Learning Tree® International

Training You Can Trust®

Exercise Manual for Course 522

Building Responsive
Web Applications With CSS3

522/MA/K.1/709/J.2

by Scott Hopkinson

Technical Editor:
Richard Innis

Exercise Manual Contents

Legend for Course Icons ii
Hands-On Exercise 1.1: Prepping HTML for Style 1
Hands-On Exercise 2.1: Styling Container Geometry 9
Hands-On Exercise 3.1: Styling Content, Part 1.... 23
Hands-On Exercise 3.2: Styling Content, Part 2.... 31
Hands-On Exercise 4.1: Positioning Containers, Part 1 41
Hands-On Exercise 4.2: Positioning Containers, Part 2 51
Hands-On Exercise 5.1: Web Typography 63
Hands-On Exercise 6.1: Building Accessible Forms 77
Hands-On Exercise 6.2: Building Accessible Tables.... 89
Hands-On Exercise 7.1: Managing a Sass Project.... 99

Legend for Course Icons

Standard icons are used in the hands-on exercises to illustrate various phases of each exercise.

👣	**Major step**	⚠	**Warning**
1. ☐	**Action**	💡	**Hint**
✓	**Checkpoint**	🛑	**Stop**
?	**Question**	🏆	**Congratulations**
ⓘ	**Information**	🎖	**Bonus**

Overview

During this course, we will convert a starting HTML template into an accessible and responsive HTML5 website powered by CSS, with a small sprinkling of JavaScript syntactic sugar added for that finishing touch.

When you need to add code to existing HTML or CSS, the code is displayed bookended by the existing code to help you find the correct location for your work. The code to add is always marked in **`bold text`**.

Our first exercise involves adding required elements for accessibility and creating an external stylesheet template.

Part 1

1. ☐ Open and briefly examine our HTML starting template in the browser of your choice (aka Chrome!):

    `C:\inetpub\wwwroot\pizzatree\Ex\Ex1.1\index.html`

2. ☐ Open the same `index.html` file in the IDE of your choice.

    This is a rather typical HTML file, devoid of HTML5 elements that assist screen readers by providing context for the content. Too many divs, zero style.

3. ☐ Within the opening `<head>` element on line 2, add a language attribute and set the value to English:

```
<!DOCTYPE html>
<head lang="en">
<link rel="icon" type="image/png" href="img/pizza_tree_ico.png" />
```

4. ☐ Directly after the same `<head>` element, add a `<meta>` element that tells the browser to render the page in Unicode, which supports many languages (you might want to use a different language than English!):

```
<head>
<meta charset="UTF-8">
<link rel="icon" type="image/png" href="img/pizza_tree_ico.png" />
```

5. ☐ Find the first `<div>` element at around line 9. This provides no context to screen readers or search engines. HTML5 provides alternative contextual HTML elements for this very purpose.

   Convert the `<div>` to a `<nav>` element, and don't forget to convert the closing `</div>` near line 17.

6. ☐ Examine around line 18. We have another useless `<div>` element, this time acting as a container for some important content—our featured content.

   Directly under the `<div>` element around line 18 and *before* the `<h2>`, add a new `<header>` element that acts as a container for our featured content. Close this new `<header>` element after the image and before the closing `<div>` element near line 21. Our featured content should now look like this:

```
<div>
    <header>
        <h2>The Pizza of your Dreams!</h2>
        <img src="img\pizza_tree_logo_lg.png" />
    </header>
</div>
```

7. ☐ Imagine trying to provide context (as a screen reader) to the `<div>` elements starting around line 24! We can do much better with `<article>` and `<section>` elements.

   Convert the container `<div>` element around line 24 into an `<article>` element (don't forget to close the match near line 37!) and convert each containing `<div>` element that has child Pizza Special `<h3><p>` pair into a `<section>` element:

```
<article>
    <section>
        <h3>Washington Special</h3>
        <p>2x LG 2-Topping &dollar;19.99
    </section>
    <section>
        <h3>Ottawa Special</h3>
        <p>3x MD 3-Topping &dollar;19.99
    </section>
    <section>
        <h3>London Special</h3>
        <p>1x LG 6-Topping &pound;19.99
    </section>
</article>
```

8. ☐ Find yet another unhelpful `<div>` element near line 38 used as a container for our testimonials and company history. Convert this to an `<article>` element.

