CS 2033

Multimedia and Communications

Lab 09: Modern Website Features

- Website Development -

REMEMBER TO BRING YOUR MEMORY STICK TO EVERY LAB!

MODERN FEATURES

Now that you are an expert in HTML, CSS, and JS, you are ready to modernize your website! This lab will give you a taste of three modern features and how to use them effectively.

Parallax is a popular feature to add aesthetic appeal to websites. Even though it doesn't add any functionality, it provides a beautiful visual effect that attracts users and makes your site look modern and relevant.

Scrollfire is another aesthetic feature that users tend to appreciate. Again there is no *need* for it as it doesn't provide any functionality or do anything useful, but it looks pleasing and modern.

Responsiveness, on the other hand, is less about aesthetics and more about functionality. Now that phones and tablets take up a huge part of the online market, we can no longer create a website that only works on our own computer monitor. Sites must work across all (or most) platforms and screen sizes ranging from phones to tablets to laptops to monitors and even TVs.

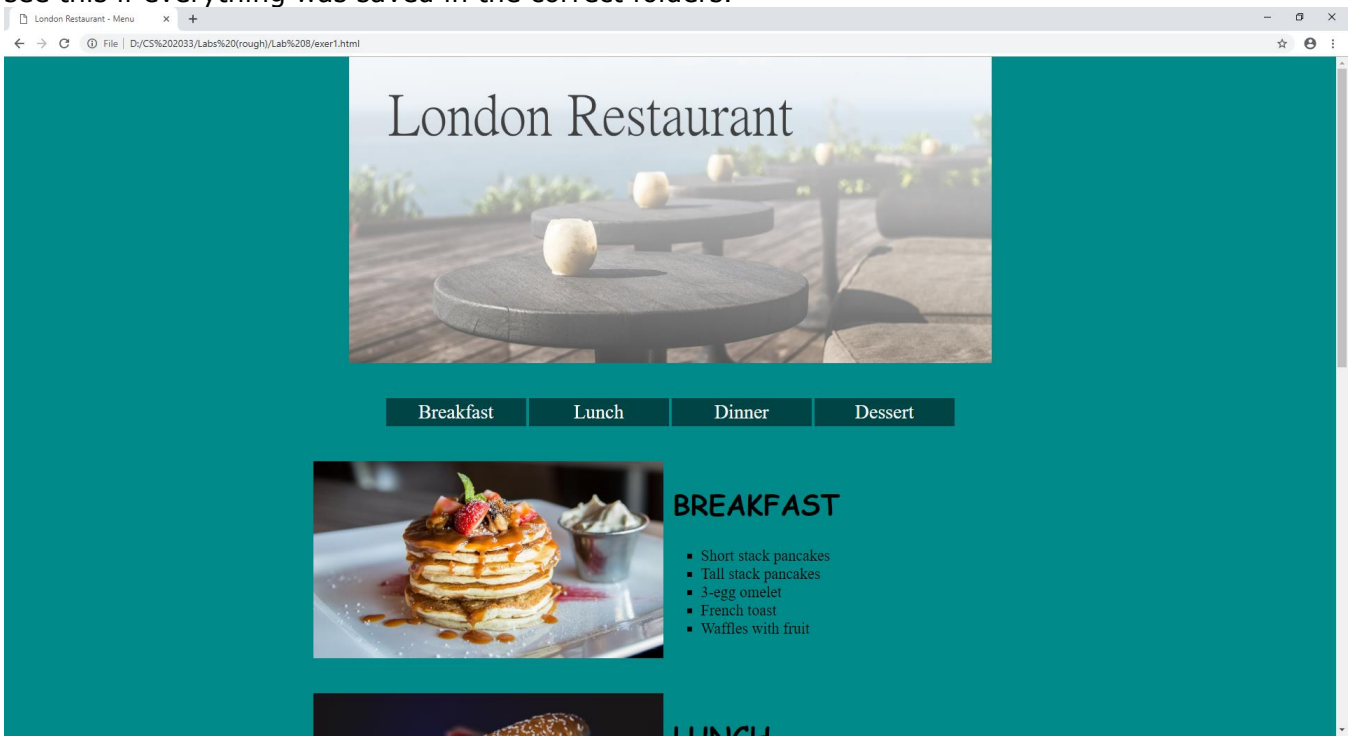
EXERCISE 1: PARALLAX

What you will learn in this exercise:

- Adding simple parallax with background-attachment:fixed
- Using JS to enhance the parallax
- Applying algebra (linear equation) to calculate parallax shift

