CS1115/CS5002

Web Development 1

Dr Derek Bridge

School of Computer Science & Information Technology

University College Cork

Multicolumn text

- In some rare cases, you have a large quantity of text
- It might look better if the text is laid out in columns, like in a newspaper
- In CSS, on the container, specify one of the following:
  - `column-count`: number of columns you'd like to break the content into
  - `column-width`: your preferred column width, leaving it to the browser to figure out how many columns to use
    - E.g. `column-width: 20em`: you get as many 20em columns as will fit; any remaining space is shared between the columns, so they end up being *at least* 20em (unless there was room for only one column, in which case it might be less than 20em)
    - Note how this is responsive — without the need for media queries!
- In fact, the content doesn't have to be text: there can be images, headings, and other elements

Multicolumn text example

```html
<body>
    <header>
        <h1>Header</h1>
    </header>
    <main>
        <h1>Heading</h1>
        <p>
            Lots of text
        </p>
        <figure>
            <img src="..." alt="..." />
            <figcaption>
                Caption
            </figcaption>
        </figure>
        <p>
            Lots more text.
        </p>
    </main>
    <footer>
        Footer
    </footer>
</body>
```

```css
main {
    column-width: 20em;
    column-gap: 2em;
    column-rule: 0.0625em solid gray;
}

main h1 {
    column-span: all;
}

figure {
    break-inside: avoid;
}

img {
    max-width: 100%;
}
```

Multicolumn text: more control

- You cannot style the individual columns but you can specify the size of the gap and whether you want a line to separate the columns
- A descendant of the container can span the columns, using `column-span: all`
- Controlling content breaks
  - Fragmentation: we probably don't want a heading to be the last thing in a column; we probably don't want an image in one column and its caption in another; ...
  - Try `break-before`, `break-after` and `break-inside` with values that include `always` and `avoid`
- It is tempting to put a height onto the container. But, if you do, then content may overflow, resulting in a horizontal scrollbar
- You can, of course, combine multicolum with media queries

Hyphenation

- What can we do if a word is too long for its column?
- For now:
  - HTML: put `&shy;` inside the word at possible places where it can be hyphenated
- In future:
  - HTML: make sure the container has a `lang` attribute
  - CSS: set `hyphens: auto` on its container

CSS shapes

- CSS shapes allows content to flow around a non-rectangular shape
- (In the future, it might be possible to have text flow *within* a shape)
- The element around which the content flows
  - must be floated left or right and have a width and height
  - must have the `shape-outside` property whose values include `circle`, `ellipse`, `inset` (= rectangle) or `polygon`

CSS shape example

```
<section id="seashells">
    <div id="#shape">
    </div>
    <p>
        Text about seashells.
    </p>
    <p>
        More text about seashells.
    </p>
</section>
```

Compare this:

```
#shape {
    float: left;
    height: 10em;
    width: 10em;
}
```

with this:

```
#shape {
    float: left;
    height: 10em;
    width: 10em;
    shape-outside: circle();
}
```

CSS shapes: more control

- By default, the circle's radius will be 50% of the element's width, and its centre will be at 50%, 50% of the element
- Change the radius: `shape-outside: circle(20%);`
- Change the radius and position of the the centre: `shape-outside: circle(20% at 30% 70%);`
- **Gotcha**: if you make the shape bigger than the element, the content flows around the element, not the shape
- **Q:** So, as a percentage, what's the largest that will have any effect?
- We can specify a margin, e.g. `shape-margin: 2em` but, without great care, this often results in the gotcha

CSS shapes: generated content

- The problem with the example is the `div` spoils our HTML. **Q:** In what way?
- To avoid this, people make use of CSS **generated-content**

```
<section id="seashells">
    <p>
        Text about seashells.
    </p>
    <p>
        More text about seashells.
    </p>
</section>
```

```
#seashells::before {
    content: "";
    float: left;
    height: 10em;
    width: 10em;
    shape-outside: circle();
}
```

