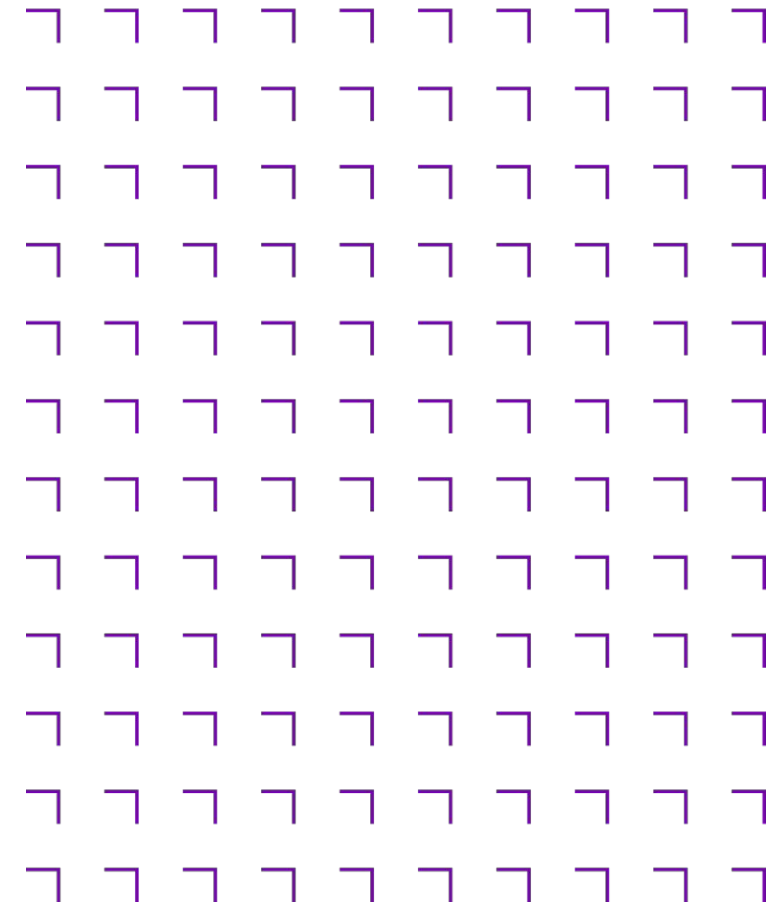
Assessment







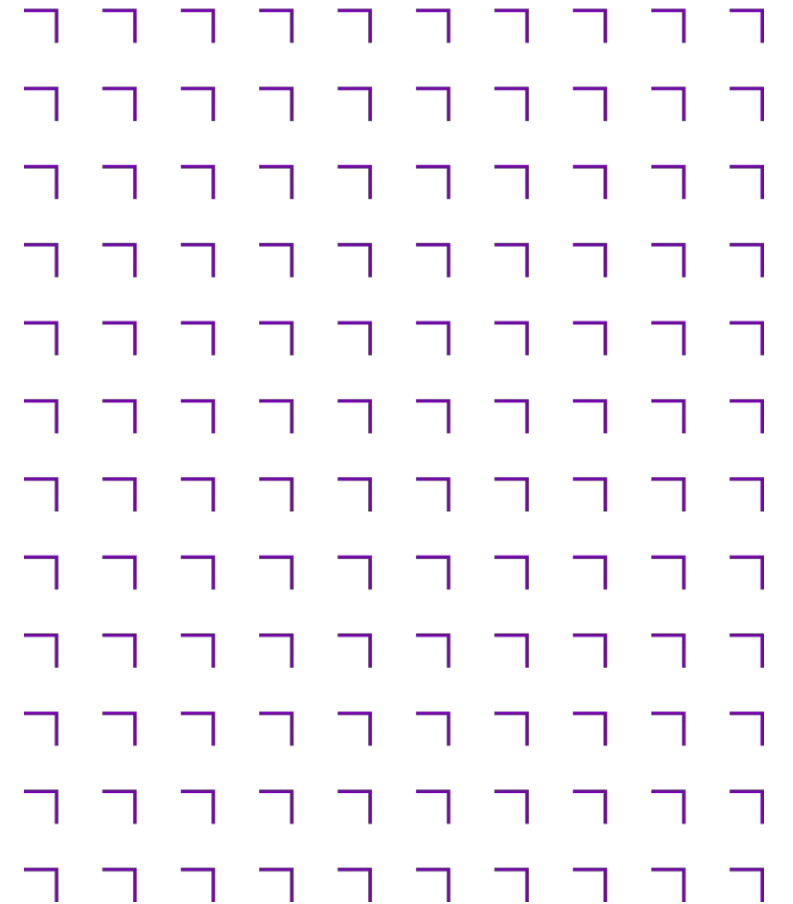
In This Unit



7. CSS - Advanced

Title
Gradients, Transforms, Transition, Animations
CSS architecture and BEM methodology
CSS Preprocessors and SASS

Gradients, Transforms, Transition, Animations



Round Corners

- **Border Radius**

```
p {
  border: 2px solid red;
  border-radius: 5px; /* set border radius for all four sides */
  /* or */
  border-top-left: 5px; /* can also be set individually */
  border-top-right: 5px;
  border-bottom-left: 5px;
  border-bottom-right: 5px;
}
```

-

Ellipses

You can specify different horizontal and vertical radii by splitting values with a “/”

```
div {
  background: #fff;
  width: 100px;
  height: 150px;
  border-radius: 50px/100px;
  border-bottom-left-radius: 50px;
  border-bottom-right-radius: 50px;
}
```

Shadows

Can be applied to both boxes and to text

♦ Box Shadows

`box-shadow` is the standard CSS property to get you going and it can have a value comprising several parts:

```
box-shadow: <horizontal-offset>,<vertical-offset>, <blur radius>, <spread distance>, <colour>
```

```
box-shadow: 5px 5px 3px 1px #999
```

- horizontal offset — how far the shadow is nudged to the right (or left if it's negative)
- vertical offset — how far the shadow is nudged downwards (or upwards if it's negative)
- blur radius (optional) — the higher the value the less sharp the shadow. ("0" being absolutely sharp)
- spread distance (optional) — the higher the value, the larger the shadow ("0" being the inherited size of the box)
- color (optional)

Shadows

- ◆ **Inner Box Shadows**

You can also apply shadows to the inside of a box by adding “inset” to the list:

```
box-shadow: inset 0 0 7px 5px #ddd;
```

- ◆ **Text Shadows**

You can also apply shadows to the outline of text:

```
text-shadow: -2px 2px 2px #999;
```

Similar to box-shadow except there is no spread distance or inset option for text-shadow