1. Navigate to your USB folder (F:) and into the ***cs2033*** subfolder (should have been created in a previous lab).
2. Create a folder called ***lab09*** inside ***cs2033***. Within ***lab09*** create two subfolders, one called ***img*** and one called ***css***.
3. Open http://www.csd.uwo.ca/~bsarlo/cs2033b/labs/Lab09/img/, download all the images, and move them into ***lab09*** > ***img.***
4. Open http://www.csd.uwo.ca/~bsarlo/cs2033b/labs/Lab09/css/, download ***main-styles.css***, and move it into ***lab09*** > ***css***.
5. Open http://www.csd.uwo.ca/~bsarlo/cs2033b/labs/Lab09/, download ***exer1.html***, and move it into ***lab09***.
6. Open exer1.html in Brackets and immediately save it in the same folder as ***exer1_complete.html***. All work in this exercise should be done in exer1_complete.html unless otherwise specified.

7. Open exer1_complete.html in Chrome as well to see how it looks from the start. You should see this if everything was saved in the correct folders.



8. This site looks decent but not amazing. We want to add a little spice to it with parallax. There is an image in the img folder called salmon.jpg which we will use as the background image for the parallax.
9. Rather than inserting the picture as an img element, we will create a div that has this image embedded as the background. Let's place it between the breakfast and lunch sections.
10. Click between these two sections and add a few lines. In there, create a div with the class "parallax-img" and the ID "para" (we need an ID because we'll be using JS to access it shortly). The code around that area should look like this (the provided breakfast and lunch sections are partially shown for context):

```
                <li>French toast</li>
                <li>Waffles with fruit</li>
            </ul>
        </div>
    </section>


    <div class="parallax-img" id="para"></div>


    <a name="lunch"></a>
    <section>
        <div class="box"><img src="img/burger.jpg" class="sect-img" /></div>
        <div class="box menu-group">
            <h1>LUNCH</h1>
            <ul>
                <li>Hamburger</li>
```

11. Refresh the browser. Nothing changes because we haven't yet done anything to this div.
12. In the head section of this page, create a style tag so that we can begin customizing our parallax div:
    <style type="text/css">
    </style>

13. Within that style tag, create a rule-set to apply to the class "parallax-img" and within it, set the width to 100% and the height to 300px. We also want to add space around the bottom so it's away from the food sections, so add another style to change margin-bottom to 30px.
14. Refresh the browser. You should now see empty space between the breakfast and lunch sections.
15. Now we have to set the background image in the div. Go back into the ruleset for .parallax-img. Set the background-image property to url("img/salmon.jpg"). The ruleset should currently look like this:

```
<style type="text/css">

    .parallax-img {
        width:100%;
        height:300px;
        margin-bottom: 30px;
        background-image: url("img/salmon.jpg");
    }

</style>
```

16. Save the file and refresh the browser and you should the top of the image appearing. Notice it's only a small section because our div is only the width of the wrapper and the height of 300px. Also note that it doesn't show the whole image and stretch it to these dimensions the way an img element would be shown with the same width and height given.
17. Add another style in the ruleset that sets the background-size to 'cover' (quotations not needed in CSS). This indicates that we want the full size of the image shown which just makes a subtle difference if you save and refresh the browser.
18. We do not yet have any parallax. If you scroll up and down the site, this image just sits there like any other element.
19. The next step is to set the background-attachment to 'fixed' (again no quotations) within the ruleset. This means that the image should remain fixed in place when you scroll rather than shifting with everything else. The ruleset should now look like this:

```
.parallax-img {
    width:100%;
    height:300px;
    margin-bottom: 30px;
    background-image: url('img/salmon.jpg');
    background-size:cover;
    background-attachment:fixed;
}
```

20. Save the file and refresh the browser. Scroll up and down several times and watch the salmon picture as you do. Notice that you only see 300px of the image's height at any given time and yet you can see the entire height of the picture from scrolling up and down because it is stationary "behind" the webpage. This is the basic form of parallax!

Enhancing the parallax

21. This current type of parallax is a great start and some websites have this type alone so it's not bad. However, we're feeling adventurous so let's enhance it with some JS code!
22. In the head section of the same HTML file, add a section for internal JS:
    <script type="text/javascript">
    </script>

23. Within this JS area, create a function called scroll() that takes no input parameters:

```
<script type="text/javascript">

    function scroll() {

    }

</script>
```