9. ☐ The last `<div>` element contains our copyright information and is clearly a footer, so convert it to a `<footer>` element.

Part 2

10. ☐ Save your accessible HTML5 starter template and switch back to your browser. Reload the page and you should notice: nothing has changed! Rather, the content is the same, but now the content has context for screen readers and search engines.

    It’s now time to create our style template. Create a new folder called `/css` in the Ex 1.1 folder. In this new folder, create a new file called `default.css`

11. ☐ Add a link to load this new `default.css` file to your `index.html` document. This should be added within the `<head>` element, after any `<meta>` elements (don’t forget the rel attribute!). It does not matter if you place your `<link>` before or after the `<title>` element, but it will help with readability if you place multiple <link> elements sequentially:

```
<link rel="icon" type="image/png" href="img/pizza_tree_ico.png" />
<link rel="stylesheet" href="css/default.css" />
<title>Pizza Tree</title>
```

12. ☐ Save your `index.html` document and open the new `default.css` document you created in Step 10.

    We are going to style the following elements within this stylesheet:

```
<nav>          <h2>
<a>            <section>
<div>          <p>
<header>       <footer>
```

13. ☐ Add the appropriate element selectors to your stylesheet. We will add CSS properties and values in the next step. Your current CSS stylesheet should look like the following:

```
nav {

}
a {

}
div {

}
header {

}
h2 {

}
section {

}
p {

}
footer {

}
```

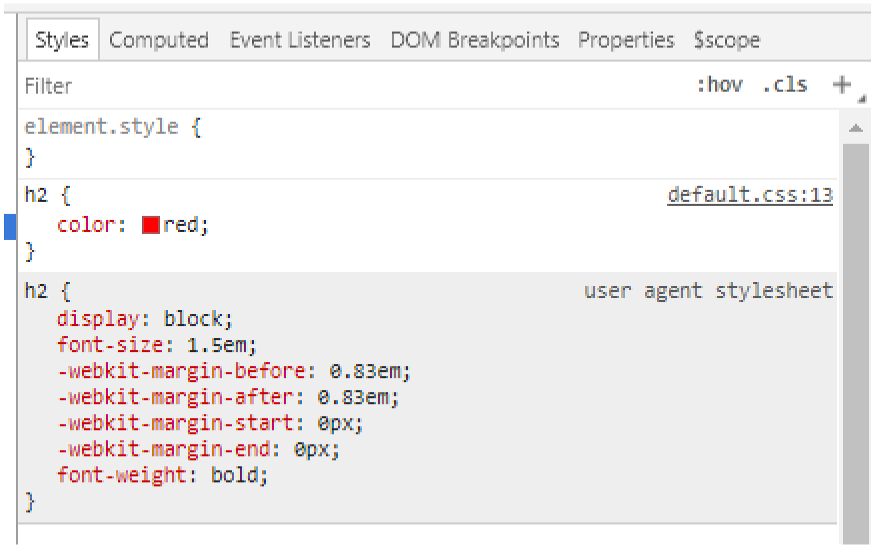
14. ☐ Before you save and test (nothing would change in appearance despite all your hard work), add a style rule for the `<h2>` element to change the text color to red:

```
h2 {
    color:red;
}
```

15. ☐ Save both your `index.html` and `default.css` files, and switch back to your browser to preview your work. The only visible change is the color of the `<h2>` element text, but our template is now accessible and follows best practices for loading external CSS documents.

16. ☐ Still in your browser, press `<F12>` to load the browser debugger panel, and inspect the `<h2>` element by pressing `<Ctrl><Shift><C>` (or the element inspect button on the upper left of the debugger panel).

As you move your cursor around the viewport, you'll see the debugger highlight each container in a pale blue box. Click the text **The Pizza of your Dreams**, and notice the debugger now shows you the CSS rule in the bottom right of the debugger tool. In addition, default user-agent styles are also displayed:



```
element.style {
}
h2 {                                          default.css:13
   color: 🟥red;
}
h2 {                                    user agent stylesheet
   display: block;
   font-size: 1.5em;
   -webkit-margin-before: 0.83em;
   -webkit-margin-after: 0.83em;
   -webkit-margin-start: 0px;
   -webkit-margin-end: 0px;
   font-weight: bold;
}
```

17. ☐ Carefully place your cursor next to the `color:red` property set within the `h2` selector from `default.css`, and uncheck the checkbox. The `h2` text in your viewport should revert to the color from the user-agent stylesheet. Click the checkbox again to change the color back to red.

