DEEPer

CSS
Week 1 Session 2

CSS

Cascading StyleSheets

01

What is CSS?

- So far, we have seen how to define the structure of a page using HTML
- CSS allows us to alter the appearance of HTML elements
- Aspects such as element colour, size, visibility, and position can all be manipulated to define a page's aesthetics
- CSS code consists of multiple rules. Each rule contains;
  - Selector(s) – syntax defining the element(s) to target
  - Properties – the CSS style to effect, e.g. font-size
  - Values – one value per property, e.g. 16px

Selectors


```
#page-title       /* Element with an id of page-title */
.heading          /* All elements with a class of heading */
h2                /* All h2 elements */
h2#page-title     /* An h2 element with an id of page-title */
h2.heading        /* All h2 elements with a class of heading */

/* Target multiple selectors - comma separator */
#page-title, .heading, h2

/* Target all span elements within .heading - space separator */
.heading span
```

Selector Specificity

- There will often be multiple CSS rules that target a single element
- The browser needs to know which rule to use
- Allowing developers to override styles is an important feature
- Different selector types have different specificity scores

Selector Specificity


```
h1              /* Element: score of 1 */
.heading        /* Class: score of 10 */
#page-title     /* ID: score of 100 */

h1.heading      /* 1 + 10 = 11 */


p.body-text {
  color: blue;
}

.body-text {
  color: red;
}


<!-- Text will be blue as 11 > 10 -->
<p class="body-text">This is some body text...</p>
```

- This allows us to override generic styles in specific cases
- For example, make all headings red, but one specific heading green

Style Example


```
// Syntax example
.heading {
  font-size: 22px;
  color: red;
}
```

- The code preceding the { } is the selector(s)
- Property: value pairs for that selector go within

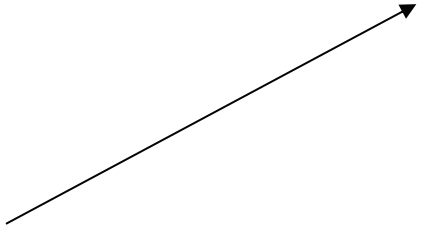
Including CSS

- CSS code can be included in page in one of two ways
- Added to the `<head/>` in a `<style>` element
- Loaded from an external .css file using a `<link/>` element

Including CSS


```html
<!DOCTYPE html>
<html>
    <head>
        <style>
            .head-style {
                color: red;
            }
        </style>
        <link href="example.css" rel="stylesheet">
    </head>
    <body>
        <p class="head-style">Styled in the head</p>
        <p class="included-style">Styled in an include</p>
    </body>
</html>
```

```css
.included-style {
    color: blue;
}
```



Typography

- Can be used to alter the appearance of text
- All elements *can* be styled to look like each other, but that doesn't mean that they *should*
- Choose the right tag for the right job – semantics!
- Fonts may not always be available, so provide a fallback
- Colours may be names or hexadecimal (#FF0000)

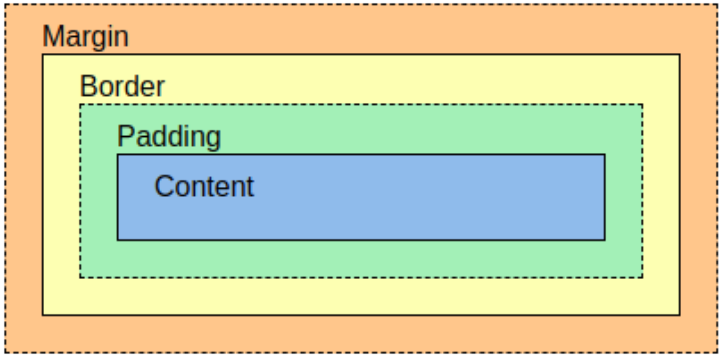
Typography


```
.my-class {
    color: red;
    font-family: "Arial", sans-serif;
    font-size: 2rem;
    font-weight: bold;
    letter-spacing: 0.1rem;
    line-height: 2.2rem;
    text-decoration: underline;
}
```