24. Inside this function, create a variable called t and assign it the value of window.scrollY (scrollX is the amount scrolled horizontally and scrollY is the amount scrolled vertically – we only want the vertical amount).
25. Then create another variable, this one called para. Use document.getElementById() to get the element with ID "para" and store it in this new variable.
26. We need to calculate what the background position will be so that it will shift as a function of the scroll amount. It should shift linearly, so use the equation of a line: y = mx+b. Before we code it, it's important to understand the equation and how to use it for parallax.
    y = background position in the div (this will shift as we scroll to create the parallax effect)
    x = how far we've scrolled from the top in pixels (this is our variable called t)
    m = how quickly to shift the image position
    b = the background position when x=0 (no scrolling)
27. Now we can begin applying this to calculate the y value (the new background position). Create the following two variables: m and b. Give m a value of 0.5 to begin, but we'll tweak this later. Give b a value of 0 since this is the default background position before scrolling.
28. Create another variable, called newY, and give it the value calculated from the linear equation using t (as the x), m, and b (note you need the asterisk * for multiplication):
    var newY = m*t + b
29. The last step for this function is to assign the new background position to the para element. There are x and y positions for horizontal and vertical shifts respectively. In our case, we only need to shift it vertically so we'll change the backgroundPositionY style from JS. Add the following line to complete the scroll() function:
    para.style.backgroundPositionY = newY + "px";
30. Your scroll() function should now look like this:

```
function scroll() {

    var t = window.scrollY;
    var para = document.getElementById("para");
    var m = 0.5;
    var b = 0;

    newY = m*t + b;
    para.style.backgroundPositionY = newY + "px";
}
```

31. Now the function is complete but we aren't calling it yet so this is the last step. We want the background position to change for the parallax effect as we scroll so we'll need to add an onscroll event handler to the body tag. Update the body tag as shown here:
    <body onscroll="scroll();">
32. Now save the file and refresh the browser. Scroll up and down and watch the salmon picture shift within its div. You might notice something strange occurring. The image shifts quickly and then it cycles around again. The reason for this is that background-images are tiled by default, and by changing the position of the background, we see the bottom of one tile and the top of the next, thus creating this ugly cycling effect. Even though this is technically parallax, it doesn't currently have the aesthetic appeal that parallax normally has. We can fix it easily though by changing the m parameter.

33. Go back up to the scroll function and change the value of m to -0.25. Save the file and refresh the browser. Scroll up and down again and watch the smooth motion of the salmon picture as you scroll. That's much better!

Note: the m value should be negative to make the image shift up as you scroll down; if m is positive, then it shifts down when you scroll down, which looks incorrect.

34. Play around with this m value and see how it changes the effect of the parallax shift. Once you have tried a few different values, revert it to a value of -0.25 and save it before going on to exercise 2.

EXERCISE 2: SCROLLFIRE

What you will learn in this exercise:

- Examining scroll position to highlight current section link
- Using scroll position to grow (scale) an image
- Using scroll position to slide in text
- Seeing what else is doable through online examples

Pre-exercise

Before beginning this exercise, familiarize yourself with some of the forms of scrollfire events that modern websites use. Open http://scrollmagic.io/examples/basic/simple_tweening.html in Chrome and scroll down and up to see what occurs. Then use the dropdown menus at the top to switch to the next "Basic" example and see how that works. Continue doing this to view all the Basic and Advanced examples on the site (don't worry about looking at the Expert ones unless you want to!).

1. Navigate to your USB folder (F:) and into the ***cs2033*** > ***lab09*** subfolder.
2. Open http://www.csd.uwo.ca/~bsarlo/cs2033b/labs/Lab09/, download ***exer2.html***, and save it into ***lab09***.
3. Open exer2.html in Brackets and immediately save it in the same folder as ***exer2_complete.html***. All work in this exercise will be done in exer2_complete.html.
4. Open exer2_complete.html in Chrome as well to see what it looks like before coding. Notice that the navigation bar sticks to the top as you scroll past it. This is called sticky and it is something that can be done with JS or just with CSS. In this case, it is done with CSS.

Highlighting the current section link

5. Click in the head section of exer2_complete.html and create a JavaScript tag:
   <script type="text/javascript">
   </script>
6. Within the JS area, create a function called scroll() that takes 0 input parameters. All the JS in this exercise will go in this function.

7. Create a variable called t and give it the window.scrollY value just like in exercise 1.
8. Create another variable and call it nav. Use document.getElementById with the id "nav" to store that element in the new nav variable. The JS should currently look like this:

```
<script type="text/javascript">
function scroll() {

    var t = window.scrollY;
    var nav = document.getElementById("nav");


}
</script>
```

9. Before we can do much else, we need to create a lot of variables that hold the HTML elements: the 4 links, the 4 sections, and then the vertical positions of those 4 sections. Start by creating a variable called l1 (that is a lower case L followed by a number 1) and it will hold the l1 element obtained from document.getElementById("l1").
10. Repeat the previous step 3 more times but change l1 in both the variable name and the getElementById parameter to l2, l3, and l4 respectively. It should look like this:

```
var l1 = document.getElementById("l1");
var l2 = document.getElementById("l2");
var l3 = document.getElementById("l3");
var l4 = document.getElementById("l4");
```

