CS7026 Media Queries Lab 22/02/18

What You’ll Learn...

- We’ll be restyling an entire page layout to work with different screen sizes and devices using Media queries to apply styles selectively based on the visitor’s device properties
- The files you need for today’s lab are available at https://www.scss.tcd.ie/Nina.Bresnihan/teaching/cs7026.html and on Blackboard
- We will look at a layout for a fictional bakery. The layout is fluid so that it adjusts to the width of the browser window, making it work at a variety of screen sizes without generating horizontal scrollbars or causing elements to overlap. However it looks better at some screen sizes than at others. On very wide or very narrow windows, the design is still usable and looks OK, but it’s not as attractive as it is within the 800- to 1200-pixel range.

Recap - What Are Media Queries?

- Media queries let you customize styles based on the characteristics of the user’s device or display. This could be the viewport width, whether it’s in portrait or landscape mode, or whether it shows colour.
- This is different from the media types, such as screen and print, that you can specify for your style sheets in CSS 2.1. With media queries, you specify not only the media type to which you want to apply a set of styles, but also one or more characteristics of the user’s display.
- Here’s an example:

  ```css
  @media screen and (max-width: 600px) {
      body {
          font-size: 88%;
      }
      #content-main {
          float: none;
          width: 100%;
      }
  }
  ```

  This media query starts with the `@media` rule, and then specifies a media type (in this case, `screen`).
- Next there’s the word `and`, followed by the characteristic we want to match against, called the `media feature`. 
This particular media feature, `max-width:600px`, tells the browser that the styles for this media query, which are contained within a set of curly brackets for the media query as a whole, should apply only up to a maximum width of 600 pixels.

If the viewport width exceeds 600 pixels, the browser will ignore the styles inside the media query.

This media query can be dropped right into your main style sheet.

If you want, however, you can apply media queries to separate style sheets on the `link` element or `@import` rule:

```css
@import url(narrow.css) only screen and (max-width:600px);
<link rel="stylesheet" media="only screen and (max-width:600px)" href="narrow.css">
```

The keyword `only` in front of the media type `screen` keeps some older browsers that don’t understand media queries from downloading and applying the style sheets universally. Most non-supporting browsers will not use the sheet anyway, but this is extra insurance. The `only` keyword isn’t needed when you place the `@media` rule directly in your main style sheet.

### Changing the Layout for Large Screens

The design of the bakery page starts looking a bit stretched out at around 1200 pixels wide, so let’s add a media query that will apply only when the window is 1200 or more pixels wide.

Add the following CSS after all the existing styles in the style sheet:

```css
@media screen and (min-width: 1200px) { }
```

This media query has to be at the end of the styles so that it will over-ride the earlier styles.

It tells the browser that we want the styles within this media query to apply to screen media types, but only if the user’s viewport width is 1200 pixels at a minimum.

Of course, right now there are no styles in the media query, just empty brackets waiting to be filled.

Since we have so much extra space in viewports over 1200 pixels wide, how about we fill those brackets with styles to change the layout from two columns to three? To do this, we’ll change the positioning of the navigation div, as well as the widths and margins of the two content divs.

Here are the current styles of these three divs, outside the media query:

```css
#nav-main {
  float: right; margin: 40px 0 0 0;
}
#content-main {
```
overflow: hidden;
float: left;
width: 70%;
margin-bottom: 40px;
}
#content-secondary {
float: right;
width: 25%;
margin-bottom: 40px;
}

Modify these styles for viewports over 1200 pixels wide by adding new rules within the media query you just created:

@media screen and (min-width: 1200px) {
    #nav-main {
        position: fixed; top: 136px;
        width: 13%;
        margin: 0;
    }
    #content-main {
        width: 58%;
        margin-left: 18%;
    }
    #content-secondary {
        width: 20%;
    }
}

This positions the navigation div under the logo, creating a third column.

To make room for it, it was necessary to

- decrease the width of the content-secondary div from 25% to 20%,
- decrease the width of the content-main div from 70% to 58%, and
- add a left margin to content-main.