Advanced Colours

- **Alpha transparency**

RGBA opens up an exciting new dimension to web design, allowing you to set the transparency of a box or text. If you wanted a smidgen of a snazzy background image to peep through a heading, for example, you might use something like this:

```
h1 {
  padding: 50px;
  background-image: url(snazzy.jpg);
  color: rgba(0,0,0,0.8); /* adds an element of transparency */
}
```

Note: `rgba` can be used anywhere where `rgb` is used

Advanced Colours

- **Hue, saturation, and lightness**

Color names aside, web colors have always been red-green-blue biased, be that through hex codes or explicit RGB (or RGBA). Your brain is trained to break down colors into red, green and blue), HSL can be more intuitive because it gives you direct control over the aspects of a color's shade rather than its logical ingredients

```
#col1 { color: hsl(36, 100%, 50%) }
#col2 { color: hsla(36, 100%, 50%, 0.5) } /* adds an element of transparency */
```

Rather than each sub-value being a part of the color spectrum, however, they are:

- Hue (“36” above): Any angle, from 0 to 360, taken from a typical color wheel, where “0” (and “360”) is red, “120” is green and “240” is blue
- Saturation (“100%” in the example): How saturated you want the color to be, from 0% (none, so a level of grey depending on the lightness) to 100% (the whole whack, please).
- Lightness (“50%” in the example): From 0% (black) to 100% (white), 50% being “normal”.

So the example used here will produce an orange (36°) that is rich (100% saturation) and vibrant (50% lightness). It is the equivalent of #ff9900, #f90, and rgb(255, 153, 0)

At-rules

At-rules are CSS statements that instruct CSS how to behave. They begin with an at sign, '@' (U+0040 COMMERCIAL AT), followed by an identifier and includes everything up to the next semicolon, ';' (U+003B SEMICOLON), or the next CSS block, whichever comes first

♦ Regular

```
/* General structure */
@identifier (RULE);

/* Example: tells browser to use UTF-8 character set */
@charset "utf-8";
```

- @charset — Defines the character set used by the style sheet
- @import — Tells the CSS engine to include an external style sheet
- @namespace — Tells the CSS engine that all its content must be considered prefixed with an XML namespace

At-rules

- **Nested**

A subset of nested statements, which can be used as a statement of a style sheet as well as inside of conditional group rules

- `@media` — A conditional group rule that will apply its content if the device meets the criteria of the condition defined using a media query
- `@font-face` — Describes the aspect of an external font to be downloaded
- `@keyframes` — Describes the aspect of intermediate steps in a CSS animation sequence

```
@media print {
  body {
    font-size: 10pt;
    font-family: times, serif;
  }
  #navigation { display: none; }
}
@font-face {
  font-family: "font of all knowledge";
  src: url(fontofallknowledge.woff);
}
```

CSS Transitions

Transitions allow you to easily animate parts of your design without the need for the likes of JavaScript

A simple example: think of a traditional `:state`, where you might change the appearance of a link when a cursor hovers over it

```
a:link {  
  color: hsl(36,50%,50%);  
}  
a:hover {  
  color: hsl(36,100%,50%);  
}
```

This creates a binary animation; a link switches from a subdued orange to a rich orange when it is hovered over

CSS Transitions

The `transition` property, however, is much more powerful, allowing smooth animation (rather than a jump from one state to another). It is a shorthand property that combines the following properties:

- ♦ `transition-property` : which property (or properties) will transition, defaults to 'all' (optional)
- ♦ `transition-duration` : how long the transition takes
- ♦ `transition-timing-function` : if the transition takes place at a constant speed or if it accelerates and decelerates, defaults to 'ease' (optional)
- ♦ `transition-delay` : how long to wait until the transition takes place, defaults to '0' (optional)

```
a:link {
  transition: all .5s linear 0; /* all properties transitioned over half a second in a linear fashion with no delay */
  color: hsl(36,50%,50%);
}
a:hover {
  color: hsl(36,100%,50%);
}
```

CSS Transitions - Targeting specific properties

While “all” can be used in transition-property (or transition), you can tell a browser only to transition certain properties, by simply plonking in the property name you want to change. So the previous example could actually include `transition:`

`color .5s ease 0;` given only the color changes

If you want to target more than one property (without using “all”), you can offer up a comma-separated list with `transition-property` .

```
a:link {
  transition: .5s;
  transition-property: color, font-size;
  ...
}
```

Or you can offer up a slew of rules for transitioning each property like so:

```
a:link {
  transition: color .5s, font-size 2s;
  ...
}
```


CSS Transitions - Easing

The `transition-timing-function` CSS property sets how intermediate values are calculated for CSS properties being affected by a transition effect

It follows a cubic Bézier curve. `ease` produces a gradual acceleration at the start of the transition and a gradual deceleration at the end. `linear` maintains a constant speed throughout

Other values include `ease-in`, `ease-out`, `ease-in-out`, `step-start` and `step-end`

Gradients

CSS gradients let you display smooth transitions between two or more specified colors

CSS defines three types of gradients:

- Linear Gradients (goes down/up/left/right/diagonally)
- Radial Gradients (defined by their center)
- Conic Gradients (rotated around a center point)

Gradients - Linear

To create a linear gradient you must define at least two color stops

Color stops are the colors you want to render smooth transitions among

You can also set a starting point and a direction (or an angle) along with the gradient effect

Syntax

```
background-image: linear-gradient(direction/angle, color-stop1, color-stop2, ...);
```

Gradients - Linear

Direction	CSS
Top to bottom	<code>background-image: linear-gradient(red, yellow);</code>
Left to right	<code>background-image: linear-gradient(to right, red , yellow);</code>
Diagonal	<code>background-image: linear-gradient(to bottom right, red, yellow);</code>
Using angles	<code>background-image: linear-gradient(180deg, red, yellow);</code>
Multiple colour stops - V	<code>background-image: linear-gradient(red, yellow, green);</code>
Multiple colour stops - H	<code>background-image: linear-gradient(to right, red,orange,yellow,green,blue,indigo,violet);</code>
Transparency	<code>background-image: linear-gradient(to right, rgba(255,0,0,0), rgba(255,0,0,1));</code>
Repeating	<code>background-image: repeating-linear-gradient(red, yellow 10%, green 20%);</code>

Gradients - Radial

A radial gradient is defined by its center and requires at least two colour stops

Syntax

```
background-image: radial-gradient(shape size at position, start-color, ..., last-color);
```

By default, shape is ellipse, size is farthest-corner, and position is center

Gradients - Radial

	CSS
Evenly Spaced Color Stops (this is default)	<code>background-image: radial-gradient(red, yellow, green);</code>
Differently Spaced Color Stops	<code>background-image: radial-gradient(red 5%, yellow 15%, green 60%);</code>
Set Shape	<code>background-image: radial-gradient(circle, red, yellow, green);</code>
Repeating	<code>background-image: repeating-radial-gradient(red, yellow 10%, green 15%);</code>