18. ☐ Click anywhere within the white space of the `h2` selector from `default.css`, and add the following property set:

```
font-size:4em;
```

You should see the viewport change again. This is an easy way to play the what-if game when testing potential changes to CSS files without having to save/reload.

Congratulations! You have finished the exercise.

This is the end of the exercise.

Overview

In this exercise, we will use CSS to identify the geometry of our HTML containers and customize this geometry to suit our design requirements. This will help us later as we position page elements with accuracy and consistency. Screenshots taken for this exercise are in 600px viewport width.

Part 1

1. ☐ Open the starting point for Hands-On Exercise 2.1 in your browser first, followed by your IDE. This is where we left off after Ex 1.1:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex2.1\index.html
```

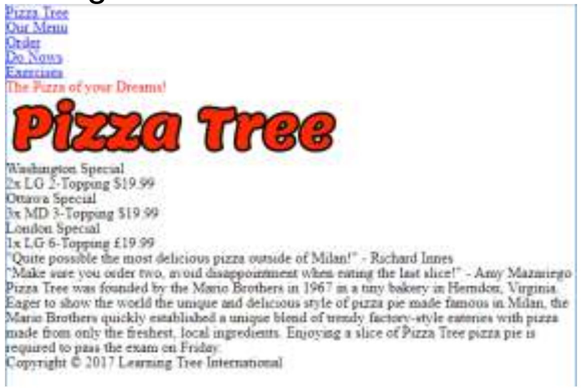
2. ☐ Open the CSS file referenced in the `<link>` element within your IDE. You may wish to close your previous exercise files.

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex2.1\css\default.css
```

3. ☐ Apply the Meyer reset to `index.html` by adding a link element before the reference to the default stylesheet. This will eliminate any inconsistencies between browser renders. This reset file has been provided in the `/css` folder of this exercise.

```
<link rel="stylesheet" href="css/reset.css" />
```

4. ☐ Save `index.html` and preview in a browser, noting the dramatic change to the basic content:



Pizza Tree
Our Menu
Order
Do Now
Exercises
The Pizza of your Dreams!

Washington Special
2x LG 2-Topping $19.99
Ottawa Special
3x MD 3-Topping $19.99
London Special
1x LG 6-Topping £19.99
"Quite possible the most delicious pizza outside of Milan!" - Richard Innes
"Make sure you order two, avoid disappointment when eating the last slice!" - Amy Mazmirgo
Pizza Tree was founded by the Mario Brothers in 1967 in a tiny bakery in Herndon, Virginia. Eager to show the world the unique and delicious style of pizza pie made famous in Milan, the Mario Brothers quickly established a unique blend of trendy factory-style eateries with pizza made from only the freshest, local ingredients. Enjoying a slice of Pizza Tree pizza pie is required to pass the exam on Friday.
Copyright © 2017 Learning Tree International

5. ☐ The reset is a great starting point for browser consistency, but there will still be human error issues when we assign heights and widths to containers that have box model properties such as border and padding.

Switch to `default.css` in your IDE and eliminate this box-model sizing issue by applying a generic selector rule as the first selector at the top of the file:

```
* {
  box-sizing: border-box
}
```

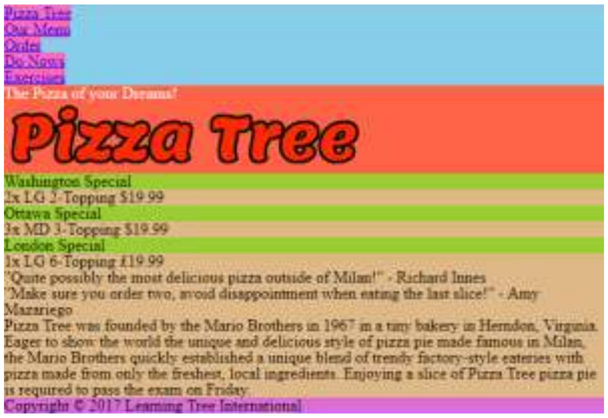
6. ☐ Apply the following background color values to the associated selectors:

```
nav       background:skyblue;
a         background:hotpink;
div       background:tomato;
section   background:yellowgreen;
p         background:burlywood;
footer    background:orchid;
```