Let’s also change the widths of the about and credits divs in the footer to match the widths of the columns above them.
Add their IDs onto the #content-main and #content-secondary rules in the media query:

```css
#content-main, #about {
    width: 58%;
    margin-left: 18%;
}

#content-secondary, #credits {
    width: 20%;
}
```

Now all the page elements are better positioned to work well in the width available. Resize your window to see how the layout automatically changes when you get past 1200 pixels wide.

### From Horizontal Nav Bar to Vertical Menu

Although everything is now in the place we want it, some of the page elements could use further cosmetic updates. E.g., the li elements in the nav-main div are floated and have left margins in order to align them all horizontally and space them out from each other, but this keeps them from stacking on top of each other, only one to a line, as we want in a vertical menu.

They also have slightly rounded top corners, which looks good when they’re horizontal, but not when they’re sitting right on top of each other.

We no longer need these styles now that we’re styling the links as a vertical menu, so we’ll over-ride them with new styles within the media query:

```css
#nav-main li {
    float: none;
    margin: 0;
}

#nav-main a {
    -moz-border-radius: 0;
    -webkit-border-radius: 0;
    border-radius: 0; }
```

Now each link is on its own line and takes up the full width of the menu.

Next, we’ll apply some styling to the menu as a whole to make it look more similar to the email newsletter box on the other side of the page.

This has a semi-transparent background, slightly rounded corners, and a soft drop shadow:

```
#nav-main {
```
position: fixed;
top: 136px;
width: 13%;
margin: 0;
-moz-box-shadow: 0 0 8px hsla(0,0%,0%,.1);
-webkit-box-shadow: 0 0 8px hsla(0,0%,0%,.1);
box-shadow: 0 0 8px hsla(0,0%,0%,.1);
-moz-border-radius: 3px;
-webkit-border-radius: 3px;
border-radius: 3px;
background: hsla(0,0%,100%,.3);
text-align: right;
}

Since the menu has its own background colour now, tone down the -gradients on the links within it, so that the two colours layered over each other don't get too opaque:

#nav-main a {
-moz-border-radius: 0;
-webkit-border-radius: 0;
border-radius: 0;
background: -moz-linear-gradient(hsla(0,0%,100%,.3), hsla(0,0%,100%,0) 15px);
background: -webkit-gradient(linear, 0 0, 0 15, from(hsla(0,0%,100%,.3)), to(hsla(0,0%,100%,0)));
}

#nav-main a:hover {
background: -moz-linear-gradient(hsla(0,0%,100%,.6), hsla(0,0%,100%,.2) 15px);
background: -webkit-gradient(linear, 0 0, 0 15, from(hsla(0,0%,100%,.6)), to(hsla(0,0%,100%,.2)));
}

These changes complete the navigation’s transformation from horizontal bar to vertical menu

Multiple Column Text
One of the main complaints people have with layouts that adjust to viewport width is that the length of lines of text can become either too short or too long to be read comfortably or look attractive. In reality, there is no magic line length that is ideal for everyone. A person’s age, reading level, native language, disability, and other factors all influence which line length s/he finds easiest to read. However, it’s true that line lengths on the extreme ends of the range don’t work well for the majority of readers and don’t always look very attractive.

We can use the multiple column properties to control line lengths. These allow you to flow the content of a single HTML element into multiple columns. You create the columns using either the `column-count` or `column-width` properties. In the latter case, the browser will decide how many columns to make based on the available space. You can also use both properties together, though you may get unexpected results...

Let’s break the introductory paragraph into two columns in both the regular layout and the wide layout.

Find the existing `h1 + p` rule in the styles outside of the media query.

Add the `column-count` property, plus the three browser-specific versions, to the rule:

```css
h1 + p {
  -moz-column-count: 2;
  -o-column-count: 2;
  -webkit-column-count: 2;
  column-count: 2;
  color: #7F4627;
  text-shadow: -1px -1px 0 hsla(0,0%,100%,.6);
  font-size: 120%;
}
```

These column boxes are not actual elements in the document tree of the HTML, rather just virtual boxes that the browser creates to flow the content of the paragraph into. The paragraph is now what the W3C calls a *multicol element* - it’s a container for a multiple-column layout.