11. Next, create another 4 variables called s1, s2, s3, and s4 which will each be obtained from document.getElementById with the IDs that match the variable names, i.e. s1 is the element with ID "s1". These four lines simply look like this:

```
var s1 = document.getElementById("s1");
var s2 = document.getElementById("s2");
var s3 = document.getElementById("s3");
var s4 = document.getElementById("s4");
```

12. The s1-s4 elements represent the breakfast, lunch, dinner, and desserts sections within the website. We want to add an underline on the corresponding links at the top as we scroll past each of these sections in the page. In order to do this, we need to see if our scroll position (t) is greater than the position of each of the sections in the page. We can use the property called offsetTop from each section to retrieve its position in pixels from the top of the page: var t1 = s1.offsetTop;
13. Repeat the above line 3 more times for t2, t3, and t4 (getting the offsetTop of s2, s3, and s4 respectively). These variables look like this:

```
var t1 = s1.offsetTop;
var t2 = s2.offsetTop;
var t3 = s3.offsetTop;
var t4 = s4.offsetTop;
```

14. Using simple conditionals we can now check which section we are in based on the scroll position (t) compared to each of the section positions (t1-t4). To simplify the code, we will start by looking at t4, followed by t3, then t2, and t1 last. This is important because if we started by checking if t > t1 (meaning we scrolled into the section 1 area), it would be true for any scrolling **below** that area too which would throw off the whole conditional (remember only the first conditional that is satisfied will execute even if multiple are true).

15. So we'll add these in reverse order like so:

```
if (t > t4) {

} else if (t > t3) {

} else if (t > t2) {

} else if (t > t1) {

}
```

16. Just to re-explain this, as we scroll down the page, one of these conditionals may be triggered. When we're near the bottom of the page, all 4 conditionals would technically be true because our t wil be greater than t1, t2, t3, and t4. We only want the t4 section to be triggered so that the corresponding link, l4, will be underlined. Similarly when we're in the 3rd section (dinner), the t value would be greater than t1, t2, and t3 (but not t4 because that section would be lower down) – but we only want the 3rd link to be underlined. For this reason, we have the conditionals in the reverse order here so that only the correct section is being triggered.
17. Within each of these 4 conditionals, we want to underline the corresponding link. We can simply access the CSS of the link and change borderBottomColor to gold. Within the t4 portion, add the following code:
l4.style.borderBottomColor = "gold";
18. Follow the same format for t3, t2, and t1 but change the link variable in each to match (l3, l2, and l1 respectively). This entire conditional section should now look like this:

```
if (t > t4) {
    l4.style.borderBottomColor = "gold";
} else if (t > t3) {
    l3.style.borderBottomColor = "gold";
} else if (t > t2) {
    l2.style.borderBottomColor = "gold";
} else if (t > t1) {
    l1.style.borderBottomColor = "gold";
}
```

19. Now, there's a minor problem with this code. When we scroll down past each section, the corresponding link will be given a gold underline. However, these underlines are never removed so all 4 would have a golden underline at the same time when you scroll to the bottom. We need to reset them all to black underlines **before** the above conditional. Click just above the conditional and add the following line to reset link l1 to black:
l1.style.borderBottomColor = "black";
20. Repeat the above line 3 more times for l2, l3, and l4 respectively (the order of these doesn't matter as long as they are all done before the conditional).

```
l1.style.borderBottomColor = "black";
l2.style.borderBottomColor = "black";
l3.style.borderBottomColor = "black";
l4.style.borderBottomColor = "black";
```

21. The last step for this portion is to add the event listener so that this function is called as we scroll down. Like the parallax exercise, add an onscroll listener to the body tag like so:
<body **onscroll="scroll();"**>
22. Save the file and reload Chrome. As you scroll down and back up, pay attention to the underlines in the navigation bar at the top.