7. ☐ Continuing within `default.css`, change the color of the `<header>` element to white; recall we assigned the value of red as a test at the end of the last exercise, but the header won't be visible on a tomato background:

```
h2 {
    color:white;
}
```

8. ☐ Save `default.css` and preview `index.html` in a browser:



Pizza Tree
Our Menu
Order
Do Nows
Exercises
The Pizza of your Dreams!
Pizza Tree
Washington Special
2x LG 2-Topping $19.99
Ottawa Special
3x MD 3-Topping $19.99
London Special
1x LG 6-Topping £19.99
"Quite possibly the most delicious pizza outside of Milan!" - Richard Innes
"Make sure you order two, avoid disappointment when eating the last slice!" - Amy Mazariego
Pizza Tree was founded by the Mario Brothers in 1967 in a tiny bakery in Herndon, Virginia. Eager to show the world the unique and delicious style of pizza pie made famous in Milan, the Mario Brothers quickly established a unique blend of trendy factory-style eateries with pizza made from only the freshest, local ingredients. Enjoying a slice of Pizza Tree pizza pie is required to pass the exam on Friday.
Copyright © 2017 Learning Tree International

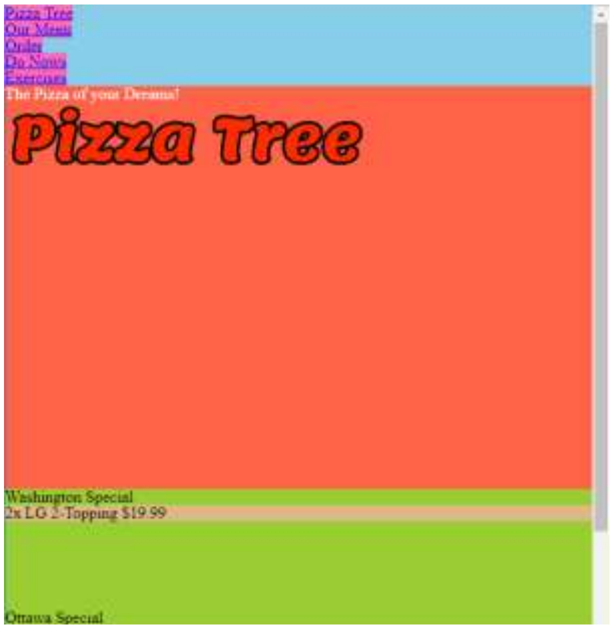
9. ☐ Assign heights to the `div` and `section` selectors to approximate their dimension as seen in the finished case study:

```
div {
    height:400px;
    background:tomato
}
section {
    height:120px;
    background:yellowgreen
}
```

10. ☐ Assign a width of 100% to the anchor selector to force the anchor to occupy the same width as the parent list items:

```
a {
     **width:100%;**
     background:hotpink
}
```

11. ☐ Save `default.css` and preview `index.html` in a browser:



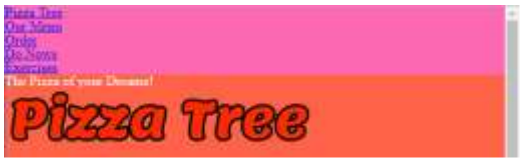
Why haven't the anchors responded to our dimension assignment the way the div and the section have?

Answer: *Anchors are inline elements and cannot be assigned height or width.*

12. ☐ Return to `default.css` and change the display property of the anchor selector from inline (which it is by default) to `inline-block`. `inline-block` forces inline elements to accept dimension properties but won't inherit the top and bottom margins of full block-level elements:

```
a {
    display:inline-block;
    width:100%;
```

13. ☐ Save `default.css` and preview `index.html` in a browser:



14. ☐ We're going to start styling individual regions of our website. It's a best practice to create separate stylesheets for separate areas of functionality, much as you would do for JavaScript files.

    Within the `/css` folder of `Ex2.1`, create the following new CSS documents:

```
nav.css
feature.css
```

15. ☐ Starting around line 9 of `default.css`, cut (remove for pasting elsewhere) the `div`, `header`, and `h2` selector contents. Switch to the new `feature.css` document you created in Step 14, and paste the selectors you just copied here.