The size parameter defines the size of the gradient: closest-side, farthest-side, closest-corner, farthest-corner

```
background-image: radial-gradient(closest-side at 60% 55%, red, yellow, black);
background-image: radial-gradient(farthest-side at 60% 55%, red, yellow, black);
```

Gradients - Conic

A conic gradient is a gradient with color transitions rotated around a center point and requires at least two colours

Syntax

```
background-image: conic-gradient([from angle] [at position,] color [degree], color [degree], ...);
```

By default, angle is 0deg and position is center

If no degree is specified, the colors will be spread equally around the center point

Gradients - Conic

	CSS
Three colours	background-image: conic-gradient(red, yellow, green);
Five colours	background-image: conic-gradient(red, yellow, green, blue, black);
Three colours and degrees	background-image: conic-gradient(red 45deg, yellow 90deg, green 210deg);
Pie chart 1	background-image: conic-gradient(red, yellow, green, blue, black); border-radius: 50%;
Pie chart 2	background-image: conic-gradient(red 0deg, red 90deg, yellow 90deg, yellow 180deg, green 180deg, green 270deg, blue 270deg);border-radius: 50%;
With Specified From Angle	background-image: conic-gradient(from 90deg, red, yellow, green);

Transform

The `transform` CSS property lets you rotate, scale, skew, or translate an element. That is, all elements whose layout is governed by the CSS box model except for: non-replaced inline boxes, `table-column` and `table-column-group` boxes. It modifies the coordinate space of the CSS [visual formatting model](#)

Common transform function include:

- ♦ `rotate`
- ♦ `scale`
- ♦ `skew`
- ♦ `translate`

Transform - Examples

```
.note {
  width: 300px;
  height: 300px;
  background: hsl(36,100%,50%);
  transform: rotate(-10deg); /* turns the box and it's content ten degrees anti-clockwise */ transform:
  skew(20deg,10deg); /* tip over the x-axis by 20 degrees on the y-axis by 10 degrees */ transform:
  skew(20deg); /* which is the equivalent of skew(20deg,0) */

  transform: scale(2); /* increase the size of the box and it's contents by two. */
                      /* Can be any positive number. Values less than one will shrink the box */

  transform: scale(1,2); /* can also scale the x and y dimensions separately */

  transform: translate(100px,200px); /* move the box 100 pixels to the right and 200 pixels down */
                                     /* Similar to position: relative; left: 100px; top: 200px; */
}
```

If you want to target an individual axis, you can also use `skewX`, `skewY`, `scaleX`, `scaleY`, `translateX`, and `translateY`

Transform - Combining transformations

Transform functions can also be combined

```
transform: rotate(-10deg) scale(2);
```

The order of the values is important - the latter will be executed before the former. It is, therefore, different to `transform: scale(2) rotate(-10deg);`, which will be rotated and then scaled

Alternatively, you could use the `matrix` function. `matrix` does the whole lot - rotating ' skewing ' scaling ' and

translating . It takes six values:

```
transform: matrix(2,-0.35,0.35,2,0,0);
```

These values aren't entirely straightforward and involve maths that, if you really want to tackle (there are benefits not only in brevity but also in precision), it may be worth giving [the specs](#) a gander

CSS Animations

CSS animations make it possible to animate transitions from one CSS style configuration to another

Consist of two components:

- ♦ a style describing the CSS animation
- ♦ a set of keyframes that indicate the start and end states of the animation's style, as well as possible intermediate waypoints

Advantages

- ♦ They're easy to use for simple animations; you can create them without even having to know JavaScript
- ♦ The animations run well, even under moderate system load. Simple animations often perform poorly in JavaScript.
The rendering engine can use frame-skipping and other techniques to keep the performance as smooth as possible
- ♦ Letting the browser control the animation sequence lets the browser optimize performance and efficiency by, for example, reducing the update frequency of animations running in tabs that aren't currently visible

CSS Animations - configuration

Style the element you want to animate with the animation property or its sub-properties. This lets you configure the timing, duration, and other details of how the animation sequence should progress

```
p {
  animation: 3s infinite alternate slidein; /* short version */
  animation-duration: 3s;
  animation-name: slidein;
  animation-iteration-count: infinite; /* animation repeat continuously */
  animation-direction: alternate; /* move back and forth across the screen */
}
```

Multiple values can also be specified:

```
p {
  animation-name: fadeInOut, moveLeft300px, bounce;
  animation-duration: 2.5s, 5s, 1s;
  animation-iteration-count: 2, 1, 5;
}
```

The `fadeInOut` animation is assigned a duration of 2.5s and an iteration count of 2

CSS Animations - at-rule

Configure the actual appearance of the animation using the `@keyframes` at-rule

```
p {
  animation-duration: 3s;
  animation-name: slidein;
}

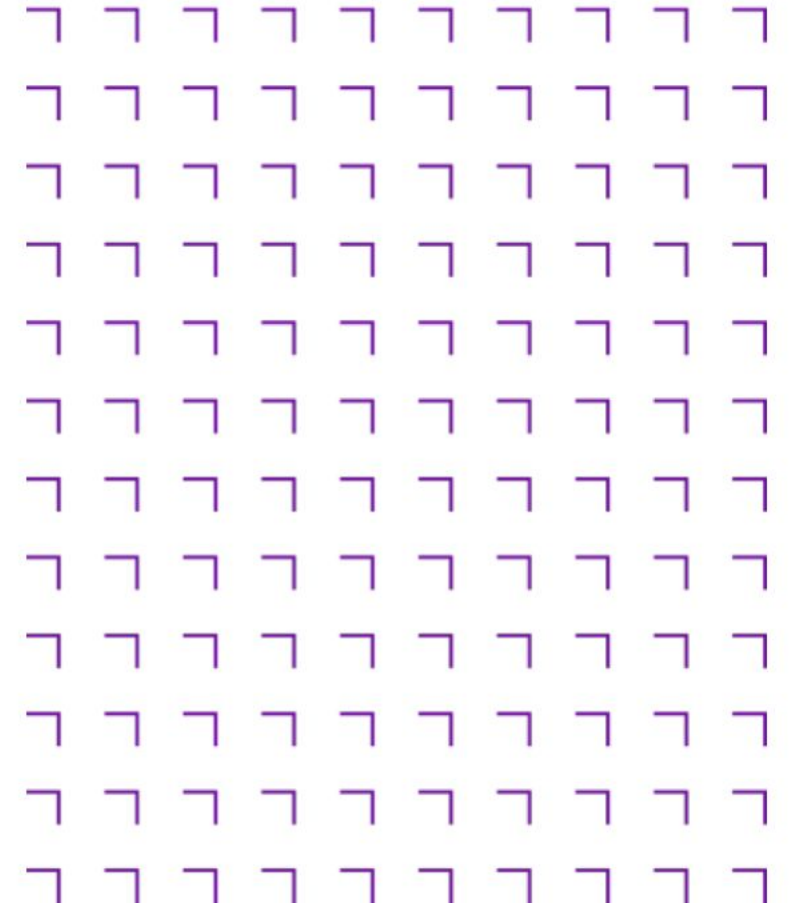
@keyframes slidein {
  from {
    margin-left: 100%;
    width: 300%;
  }
  to {
    margin-left: 0%;
    width: 100%;
  }
}
```