You can control the space between the columns using the `column-gap` property. Set it to 1.5 ems in the `h1 + p` rule:

```css
h1 + p {
  -moz-column-count: 2;
  -moz-column-gap: 1.5em;
  -o-column-count: 2;
  -o-column-gap: 1.5em;
  -webkit-column-count: 2;
  -webkit-column-gap: 1.5em;
```
If you don’t set a column-gap value, each individual browser decides how much space to add by default. So it’s best to standardise it by explicitly setting the value you want.

Here, we’ve used a value in ems so that the gap will grow larger as the text grows larger, keeping the text more readable.

Changing the Layout for Small Screens

We’ll add a second media query right below the first one you added, targeting viewports that are narrower than 760 pixels wide:

```css
@media screen and (max-width: 760px) { }
```

This tells the browser that we want the styles that we’ll add within this media query to apply to screen media types in viewports up to a maximum width of 760 pixels.

Why 760 pixels? Under 760, the layout starts looking squished, with an increasing possibility of content overflowing its containers.

Let’s start by changing the styles on the nav bar to better fit the available space.

When the window is narrowed, the entire nav bar drops onto a line below the logo, which is fine, but it stays right-aligned, which doesn’t look as good when it doesn’t have the logo to its left. So let’s change the styles on the nav bar to left-align it when it’s on a line below the logo:

```css
@media screen and (max-width: 760px) {  
  #nav-main { clear: left; float: left; }  
  #nav-main li { margin: 0 .5em 0 0; } 
}
```

Next, let’s get rid of the two columns in the introductory paragraph - they’re awkwardly narrow when the window is under 760 pixels.

Change the column count to 1 in a new h1+p rule in the second media query:

```css
h1 + p { -moz-column-count: 1; -o-column-count: 1; -webkit-column-count: 1; column-count: 1; }
```

Now the line lengths are more reasonable in the introductory paragraph, but the three side-by-side columns underneath that paragraph are still extremely narrow.

Right now, each featured product box is a div that’s floated to the left. Removing the floats will make them stack on top of each other instead, filling the whole width of the main content div.
But when they’re stacked on top of each other, the illustration that goes with each feature box doesn’t look as nice positioned at the top of the box - it makes more sense to put the illustration on the left side of the box.

So add this new rule to the media query:

```css
.feature {
    float: none;
    width: auto;
    margin: 0 0 1.6em 0;
    padding: 0 0 0 140px;
    background-position: top left;
}
```

This rule stops the feature boxes from floating and removes their percentage widths. It also removes the top padding from each box and replaces it with left padding, providing room for each illustration - a background image - to sit in on the left side of the box.

The right column is now fairly thin, increasing the chance that long words will overflow it. The headings in the column are in the greatest danger, since their all-caps style makes them take up so much room. We can lessen their chance of overflowing by decreasing their text size and letter spacing:

```css
h3 {
    font-size: 100%;
    letter-spacing: 0;
}
```

This completes the changes we’re going to make for the narrow version of the bakery page.

Save your page and view it in an up-to-date browser.

Resize the window to see the design change at very narrow and very wide widths.

**Changing the Layout for Mobile Devices**

Media queries are a great way to customize the styles on mobile devices quickly and easily.

When adding a mobile media query, what size should you target? Mobile phone screen sizes vary so dramatically that there is an argument to be made that your breakpoints should be based on your design not on devices.

The design of our bakery page starts to break down around 550 pixels. So let’s use 550 as the width to target with our third media query, which will work in 320 by 480 mobile phones as well as mobile phones with larger screens.

```css
@media screen and (max-width: 550px) {
}
```
Make sure you add this beneath the second media query (the one targeting a maximum width of 760 pixels).

That’s because the second media query applies to mobile devices as well - a mobile device with a 480-pixel-wide screen is under the maximum width of 760 pixels. If you put the 550-pixel media query before the 760-pixel media query, the 760 one would override the styles in the 550 one.

This is just how the CSS cascade works - rules that come later override rules of the same specificity that were declared earlier.

If you didn’t want the two media queries to overlap, you could add a minimum width onto the 760-pixel media query, such as:

```css
@media screen and (min-width: 551px) and (max-width: 760px) {
  #content-main, #content-secondary {
    float: none;
}
```

This media query would apply only to windows between 551–760 pixels, not to mobile devices under 551 pixels wide. This might be good or bad, depending on your particular project.