- Or they might use, e.g., an `img` in place of the `div`

```
<section id="seashells">
    <img src="seashell.png" alt="A seashell" />
    <p>
        Text about seashells.
    </p>
    <p>
        More text about seashells.
    </p>
</section>
```

```
img {
    float: left;
    shape-outside: circle();
}
```

- You can even use the alpha channel of an image to create the shape: `shape-outside: url("seashell.png")` (Advanced: provided the image is CORS-compatible, i.e. roughly, provided it's on your web server)

CSS custom properties

- You are familiar with **variables** in Python
- CSS has something similar to variables, called *Custom Properties*
    - They help us to avoid repetition in the stylesheet
    - They can give meaningful names to complex properties to improve readability
    - They can be set by JavaScript, e.g. for theme switching
- Example of defining a custom property:

```
html {
    --main-colour: green;
}
```

    - They must be defined within a CSS rule
    - Their name must begin with `--`
    - Unlike the rest of CSS, their name is case-sensitive
- Example of using the value of a custom property:

```
header > h1 {
    color: var(--main-color);
}
```

    - The `var` function retrieves the value of the custom property
    - You can use this after the colon, where a CSS property would go, but nowhere else

CSS custom properties

- The value can be any CSS property, e.g.:

```
html {
    --main-color: #0066cc;
    --main-padding: 1em 2em;
    --main-font: 16px;
    --font-factor: 1.5;
    --main-bckgrnd-img: url("background1.jpg");
}
```

- You can do calculations with the value using the `calc` function, e.g.:

```
header > h1 {
    font-size: calc((var(--main-font)) * (var(--font-factor)));
```

CSS custom properties

- Custom properties are like normal properties: inheritance, overriding of inheritance and conflict resolution work in the usual way

```
<body>
    <main>
        <h1>Seashells</h1>
        <section>
            <h1>Some facts</h1>
            <p>
                A seashell …
            </p>
            <p>
                A seashell …
            </p>
        </section>
        <section id="warning">
            <h1>Warning</h1>
            <p>
                Do not eat the seashells.
            </p>
        </section>
    </section>
</body>
```

```
html {
    --text-color: blue;
}

#warning {
    --text-color: red;
}

p {
    color: var(--text-color);
}
```

(Many people use `:root` instead of `html` as the top-level selector)

CSS transforms

- CSS transforms allow elements to be transformed in two- or three-dimensional space
- E.g. we can double the size of an image:

```
img {
    transform: scale(2);
}
```

- E.g. we can rotate it (in this case, 30 degrees clockwise):

```
img {
    transform: rotate(30deg);
}
```

- Or we can do both:

```
img {
    transform: scale(2) rotate(30deg);
}
```

- There are a couple of dozen more transforms

Hovering

- Transforms are often combined with hover effects
- E.g.

```
img:hover {
    transform: scale(2) rotate(30deg);
}
```

- Consider this:

```
img:hover {
    transform: rotate(360deg);
}
```

Q: Why don't we see anything?

CSS transitions

- CSS transitions allow changes in property values to occur smoothly over a specified duration
- The `transition` property takes four values:
  - First say which property is effected, e.g. `width` or `font-size`,... or `all`
  - Second is the duration, e.g. `1s`
  - Third is the timing function: how speed of change varies over duration. Thease are sometimes called *easing functions*, e.g.:
    - `linear`: same speed from start to end
    - `ease` (default): a slow start, then fast, then slow end
    - Others: `ease-in`, `ease-out`, `ease-in-out` and `cubic`
  - Last, you can have a delay, which says when the change will start, e.g. `0s` (default)
- E.g.

```
img {
    transition: all 1s linear 0s;
}

img:hover {
    transform: rotate(360);
}
```

CSS animations

- CSS even has animations now, using `@keyframes`
- E.g.

```
img {
    position: relative;
    animation: myanimation 5s infinite;
}

@keyframes myanimation {
    0%   {left: 0px;}
    50%  {left: 400px;}
    100% {left: 0px;}
}
```