Activity

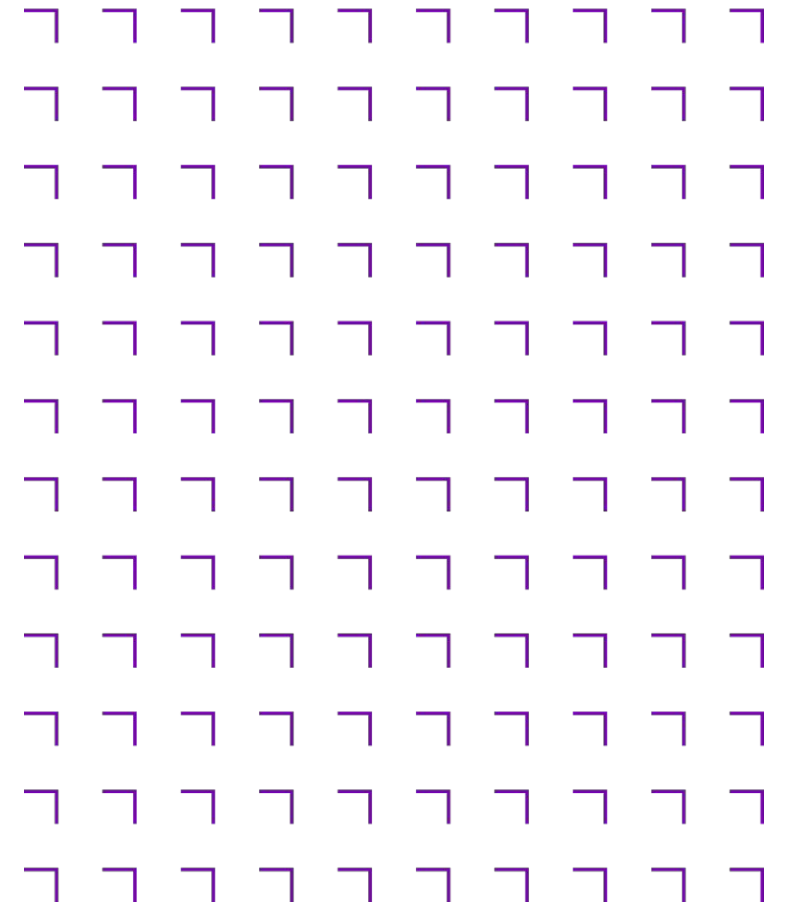


Breakout

- Join a breakout room
- Download the unit 7 exercises code from Moodle
- Follow the instructions and complete the exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour



CSS Architecture and BEM Methodology



CSS Architecture

CSS is notoriously difficult to manage in large, complex, rapidly-iterated systems

One reason is CSS lacks a built-in scoping mechanism

Everything in CSS is global any change you make has the potential to cascade and alter the presentation of unrelated bits of the UI

Extended CSS languages, a.k.a. **CSS preprocessors**, such as **Sass**, **Less** and **Stylus** make things a little easier by offering up features that make writing CSS easier

Until CSS gets its own native scoping mechanism, we need to devise our own system for locking down styles to specific sections of an HTML document

BEM is one such solution

BEM (Block, Element, Modifier)

BEM

BEM (Block, Element, Modifier) is a component-based approach to web development

The idea behind it is to divide the user interface into independent blocks

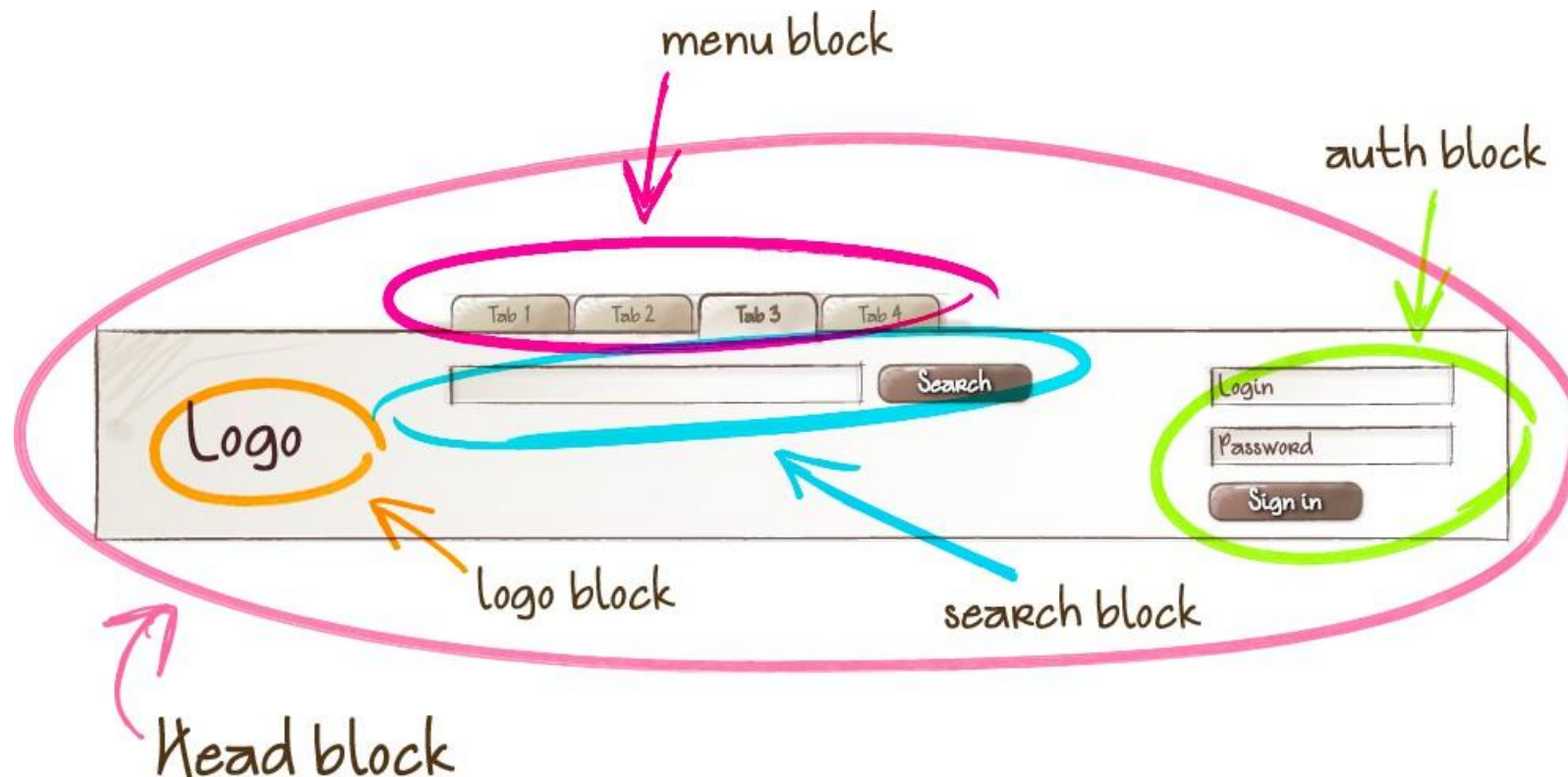
This makes interface development easy and fast even with a complex UI, and it allows reuse of existing code without copying and pasting