He's so **stylish** it hurts

Box Model

- Think of all elements as being encased within a box with 3 aspects
- Padding: the space inside the box, between content and border
- Border: the outline of the box
- Margin: the space outside the box (between elements)



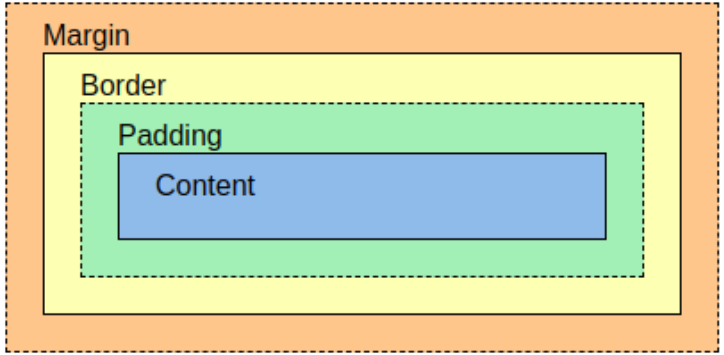
Defining Box Properties


```
/* The following properties function identically for Margin */

padding: 4px; /* All 4 sides will be 4px */
padding: 5px 10px; /* Top & Bottom 5px, Left & Right 10px */
padding: 5px 6px 7px 8px; /* Top 5px, Right 6px, Bottom 7px, left 8px */

padding-left: 5px; /* Set the left-side padding only. Also -right, -top and -bottom */
```

```
/* border: size type colour */
border: 2px solid red;
border-right: 1px dashed blue;
border-left: none;
```

Box Model


```
<style>
    p {
        display: inline-block;
        background: lightblue;
        margin: 5px;
        border: 2px solid red;
        border-right: 1px dashed blue;
        border-left: none;
        padding: 5px 10px;
        padding-right: 2px;
    }
</style>

<p>Paragraph 1</p>
<p>Paragraph 2</p>
<p>Paragraph 3</p>
```