16. ☐ Switch to `index.html` in your IDE, and add a stylesheet link to your new `feature.css` stylesheet directly after the link to `default.css` around line 9:

```
<link rel="stylesheet" href="css/default.css" />
<link rel="stylesheet" href="css/feature.css" />
<title>Pizza Tree</title>
```

17. ☐ Save all three open files in your IDE: `index.html`, `default.css`, and `feature.css`. You won't see any changes if you preview `index.html` in your browser, but we are on our way to a modularized project that will be easy to manage and test.

18. ☐ Switch to `feature.css` in your IDE and center the content of the header selector:

```
header {
    text-align:center;
}
```

19. ☐ Switch back to `index.html`. We are going to assign a background image of a delicious pizza to our menu and features, but not to the rest of our content. This requires adding a container for this background image that contains our menu and features.

    Wrap a `<div>` element starting around the `<nav>` element around line 14 and ending after the pre-existing `<div>` element around line 28.

20. ☐ Give this new `<div>` element an `id` of `"container"` and indent the now child `<nav>` and sibling `<div>` elements appropriately:

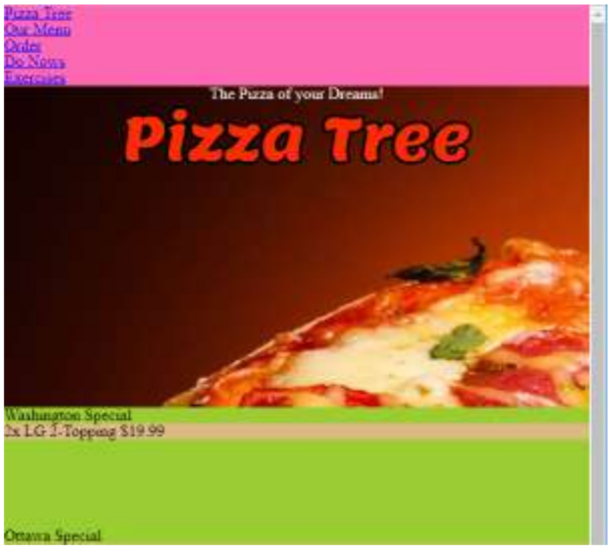
```
<div id="container">
    <nav>
        <ul>
        <li><a href="#">Pizza Tree</a></li>
        <li><a href="#">Our Menu</a></li>
        <li><a href="#">Order</a></li>
        <li><a href="#">Do Nows</a></li>
        <li><a href="#">Exercises</a></li>
        </ul>
    </nav>
    <div>
        <header>
            <h2>The Pizza of your Dreams!</h2>
            <img src="img\pizza_tree_logo_lg.png" />
        </header>
    </div>
</div>
```

21. ☐ Switch to `default.css` and add a selector that targets the `<div id="container">` element you added in the last step. Set the background to the image `pizzabg.jpg` located in the `/img` folder of Ex2.1:

```
div#container {
    background-image:url(../img/pizzabg.jpg);
}
```

22. ☐ Remove the tomato background color from the `div` selector within `feature.css`.

23. ☐ Save `feature.css`, `default.css`, and `index.html`, and preview `index.html` within a browser:

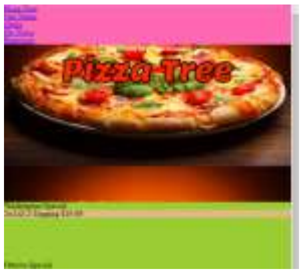


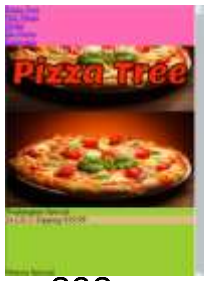
24. ☐ The pizza is far too large for a 600px-wide viewport. We also need a transparent gradient background on our `nav` element so the gradient glow from the pizza shows through the `nav` element, creating the look from our completed case study.