- ♦ Block
- ♦ Element
- ♦ Should I create a block or an element?
- ♦ Modifier
- ♦ Mix
- ♦ File structure

Blocks, elements, and modifiers are all called BEM entities

BEM - Block

- A logically and functionally independent page component, the equivalent of a component in Web Components
- A block encapsulates behavior (JavaScript), templates, styles (CSS), and other implementation technologies
- Blocks being independent allows for their re-use, as well as facilitating the project development and support process



BEM - Block

Features

- ♦ **Nested structure**

Blocks can be nested inside any other blocks

For example, a head block can include a logo (logo), a search form (search), and an authorization block (auth)

- ♦ **Arbitrary placement**

Blocks can be moved around on a page, moved between pages or projects

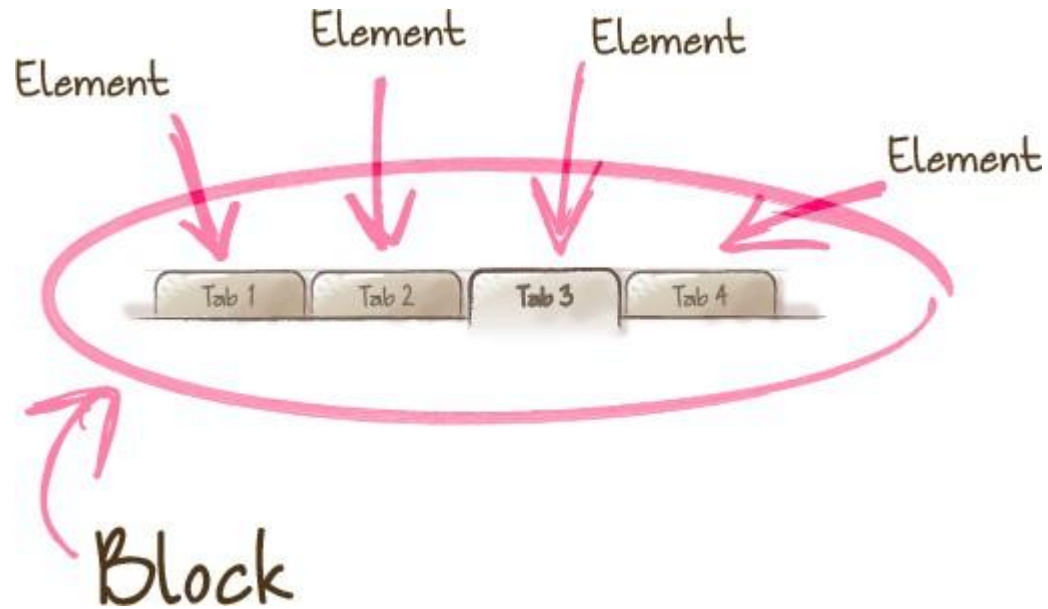
The implementation of blocks as independent entities makes it possible to change their position on the page and ensures their proper functioning and appearance

- ♦ **Re-use**

An interface can contain multiple instances of the same block

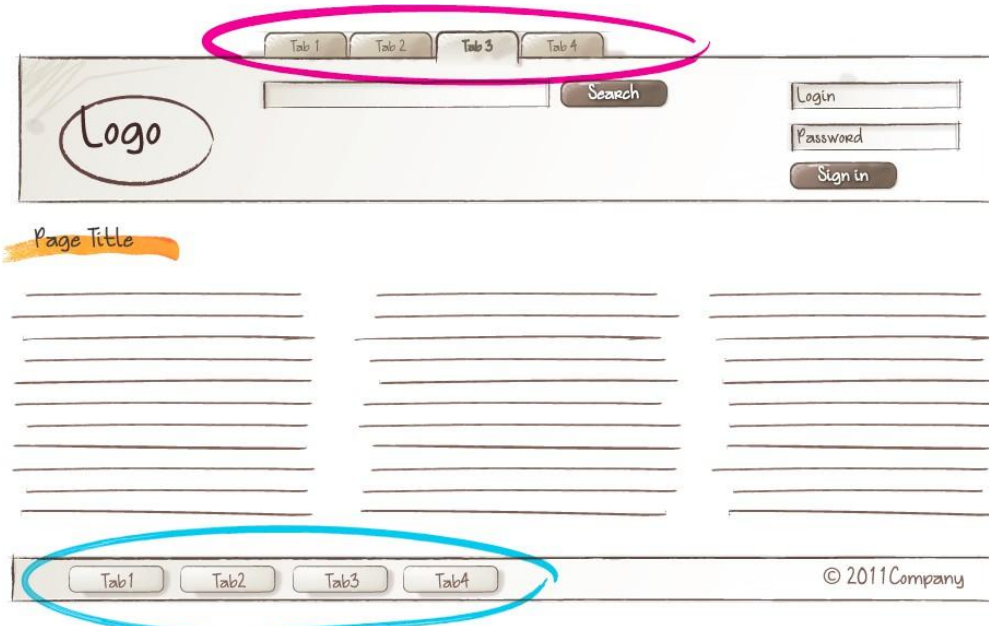
BEM - Element

- A constituent part of a block that can't be used outside of it
- For example, a menu item is not used outside of the context of a menu block, therefore it is an element