Transitions on scrolling

23. You may have noticed a bowl of soup in the webpage after the 4 sections. We are going to make this image grow when we scroll near it. There is also a message hidden on the side that will slide in to the middle of the page when we scroll to that area.
24. Go back into the scroll() function we've been working on. After the conditional in there for the underlines, create a new variable called soup that is the element with ID "soup".
25. Right after that, add a single if-statement that checks if we have scroll into/past the area where that soup image is (around 2400px from the top):

```
var soup = document.getElementById("soup");
if (t > 2400) {

}
```

26. Now we want to access soup's CSS within this JS to make it larger when we scroll into the 2400 range. There's a CSS style called transform with a sub-property called scale which is used to resize images larger or smaler. We'll use this to double the image's size. Within the t > 2400 conditional, add the following line to grow the image:
soup.style.transform = "scale(2, 2)";
27. Below that conditional, create another variable called msg to hold the element "msg".
28. Create another if-statement like the previous one which checks if t > 3000. In this conditional, we want to set the marginLeft property of msg to 0px (it is 2000px by default from the CSS, putting it off screen). These last few steps should look like this:

```
var soup = document.getElementById("soup");
if (t > 2400) {
    soup.style.transform = "scale(2, 2)";
}

var msg = document.getElementById("msg");
if (t > 3000) {
    msg.style.marginLeft = "0px";
}
```

29. Save this file and refresh the browser. Make sure you are scrolled to the very top and refresh again. Now slowly scroll down and watch the soup bowl grow when you approach it. A little further down, you should see the message slide in from the right side into the middle of the screen.

EXERCISE 3: RESPONSIVE

What you will learn in this exercise:

- Using JS to load different stylesheets
- Adding media queries in the <link> tags
- Adding media queries in the stylesheets
- Tips on layout structures between different devices/screens
- Using Chrome's developer tools to help with responsive design

1. Navigate to your USB folder (F:) and into the ***cs2033*** > ***lab09*** subfolder.
2. Open http://www.csd.uwo.ca/~bsarlo/cs2033b/labs/Lab09/, download ***exer3.html***, and save it into ***lab09***.
3. Open exer3.html in Brackets and immediately save it in the same folder as ***exer3_complete.html***. Most work will be done in exer3_complete.html and some will be done in additional CSS files as specified.
4. Before you begin adding any code, open exer3_complete.html in Chrome and resize the browser big and small width-wise (don't worry about resizing its height). As you resize, make a mental note of what happens to the site, i.e. what gets cut off at various sizes.

Creating responsive styles

5. Create a new file in Brackets and immediately save it into ***cs2033*** > ***lab09*** > ***css*** with the name mob-styles.css (styles for small mobile screens).
6. Repeat step 10 but name this one tab-styles.css (styles for tablet/medium screens).
7. Open both mob-styles.css and tab-styles.css in Brackets.
8. Go into mob-styles.css so we can add mobile styles. The first responsive style change is giving the wrapper class a %-based width. Start by adding the following CSS rule-set:
```
.wrapper {
    width:100%;
}
```
9. Next we need the box class to have display:block so that they are single-column instead of inline-block which creates a 2-column layout on a computer screen. Add the following CSS:
```
.box {
    display:block;
}
```
10. The banner at the top of the site also needs to change to a %-based width:
```
.banner img {
    width:100%;
}
```
11. Each of the 4 menu sections (breakfast, lunch, dinner, and desserts) has a big image of one of the items in the corresponding menu. These images are being reduced to single column from the above box class rule-set, however it also needs to be resized to fill the window:
```
.sect-img {
    width:100%
}
```
12. Another major style change for responsiveness is the navigation menu layout. Rather than a horizontal navbar, mobile screens should show the links stacked on top of one another in a vertical structure. We'll add a rule-set next to change 4 styles in the nav list items:
```
ul.nav li {
    font-size:28px;
    width:100%;
    height:40px;
    line-height:40px;
}
```
13. Lastly, each of the menu sections also has a list of meals in a bulleted list. These use the menu-group class, which has a width of 450px for computer screens, but cannot be that large on mobile screens. To be safe, we'll use width:auto which means it can be as big as it needs to be to include the text content within it but it won't go beyond the edge of the screen. We also need to remove the left margin here. This rule-set looks like this:
```
.menu-group {
    width:auto;
```

```
    margin-left:0;
}
```

14. Save mob-styles.css. Now select all 6 rule-sets you just added into this stylesheet, copy it all (Edit > Copy or Ctrl+C), and then go into tab-styles.css and paste it there (Edit > Paste or Ctrl+V). We're doing this because most of these rule-sets can now be used for both mobile and tablets alike. Only one minor change needs to occur for the tablet stylesheet.
15. Let's make this one small change now. In tab-styles.css, click into the wrapper class at the top and change the width to 90%. Then add a new line within the same rule-set and give it a top margin of 5% so that the banner isn't tight to the top. The wrapper class in tab-styles.css should now look like this:

```
.wrapper {
    width:90%;
    margin-top:5%;
}
```