In our case, it would mean repeating a lot of the rules from the 760-pixel media query in the 550-pixel one, since we want a lot of the styles to be the same in both. For instance, we want the intro paragraph to have only one column of text in both the 550-pixel layout and the 760-pixel layout. When these two media queries overlap, we only have to declare the one column in the 760-pixel media query, and then it will also apply to windows under 550 pixels. In our example page, overlapping the media queries lets us reuse several styles and keep our CSS more streamlined.

On other sites, however, you may want very different styles at each width, so it may make more sense to not let your media queries overlap. Keeping them separate may also be less confusing for you, as you don’t have to keep track of the cascade.

Again, there’s no right or wrong answer here - it all depends on what you’re trying to accomplish. In this case, we’re going to leave the 760-pixel media query as it is, and make sure the 550-pixel media query comes below it so that both apply to windows under 551 pixels wide.

### Removing Floats

The primary change we need to make to the mobile design of the site is getting rid of the floats so that the entire page is one column.

Most mobile web pages are a single column - there’s simply not enough room for columns to sit side by side on those little screens.

Add the following rules to the third media query:

```css
@media screen and (max-width: 550px) {
  #content-main, #content-secondary {
    float: none;
}
```
Now the sidebar column displays under the main content column, and the “Credits” block in the footer displays under the “About” block.

The top margin added to the credits div keeps the blocks in the footer spaced out from each other.

Reducing Heights

Another useful change to make to many mobile pages is to reduce the vertical space that elements take up, reducing the amount that users have to scroll down the long single column.

The text in the tagline and introductory paragraph doesn’t need to be quite so large when viewed up close on a mobile device, so you can reduce both font sizes by creating new `h1` and `h1 + p` rules:

```
    h1 { font-size: 225%; }
    h1 + p { font-size: 100%; }
```

Working our way further down the page, you’ll see that the product icons look rather large in the context of such a narrow window, and the text beside them could use more room.

Luckily, the icon set I’ve used for the illustrations came in three sizes: 128 pixels, 64 pixels, and 48 pixels.

We can switch the background images to the 64-pixel size in our mobile media query:

```
    .feature { padding-left: 70px; }
    #feature-candy { background-image: url(images/icon_candy_64-trans.png); }
    #feature-pastry { background-image: url(images/icon_pastry_64-trans.png); }
    #feature-dessert { background-image: url(images/icon_dessert_64-trans.png); }
```
Next, check out the email newsletter subscription block. The text field within it takes up its full width, but there’s now room to display the label text and button on the same line as the text field, at least on larger mobile screens.

Add these rules to the media query:

```
#form-newsletter * { display: inline; }
#form-newsletter input[type=text] { width: auto; }
```

These changes tighten up the newsletter block’s appearance.

In portrait-oriented mobile screens, the subscribe button will drop down to a second line, but even then the form still makes better use of the space overall.

Finally, we can make a small change in the footer to slightly reduce its height.

Float the `dt` elements within the credits div:

```
#credits dt {
    clear: left;
    float: left;
    margin: -.05em .2em 0 0;
}
```

### Preventing Overlapping Header Elements

In small mobile screens, the possibility of page elements overlapping each other is of course increased. You can see this problem in the header of our example page. With the viewport at 550 pixels wide, the search form fits fine beside the logo, but at around 400 pixels they start to overlap. If the user has a larger text size, the overlap will happen even sooner.

To reduce the chance of overlap, reduce the width of the text field in the search form by adding this rule to the third media query:

```
#form-search input[type=text]{width:100px; }
```

Next, add a fourth media query below the 550-pixel one. This media query will target windows less than 401 pixels wide:

```
@media screen and (max-width: 400px) { }
```

Add a rule within this media query to make the label in the search form display as a block-level element so it will sit on a line above the text field:

```
@media screen and (max-width: 400px) {
    #form-search label { display: block; }
}
```

Now the search form takes up less width at both 550 pixels wide and 400 pixels wide, and it’s not likely to overlap the logo even in 320-pixel wide mobile phone screens.