BEM - Modifier

- A BEM entity that defines the appearance and behaviour of a block or an element. Usage is optional
- Modifiers are similar in essence to HTML attributes. The same block looks different due to the use of a modifier
- For instance, the appearance of the menu block may change depending on a modifier that is used on it
- Modifiers can be changed in runtime (for example, as a reaction to a DOM event of the block), or via other blocks
- For example, if the wrong credentials are entered when a user clicks the Sign In button (the 'click' DOM event), the 'visible' modifier is set on a hidden block with error messages



BEM - Mix

- ◆ An instance of different BEM entities being hosted on a single DOM node
- ◆ Mixes allow us to:
 - Combine the behaviours and styles of several BEM entities while avoiding code duplication
 - Create semantically new interface components on the basis of existing BEM entities
- ◆ Let's consider the case of a mix comprising a block and an element of another block
- ◆ Let's assume that links in your project are implemented via a link block. We need to format menu items as links
- ◆ There are several ways to do that
 - Create a modifier for a menu item that turns the item into a link. Implementing such a modifier would necessarily involve copying the behaviour and styles of the link block. That would result in code duplication
 - Have a mix combining a generic link block and a link element of a menu block. A mix of the two BEM entities will allow us to use the basic link functionality of the link block and additional CSS rules of the menu block without copying the code

BEM tree

- A representation of a web page structure in terms of blocks, elements, and modifiers. It is an abstraction over a DOM tree that describes the names of BEM entities, their states, order, nesting, and auxiliary data
- In real-life projects, a BEM tree can be presented in any format that supports the tree structure, for example the DOM:

```
<header class="header">
  <img class="logo">
  <form class="search-form">
    <input class="input">
    <button class="button"></button>
  </form>
  <ul class="lang-switcher">
    <li class="lang-switcher item">
      <a class="lang-switcher link" href="url">en</a>
    </li>
    <li class="lang-switcher item">
      <a class="lang-switcher link" href="url">ga</a>
    </li>
  </ul>
</header>
```


BEM tree

The corresponding BEM tree will look like this:

```

  \ \
header
  logo
    search-form
      input
      button
  lang-switcher
    lang-switcher item
      lang-switcher
      link
    lang-switcher item
      lang-switcher
      link
  \ \

```

BEM - Naming convention

The name of a BEM entity is unique. The same BEM entity always has the same name in all technologies (CSS, JavaScript, and HTML). The primary purpose of the naming convention is to give names meaning so that they are as informative as possible for the developer

```
block-name    elem-name_mod-name_mod-val
```

- Names are written in lowercase Latin letters
- Words are separated by a hyphen (-)
- The block name defines the namespace for its elements and modifiers
- The element name is separated from the block name by a double underscore ()
- The modifier name is separated from the block or element name by a single underscore (_)
- The modifier value is separated from the modifier name by a single underscore (_)
- For boolean modifiers, the value is not included in the name

Important: Elements of elements do not exist in the BEM methodology. The naming rules do not allow creating elements of elements, but you can nest elements inside each other in the DOM tree

BEM - Examples

In HTML, BEM entities are represented by the class attribute. In BEM, for any of the technologies, there is a call to the class:

- ♦ **Menu**

```
<div class="menu">...</div>
```

```
.menu { color: red; }
```

- ♦ **Element**

```
<div class="menu">...
  <span class="menu item"></span>
</div>
```

```
.menu item { color: red; }
```

BEM - Examples

- ♦ **Block modifier name**

```
<div class="menu menu_hidden"> ... </div>
<div class="menu menu_theme_islands"> ... </div>
```

```
.menu_hidden { display: none; }
.menu_theme_islands { color: green; }
```

- ♦ **Element modifier name**

```
<div class="menu">
  ...
  <span class="menu item menu_item_visible menu_item_type_radio"> ... </span>
</div>
```

```
.menu_item_visible {}
.menu_item_type_radio { color: blue; }
```

BEM - Alternative naming schemes

There are alternative solutions that are actively used in the BEM community. To have all technologies apply identical names that were created using alternative naming schemes, use the bem-naming tool. By default, bem-naming is configured to use the methodology's standard naming convention, but it allows you to add rules so you can use alternative schemes.

Two Dashes style

```
block-name    elem-name--mod-name--mod-val
```

- Names are written in lowercase Latin letters
- Words within the names of BEM entities are separated by a hyphen (-)
- The element name is separated from the block name by a double underscore ()
- Boolean modifiers are separated from the name of the block or element by a double hyphen (--)
- The value of a modifier is separated from its name by a double hyphen (--)

Important: A double hyphen inside a comment (--) may cause an error during validation of an HTML document

BEM - Alternative naming schemes

CamelCase style

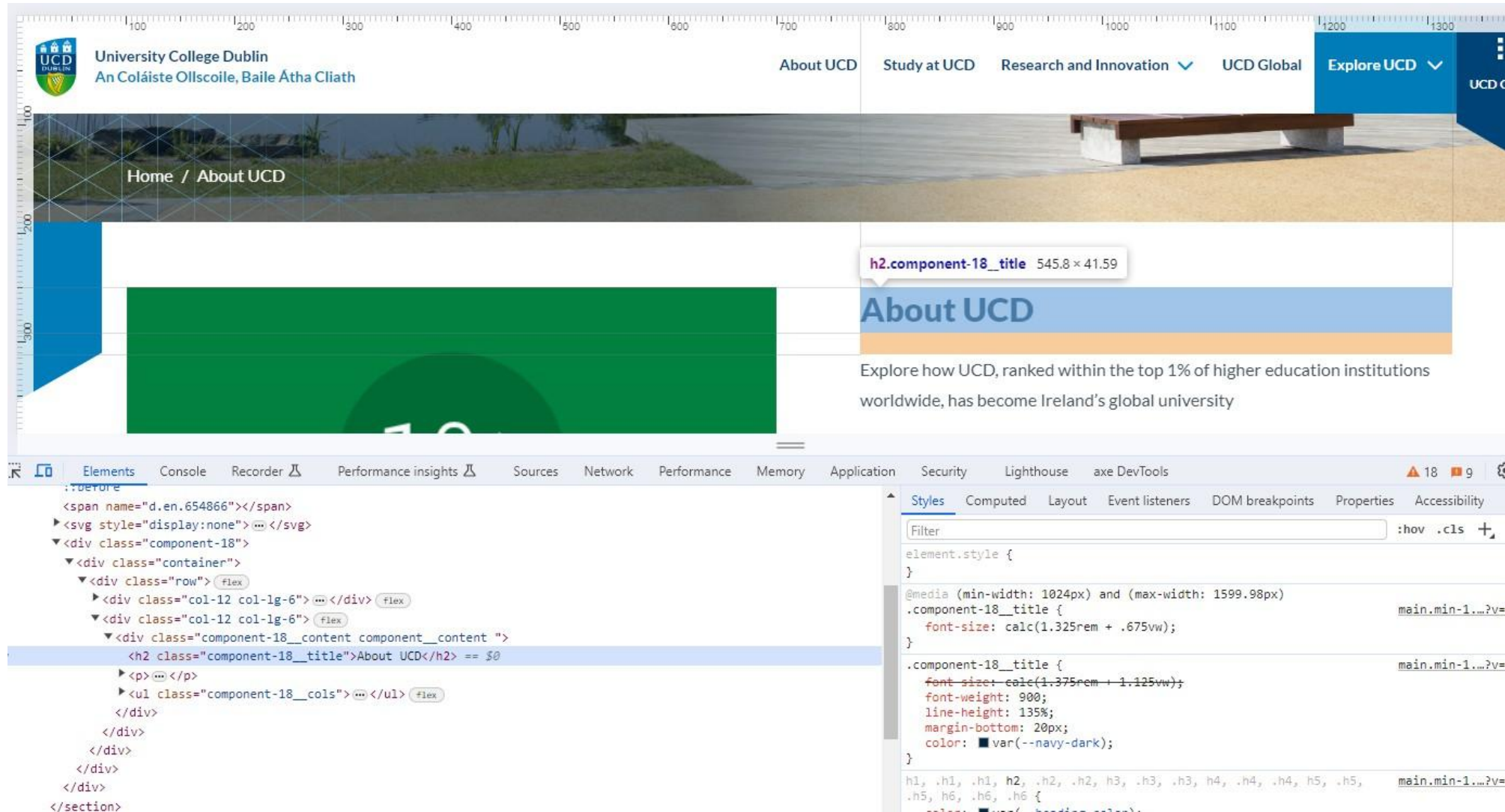
```
blockName-elemName_modName_modVal
```