16. That's it for creating the responsive stylesheets! They won't be applied yet though as we haven't loaded these files into the HTML page. The two screenshots here show you what the completed responsive stylesheets should now look like:

mob-styles.css

```
.wrapper {
    width:100%;
}

.box {
    display:block;
}

.banner img {
    width:100%;
}

.sect-img {
    width:100%;
}

ul.nav li {
    font-size:28px;
    width:100%;
    height:40px;
    line-height:40px;
}

.menu-group {
    width:auto;
    margin-left:0;
}
```

tab-styles.css

```
.wrapper {
    width:90%;
    margin-top:5%;
}

.box {
    display:block;
}

.banner img {
    width:100%;
}

.sect-img {
    width:100%;
}

ul.nav li {
    font-size:28px;
    width:100%;
    height:40px;
    line-height:40px;
}

.menu-group {
    width:auto;
    margin-left:0;
}
```

JavaScript-loaded stylesheets

17. Click in the head section of exer3_complete.html below the <title> tag. Create a script tag for holding JavaScript:
    <script type="text/javascript">
    </script>
18. In that JS area, create a variable called w and give it the value window.innerWidth;
19. Then add a 2-part conditional that first checks if w <= 640, and if not, then it checks if w <= 1020 (because this comes second, it will only trigger if w is between 641 and 1020).
20. One global stylesheet was loaded directly in HTML at the top of the head section and it contains styles that apply to the whole page. For the responsiveness, we will now load a different secondary stylesheet for each of the screen size ranges. For <= 640px it will be mob-styles.css and for 641px to 1020px it will be tab-styles.css. These are done just like a regular HTML <link> tag to load a CSS file but they are put within the document.write() function so that the JavaScript executes these lines.
21. The JavaScript portion should look something like this (although you don't need to add the comments that I included):

```
<script type="text/javascript">
    var w = window.innerWidth;

    if (w <= 640) {
        document.write('<link rel="stylesheet" href="css/mob-styles.css">');
    } else if (w <= 1020) {
        document.write('<link rel="stylesheet" href="css/tab-styles.css">');
    }

</script>
```

22. Save the file and refresh Chrome. Resize the browser as small as it will go width-wise. Does the website change at all? No. While it's at this small size, refresh again. Now you should see the mobile styles take effect. Scroll down to observe all the contents/elements and see how they have changed to a single column.
23. Now resize the window to somewhere in the medium range (roughly between 1/3 and 1/2 the width of the entire screen). Again you won't see anything change right away so refresh again at this size and now you should see that the top banner is pushed down a bit and there are margins along the sides as well. Most of the other elements follow the same vertical structure that you observed in the smallest window size in the previous step.
24. Finally, resize your window to be large again – either full screen or close to it. Once again the styles won't be changed upon resizing so you will have to refresh to let them take effect. When you do, you should see what you saw before adding any responsiveness to the site: 2 columns for the menu sections (picture on the left side and menu header and list on the right side) and a horizontal navigation bar.
25. Now the website works at several different sizes so it is responsive! However, this JavaScript approach is not very efficient as you may have noticed. You had to refresh the browser for it to take effect because the JavaScript code runs as the page loads so it has to retrieve the new width each time and load a stylesheet accordingly. Theoretically we could use an onresize event handler to load the stylesheets as we resize, but this is very inefficient as it would be loading those files repeatedly and eating up memory in the computer. There are better options directly in CSS without using JS. Before continuing, comment out the JavaScript code here. The easiest way to do this is click in the line just above the opening <script> tag and add the characters <!-- and then click in the line just after the closing </script> tag and add -->. You should see everything between there is

now grayed out to indicate that it's commented out and won't be executed anymore.

```
<!--
<script type="text/javascript">
    var w = window.innerWidth;

    if (w <= 640) {
        document.write('<link rel="stylesheet" href="css/mob-styles.css">');
    } else if (w <= 1020) {
        document.write('<link rel="stylesheet" href="css/tab-styles.css">');
    }

</script>
-->
```

HTML media queries

26. In exer3_complete.html, click at the end of the <link> line that is loading main-styles.css and hit Enter a few times to create more space.
27. Just after that <link> line, add two new <link> lines that are similar to the first one but these will load css/mob-styles.css and css/tab-styles.css respectively (you may copy and paste that first line to create these next two lines more quickly, but make sure you change the filenames).
28. The original <link> loads main-styles.css and this file will apply to the site at any screen size so it's fine as is. The new two links are only meant to load if the screen (or browser window) is small. We can add media queries as attributes within these tags to indicate the size ranges that they should apply to.
29. In the <link> line that's loading mob-styles.css, add the following attribute: media="(max-width: 640px)"
30. In the <link> line that's loading tab-styles.css, add the following attribute: media="(min-width: 641px) and (max-width: 1020px)"
31. The three <link> lines in the HTML should now look like this:

```
<link rel="stylesheet" href="css/main-styles.css">
<link rel="stylesheet" media="(max-width: 640px)" href="css/mob-styles.css">
<link rel="stylesheet" media="(min-width: 641px) and (max-width: 1020px)" href="css/tab-styles.css">
```

32. Save the file and refresh your browser. Now resize the browser to as small as it goes, then to medium width, and then to large. You should notice that the responsive styles from mob-styles.css and tab-styles.css ar taking effect immedately as you resize so you don't have to refresh! This is a much more clean and efficient approach to loading the different stylesheets for responsiveness.