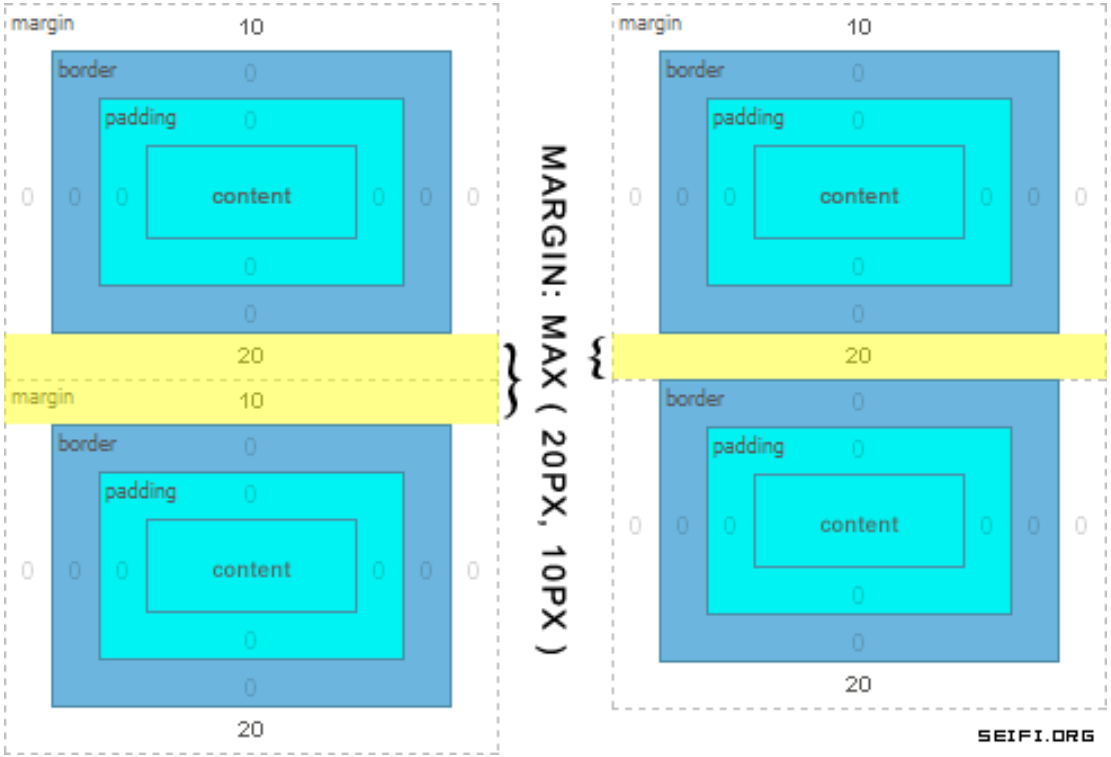


Paragraph 1 Paragraph 2 Paragraph 3

Collapsing Margins

- CSS has a quirk where margins can overlap each other
- This is visible in the previous example
- We'd expect a 5px margin on each element to result in a 10px gap between two elements
- Instead, they overlap and so only a 5px gap is present

Collapsing Margins



Containers

- Content usually doesn’t span a page’s full width – think huge monitors!
- Most pages will have a “container” which contains elements to a fixed width
- The container is most often centered on the page
- Sometimes pages have full-width sections which aren’t in the container (e.g. header / footer)

Container


```
<style>
    body {
        background: lightyellow;
        text-align: center;
        padding: 10px;
    }

    .container {
        width: 980px;
        background: lightcoral;
        text-align: left;
        margin: auto;
    }
</style>

<body>
    <div class="container">
        <h1>This is some content!</h1>
    </div>
</body>
```

Backgrounds

- Fills an element, from content to padding
- Can be a combination of image and colour
- Images can be repeated or stretched to fit
- Colours may be names or hexadecimal (#FF0000)
- The element won’t stretch to fit, so may need width / height specified
- Can be a gradient

Backgrounds


```
1   .cat {
2       background-image: url(http://www.randomkittengenerator.com/cats/rotator.php);
3       background-size: cover;
4       width: 250px;
5       height: 250px;
6   }
7
8   .green {
9       background-color: green;
10  }
11
12  .gradient {
13      background-image: linear-gradient(black, white);
14  }
```

Display

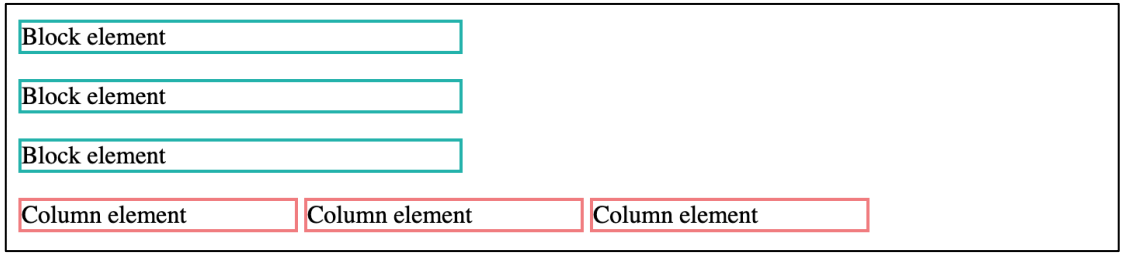
- The display property defines the way in which an element and its children renders in terms of layout
- For now we are interested in four values
- `none` – element is hidden
- `block` – expands to full width, with line-breaks before and after
- `inline` – renders in-line with other in-line elements
- `inline-block` – same as inline, but honours width, padding and margin

Display


```
.hidden {
    display: none;
}

.inline {
    display: inline;
}

.inline-block {
    display: inline-block;
    width: 30%;
    vertical-align: top;
}
```

Display


```
<style>
    .block {
        display: block;
        border: 2px solid lightseagreen;
    }

    .column {
        width: 25%;
        display: inline-block;
        border: 2px solid lightcoral;
    }
</style>

<body>
    <p class="block">Block element</p>
    <p class="block">Block element</p>
    <p class="block">Block element</p>

    <div class="column">Column element</div>
    <div class="column">Column element</div>
    <div class="column">Column element</div>
</body>
```



Display

- All HTML elements have a default display value
- We can override the value with CSS

```
p, div, h1, section, form   /* display: block */
span, strong, a             /* display: inline */
```