- Names are written in Latin letters
- Each word inside a name begins with an uppercase letter
- The separators for names of blocks, elements, and modifiers are the same as in the standard scheme

Other styles include **React style** and **No Namespace style**

You can create your own custom naming solution for BEM entities. The most important thing is that your new naming system makes it possible to programmatically separate blocks from elements and modifiers

BEM - Example - UCD



The screenshot shows the UCD website with a ruler overlay indicating dimensions. The DevTools console shows the following HTML structure:

```

<span name="d.en.654866"></span>
<svg style="display:none"></svg>
<div class="component-18">
  <div class="container">
    <div class="row">
      <div class="col-12 col-lg-6">
        <div class="col-12 col-lg-6">
          <div class="component-18__content component__content">
            <h2 class="component-18__title">About UCD</h2>
            <p></p>
            <ul class="component-18__cols">
            </ul>
          </div>
        </div>
      </div>
    </div>
  </div>
</section>

```

The Styles panel shows the following CSS rules:

```

@media (min-width: 1024px) and (max-width: 1599.98px) {
  .component-18__title {
    font-size: calc(1.325rem + .675vw);
  }
}

.component-18__title {
  font-size: calc(1.375rem + 1.125vw);
  font-weight: 900;
  line-height: 135%;
  margin-bottom: 20px;
  color: var(--navy-dark);
}

h1, .h1, .h1, h2, .h2, h3, .h3, .h3, h4, .h4, .h4, h5, .h5,
.h5, h6, .h6, .h6 {
  font-size: 1.2em;
  font-weight: 400;
  line-height: 1.5;
  color: #000000;
}

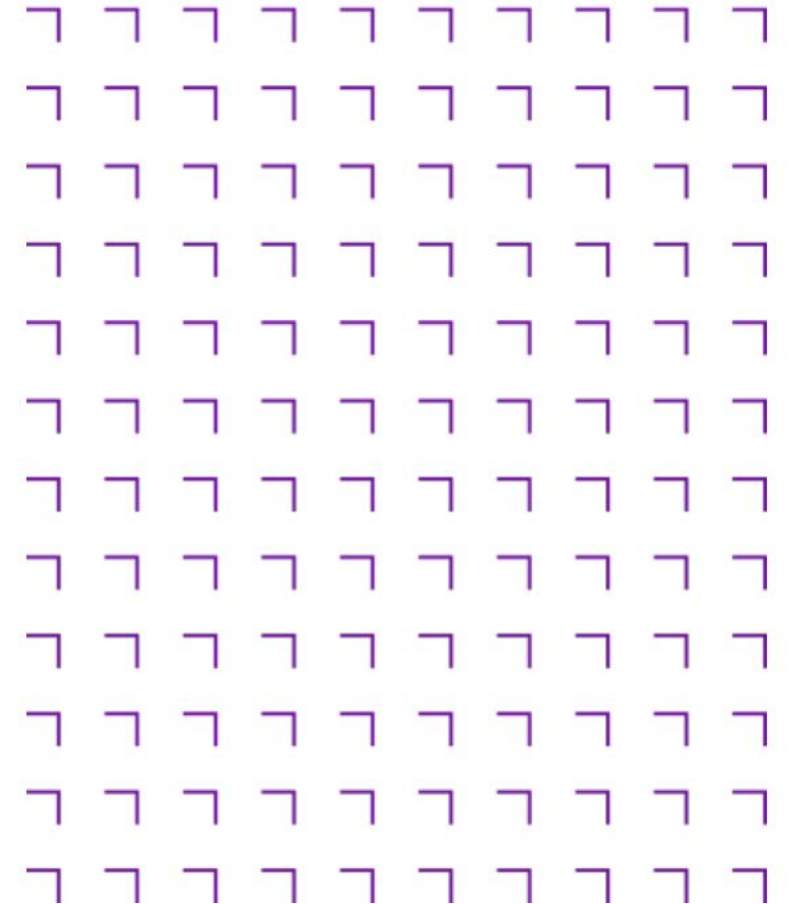
```