CSS media queries

33. The last approach to responsiveness is similar to the previous one, but it is done directly in CSS rather than in the HTML. Media queries are still used here but rather than loading a different stylesheet for each size range, here we can load one larger stylesheet that contains both sets of media queries.

34. Before beginning this portion, comment out the two link lines added in the previous steps again using the <!—and --> characters and ensure they are grayed out.

```
<!--
<link rel="stylesheet" media="(max-width: 640px)" href="css/mob-styles.css">
<link rel="stylesheet" media="(min-width: 641px) and (max-width: 1020px)" href="css/tab-styles.css">
-->
```

35. Create a new file in Brackets and immediately save it in ***cs2033*** > ***lab09*** > ***css*** with the name mq-styles.css.
36. We'll start by creating a media query for small mobile devices in mq-styles.css. These are similar to the media queries that went into the HTML <link> tags in the previous portion in that the min-width and max-width can be specified. Here the media queries are CSS structures with curly brackets { and } that will contain rule-sets that only take effect if the media query conditional is met.
37. Create the first media query (for mobile devices) with the following CSS:
    @media (max-width: 600px) {

    }
38. Below that, create the second media query (for tablets/medium screens):
    @media (min-width: 641px) and (max-width: 1020px) {

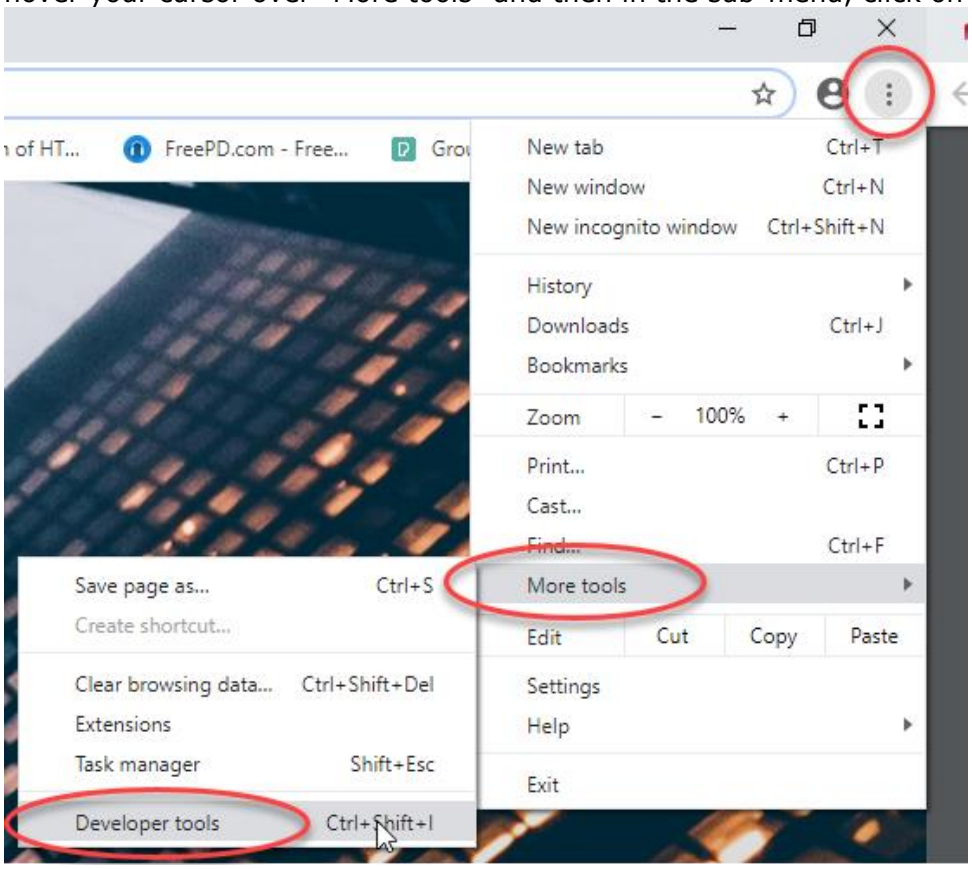
    }
39. Click into mob-styles.css, select everything in there, and copy it. Now click back into mq-styles.css and into the first media query structure you created above and paste the code that was copied from mob-styles.css.
40. Click into tab-styles.css, select everything in there, and copy it. Now click back into mq-styles.css and into the second media query structure you created above and paste the code that was copied from tab-styles.css. Save mq-styles.css.
41. The last step is to load this file from the HTML page so go back into exer3_complete.html. Select the <link> line that loads css/main-styles.css and paste it immediately below that line so there are two identical lines. Now change the second line to load css/mq-styles.css.
42.
43. Save mq-styles.css and refresh the browser. Resize the window to very small, then to medium width, and then to large. At each size range, notice that the styles are changing just like they did in the previous portion (HTML media queries) so you don't have to refresh for the site to respond to resizing.