Sizing Elements

- There are a few units of measurement
- Up until now, we've seen "px" and "rem"
- There are two types of unit: absolute, and relative
- Absolute units are fixed sizes
- Relative units are variable based on the parent's values

Sizing - Absolute Units

- Less favourable because they don’t scale to screen size
- Physical measurements can be useful for print media (cm, mm, in)
- 1px = 1/96 of an inch
- 1pt = 1/72 of an inch

Sizing - Relative Units

- Scale to the size of the parent element
- % is relative to the parent’s value for the same property
- em is relative to the current element’s font size
- rem is relative to the root element’s font size (<html>)
- vh/vw are relative to the size of the viewport (screen)

Sizing - Relative Units


```
<style>
    .parent {
        font-size: 20px;
    }

    .child {
        font-size: 1.5em;
    }
</style>

<div class="parent">
    Parent Text

    <p class="child">Child text</p>
</div>
```

- em units are relative to the parent
- Child text will be 30px

Sizing – Relative Units


```
<style>
    html {
      font-size: 10px;
    }

    h1 {
      font-size: 2rem;
    }
</style>

<h1>Heading</h1>
```

- rem is relative to the root element's size
- h1 text will be 20px
- rem is useful to keep all font sizes relative to one value

Responsive CSS

- Responsive sites are those that reform themselves to work well on devices of multiple sizes
- This may include changing layout (e.g. stacking columns vertically on mobile)
- Some elements may also be removed altogether on smaller screens
- When developing for different devices, we need to consider the usability of solutions
- For example button sizes should be larger on touch screen devices

Responsive CSS

- When implementing responsive sites, we tend to develop “mobile first”
- Linear approach – elements will be top to bottom
- Elements which don’t need to be full width on larger displays can be adapted into columns
- Grids lend themselves to responsive layouts