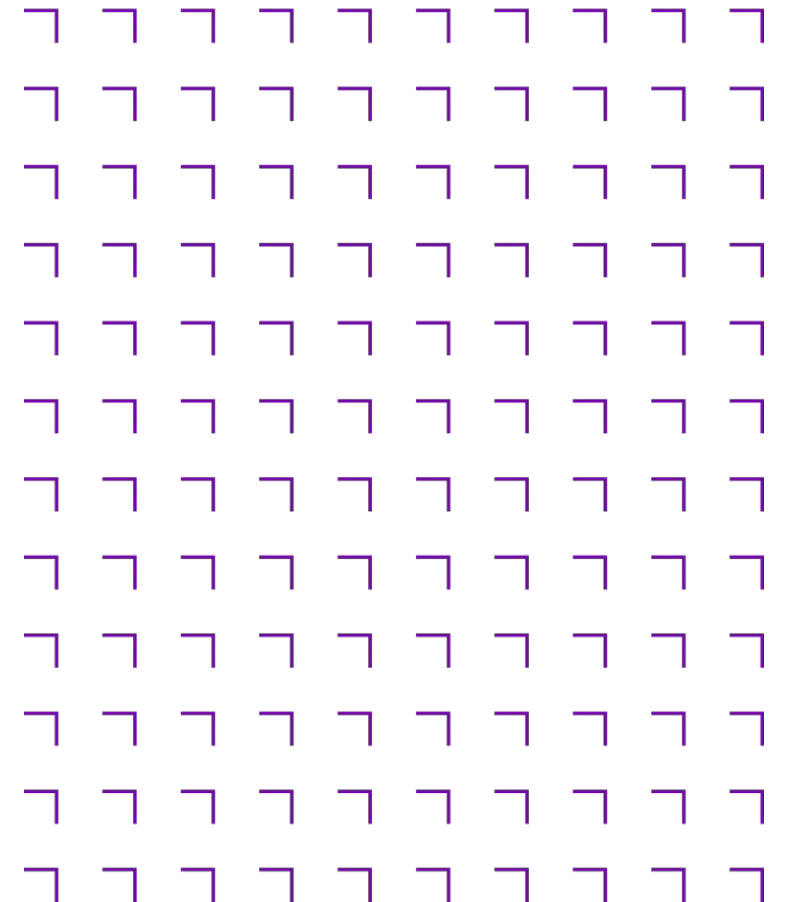
Activity

Breakout

- Join a breakout room
- Continue working on the unit 7 exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour



CSS Preprocessors and SASS



What are CSS Preprocessors

- Scripting languages that extend CSS functionality for cleaner, more powerful stylesheets
- Compiled into regular CSS for browser compatibility
- Overcome limitations of basic CSS in data management, code organization, and logic

Popular Preprocessors: Sass, LESS, and Stylus

- 80% similarity: All offer variables, mixins, nesting, media queries, loops & conditionals
- Key differences (20%) in advanced features:
 - Sass: `@extend`, `@media` within nesting, `@content` for flexible mixins, Ruby-based (frameworks: Gumby, Foundation)
 - LESS: Real-time browser compilation, mixins embedded in properties, Node.js based
 - Stylus: Minimal syntax (optional braces/colons), powerful in-language functions/conditionals, Node.js based

Note: The importance of preprocessors has decreased in recent years as new features get added to CSS

- [CSS Variables](#)
- [CSS Nesting](#)

Preprocessors - 5 Reasons to Use a CSS Preprocessor (SCSS examples)

- ◆ **Maintainability:**

- Declare variables for easy color/font changes:

```
$primary_color: #346699;  
$secondary_color: #769bc0;
```

- Use variables throughout your stylesheet:

```
a { color: $primary_color; }  
nav { background-color: $secondary_color; }
```

Preprocessors

- DRY (Don't Repeat Yourself): Share styles with `@extend`

```
.main-heading {  
  font-family: Tahoma, Geneva, sans-serif;  
  font-weight: bold;  
  font-size: 20px;  
  text-transform: uppercase;  
  color: blue;  
}  
.secondary-heading {  
  @extend .main-heading;  
  font-size: 16px;  
}
```

Preprocessors

- **Organization: Nest selectors for clear hierarchy**

Very important because it creates a visual hierarchy, similar to what you are used to with HTML. This results in less repetition of classes or divs needed since it is now in a cascade approach

```
h2 {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 20px;
  text-transform: uppercase;

  a {
    color: blue;

    &:hover {
      text-decoration: underline;
      color: green;
    }
  }
}
```

-

Time-Saving:

Less code duplication, more efficiency.

Preprocessors

- **Import**

The standard CSS `@import` allows you to split your css into multiple files

The problem with this is that it creates additional HTTP requests

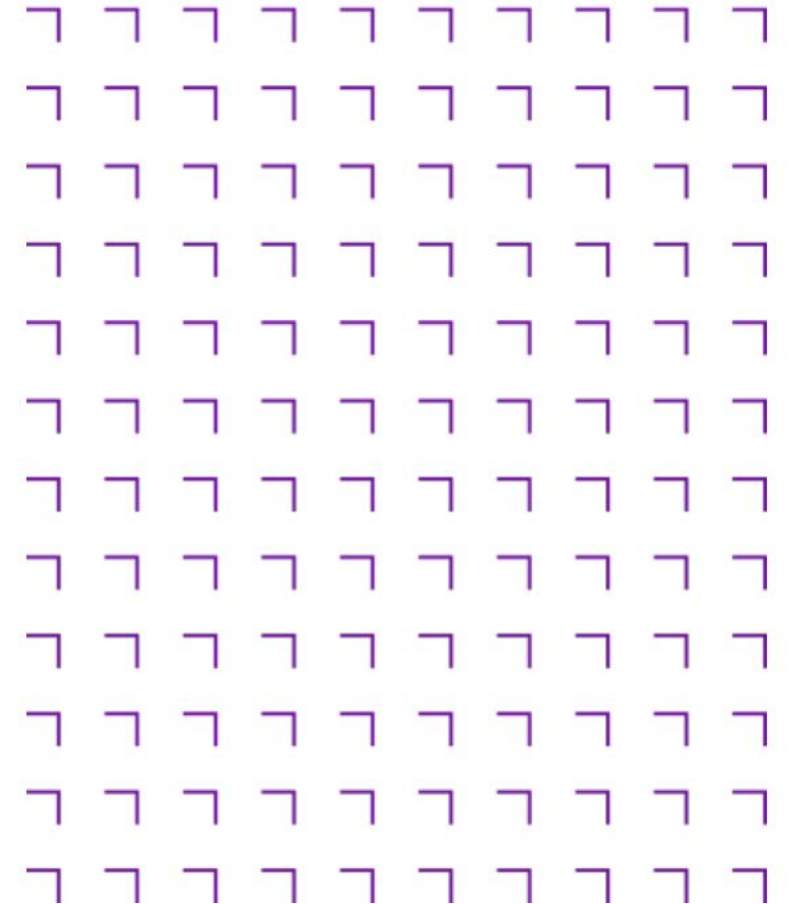
Sass and Less work a little different. Instead of creating another HTTP request, they combine the files into a smaller number of file(s). Similar to concatenation



Activity

Breakout

- Join a breakout room
- Continue working on the unit 7 exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour



Summary



Completed this Week

- Filters, Transforms, Transition, Animations
- CSS architecture and BEM methodology
- CSS Preprocessors and SASS

For Next Week

- Complete the remaining exercises for unit 7 before next class
- Review the slides and examples for unit 8