Note: these CSS media queries could even be put into main-styles.css after all the global rule-sets and it would work the same as this. We are using mq-styles.css to hold the media queries for simplicity so the file doesn't get too large and messy, but it would still work the same in one file.

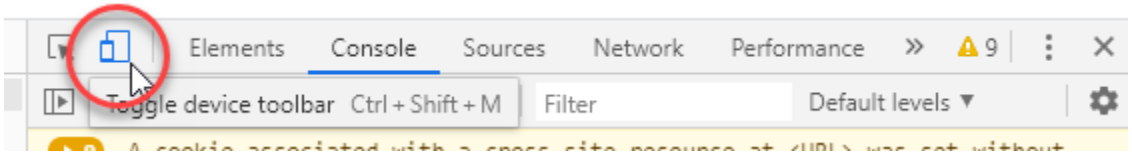
Google Chrome's responsive mode

44. In this final portion of the lab you will use a tool built in to Google Chrome that helps with testing your site for responsiveness. It's essentially an emulator which means it acts like other devices so you can see what the website looks like on the different devices very easily. They have a bunch of common phones and tablets pre-loaded in the emulator and they also provide options to customize the screen to any size you want.

45. In Chrome, click the icon of 3 vertically stacked dots in the top-right corner. In that menu, hover your cursor over "More tools" and then in the sub-menu, click on "Developer tools".

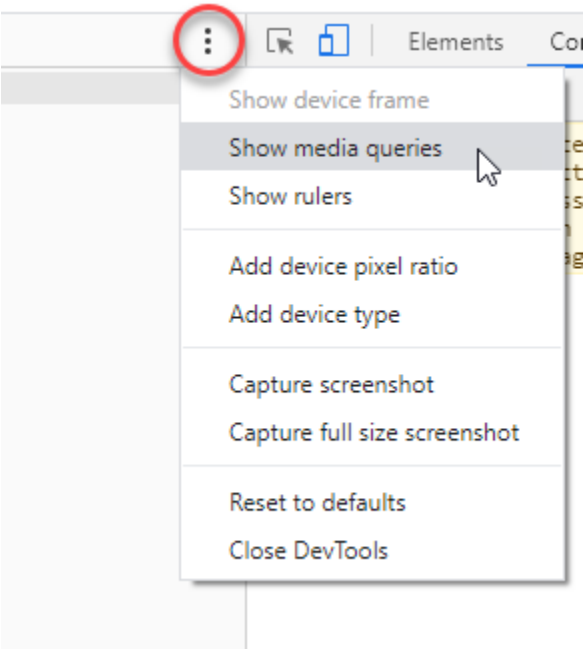


46. You should now see a panel open on the right side of the browser. There are numerous tools in there that help with website development. We just want to use the Device Toolbar so click on the little icon with the phone and tablet near the top of this panel. If you hover over it, you should see the label "Toggle device toolbar". When you click that to toggle it on, the left part of this toolbar changes, reducing the size of the screen's viewport.



47. At the top of this area, there are several form inputs that you can change. Firstly there is a select dropdown menu where you can choose from various phone or tablet models. Try changing the device in that list and watch as the emulator changes size for each one. You should see that the responsive styles you added in mq-styles.css are being applied so the selected device's size is triggering the respective media query styles.
48. There are several other parameters and settings you can change in here, like toggling the device frame (only works for some of the devices), toggling the media queries as coloured rectangles above the device, and toggling rulers to see the dimensions more clearly. Some of these settings are hidden in the menu under the three dots icon (NOT the one you clicked

in step 44, but it looks the same! This one is in the upper-right corner of the left panel).



49. Play around with these different device models and the settings for a few minutes until you are comfortable with how it all works.

This concludes this lab session. Call your TA over to check your work and receive your mark for this lab.

REMEMBER TO REMOVE YOUR MEMORY STICK FROM YOUR MACHINE AND PUT IT IN YOUR BACKPACK! (don't forget it)! ☺