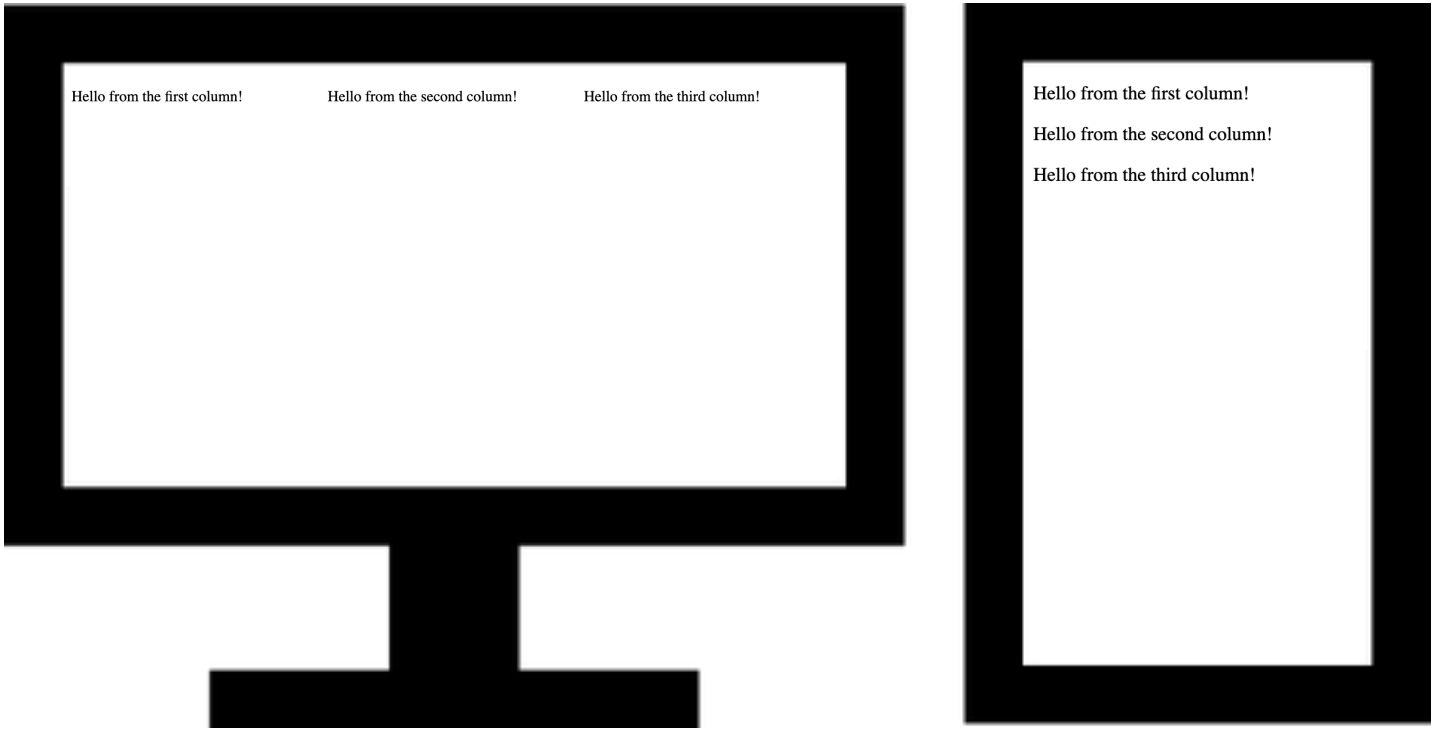
Responsive CSS - Requirements

- We must define a viewport in the head
- Media queries allow us to target different screens and mediums
- We can also target print media as well as different screen sizes
- There are “standard” media queries which allow us to target specific screen sizes and devices

Responsive CSS


```
<!DOCTYPE html>
<html>
    <head>
        <style>
            @media screen and (min-width: 1024px) {
                div.column {
                    width: 33%;
                    display: inline-block;
                }
            }
        </style>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div class="column">
            <p>Hello from the first column!</p>
        </div>
        <div class="column">
            <p>Hello from the second column!</p>
        </div>
        <div class="column">
            <p>Hello from the third column!</p>
        </div>
    </body>
</html>
```

Responsive CSS

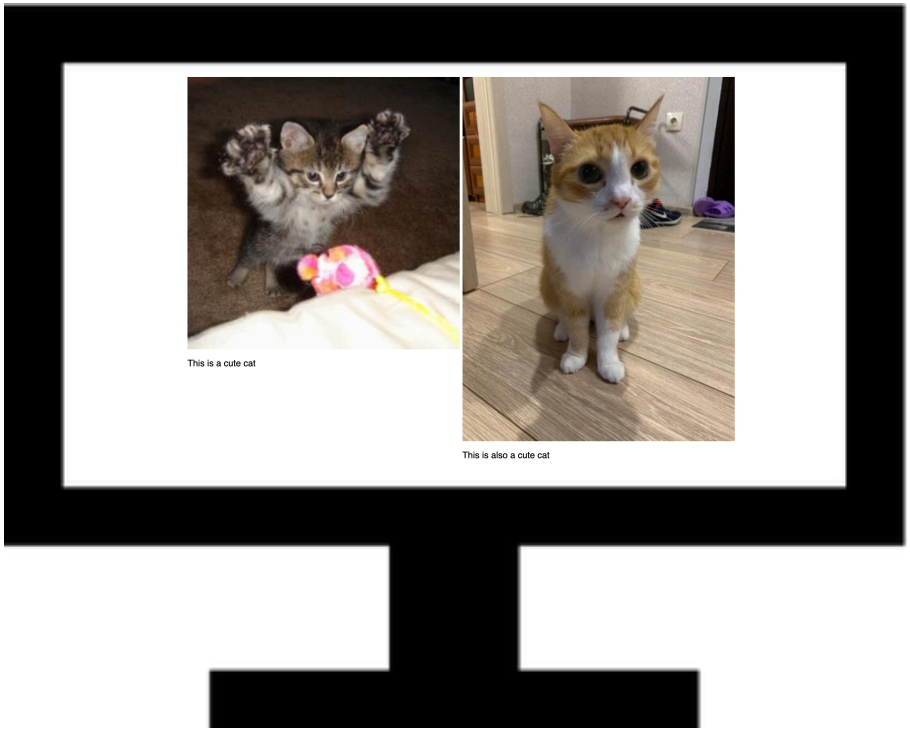


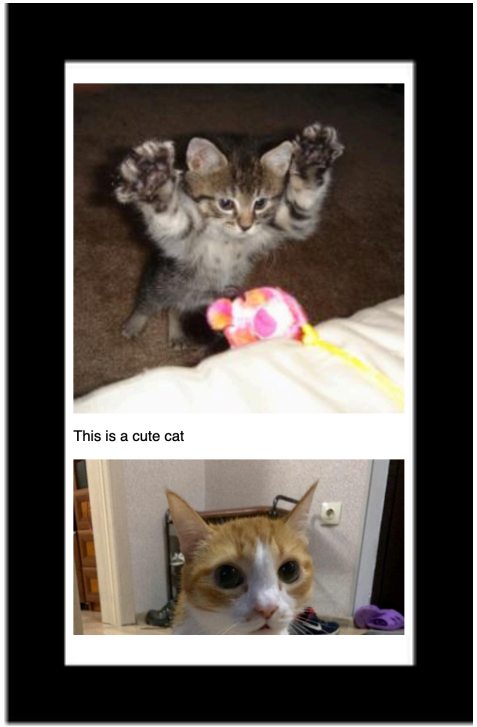
Responsive CSS


```css
body {
  text-align: center;
  font-size: 16px;
  font-family: sans-serif;
}

.container {
  text-align: left;
  margin: auto;
}

img {
  width: 100%;
  display: block;
}

@media screen and (min-width: 768px) {
  .col {
    width: 49%;
    display: inline-block;
    vertical-align: top;
  }
}

@media screen and (min-width: 1024px) {
  .container {
    width: 980px;
  }
}
```

```html
<!DOCTYPE html>
<html>
  <head>
    <link href="responsive.css" rel="stylesheet">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
  </head>
  <body>
    <div class="container">
      <div class="col">
        <img src="http://www.randomkittengenerator.com/cats/rotator.php" alt="cat">
        <p>This is a cute cat</p>
      </div>
      <div class="col">
        <img src="https://cataas.com/cat" alt="cat">
        <p>This is also a cute cat</p>
      </div>
    </div>
  </body>
</html>
```

Responsive CSS





Pseudo Classes & Elements

- CSS allows more complex selectors (e.g. hover state and links which have already been visited)
- These allow us to create more interactive elements
- We can also “inject” additional elements before or after an element
- Injected elements need a “content” property to be displayed

Pseudo Classes & Elements


```
a {
    color: black;
    text-decoration: none;
}

a:hover {
    color: red;
}

a::after {
    content: " ";
    background-image: url(link-icon.png);
    background-size: contain;
    display: inline-block;
    width: 1em;
    height: 1em;
    margin-left: 5px;
}
```

Click Here 

CSS Animations

- It’s possible with CSS to animate any styled aspects of an element
- This is commonly used to smoothly change a CSS property during a pseudo class event
- To animate, we can use the `transition` property

CSS Animations


```
<!DOCTYPE html>
<html>
  <head>
    <style>
      a {
        color: red;
      }

      a.transition {
        transition: .5s ease-in;
      }

      a:hover {
        color: blue;
      }
    </style>
  </head>
  <body>
    <a href="#">Click Here</a><br>
    <a href="#" class="transition">Transitioned</a>
  </body>
</html>
```

Custom CSS Animations

- Instead of using a pseudo class, we can also define our own CSS animations which will permanently run (by default) when the element is first rendered
- These can also be delayed, repeated, and have their timings customised
- Custom animations can be defined using the `@keyframes` syntax

Custom CSS Animations


```css
@keyframes glow-from-to {
  from { color: red; }
  to { color: blue; }
}

@keyframes glow-percentage {
  0%, 100% { color: red; }
  50% { color: blue; }
}

a.glow-from-to {
  animation-name: glow-from-to;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}

a.glow-percentage {
  animation-name: glow-percentage;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}
```

Click Here
Click Here

CSS Animations: Resources

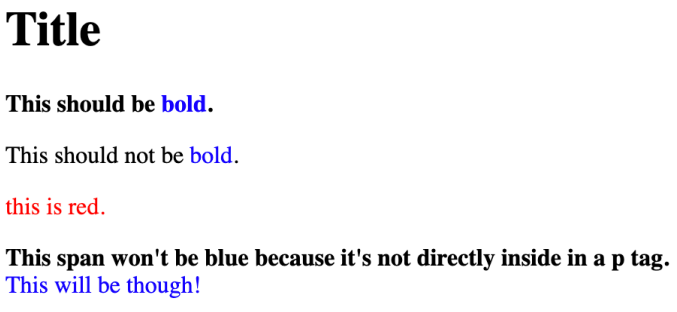
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations
- https://www.w3docs.com/learn-css/animation.html
- https://css-tricks.com/almanac/properties/a/animation/

Advanced Selectors

- On top of the selectors we saw earlier, there are also three useful options available in CSS:
- Universal selector (*****) - Selects **all** elements
- Child selector (**>**) - Selects elements **directly inside** an element
- Adjacent selector (**+**) – Selects the next element **after** an element

Advanced Selectors


```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div.red * { color: red; } /* all elements inside a div.red */
      p > span { color: blue; } /* any span tag DIRECTLY inside a p tag */
      h1 + p { font-weight: bold; } /* a p tag immediately after an h1 tag */
    </style>
  </head>
  <body>
    <h1>Title</h1>
    <p>This should be <span>bold</span>.</p>
    <p>This should not be <span>bold</span>.</p>
    <div class="red">
      <p>this is <span>red<span>.</p> <!-- The red is taking precedence here so no blue! -->
    </div>
    <p>
      <strong><span>This span won't be blue because it's not directly inside in a p tag.</span></strong>
      <br>
      <span>This will be though!<span>
    </p>
  </body>
</html>
```

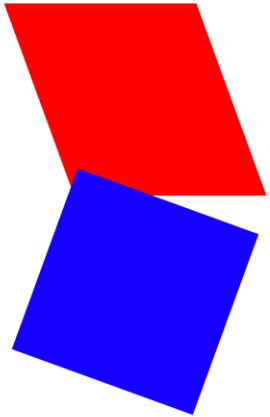


CSS Transformations

- It's possible to *transform* certain elements using various functions
- This property allows us to transform the shape of an element
- Available functions include:
  - rotate()
  - translate()
  - skew()
  - matrix()

CSS Transformations


```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        padding: 50px;
      }

      div {
        width: 100px;
        height: 100px;
      }

      div.skew {
        transform: skew(20deg);
        background: red;
      }

      div.rotate {
        transform: rotate(20deg);
        background: blue;
      }
    </style>
  </head>
  <body>
    <div class="skew"></div>
    <div class="rotate"></div>
  </body>
</html>
```



CSS Filters

- It’s possible to manipulate an element (including images) using CSS
- The `filter` property has several useful functions to achieve this
- Available functions include:
  - blur()
  - brightness()
  - contrast()
  - hue-rotate()

CSS Filters


```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="13-Filters.css">
  </head>
  <body>
    <div></div> <!-- plain cat! -->
    <div class="blur">blur</div>
    <div class="hue-rotate">hue-rotate</div>
    <div class="contrast">contrast</div>
    <div class="grayscale">grayscale</div>
  </body>
</html>
```

```css
div {
  background-image: url(https://placekitten.com/300/300);
  width: 300px;
  height: 300px;
  float: left;
  color: red;
  font-weight: bold;
}

div.blur {
  filter: blur(5px); /* specify a pixel value */
}

div.hue-rotate {
  filter: hue-rotate(45deg); /* rotate up to 360 degrees */
}

div.contrast {
  filter: contrast(150%); /* percentage, allowed over 100% */
}

div.grayscale {
  filter: grayscale(100%); /* percentage, up to 100% */
}
```

Inspecting CSS in the Browser

- CSS can be viewed and altered within the browser
- These changes won't be saved, but offer a glimpse of what a change might look like
- Can be done on any web page – although some styles might override yours!
- Useful for debugging and tweaking things live before copying the styles into your codebase

Inspecting CSS in the Browser