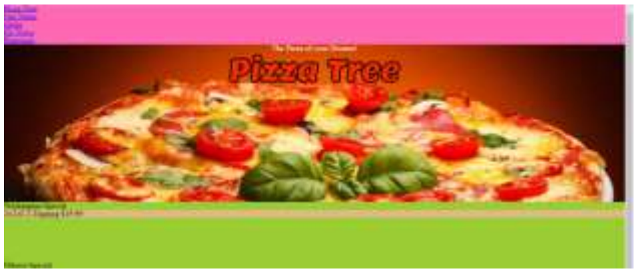
    Switch back to `default.css` and update the `div#container` selector with a background-size rule of 100%:

```
div#container {
    background-image:url(../img/pizzabg.jpg);
    background-size:100%;
}
```

25. ☐ Save `default.css` and preview your changes in `index.html` in a browser:


600px


392px


1280px

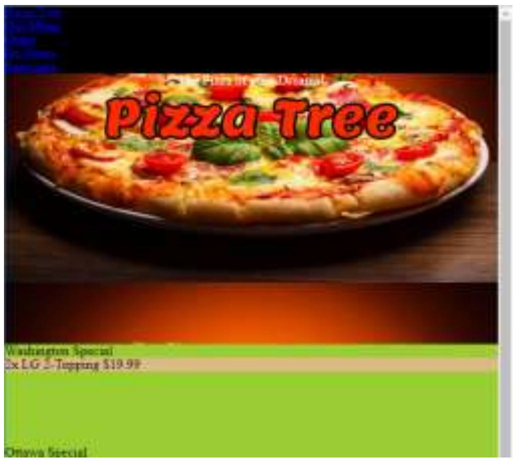
Setting the background image to 100% width does not work for all viewports! Neither will setting it to auto, or any other value we have covered so far. We need a responsive value, which we will cover in a later chapter.

26. ☐ Return to `default.css` in your IDE and cut the nav and anchor selectors. Cut and Paste the `nav` and `anchor` element from `default.css` into `nav.css`:

```
nav {
    background:skyblue;
}
a {
    display:inline-block;
    width:100%;
    background:hotpink
}
```

27. ☐ Within your new `nav.css` document, change the background color of the anchor selector to transparent (or simply remove the old background property:value). This will allow the nav element background to show through the anchors.

28. ☐ Save `nav.css` and `default.css`, and preview `index.html` in a browser:



Washington Special
2x LG 2-Topping $19.99

Ottawa Special

Does your result resemble the above screenshot?

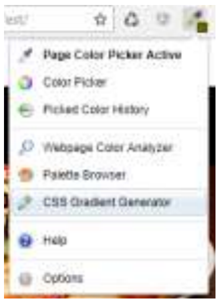
***Answer:** No! We fell victim to a common issue when modularizing our CSS code—we forgot to add the link referencing our new external `nav.css` file.*

29. ☐ Return to `index.html` in your IDE and add the missing external stylesheet reference:

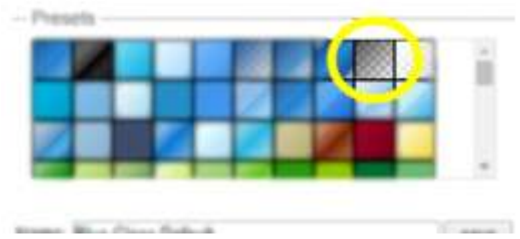
```
<link rel="stylesheet" href="css/reset.css" />
<link rel="stylesheet" href="css/default.css" />
<link rel="stylesheet" href="css/nav.css" />
<link rel="stylesheet" href="css/feature.css" />
```

30. ☐ Save `index.html` and preview in a browser. Your nav element should now have the missing `skyblue` background, but the pizza gradient glow isn't showing through.

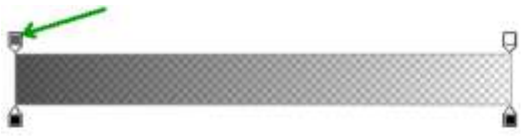
31. ☐ From Chrome, press the Colorzilla menu and select the **CSS Gradient Generator**.



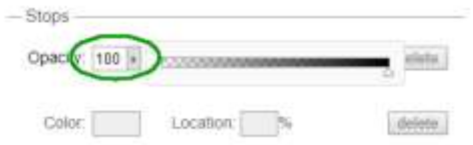
32. ☐ From the Presets swatch library, select the second gradient from the top right as in the screenshot below:



33. ☐ With the transparent black preset selected, press the top left opacity stop in the color stop mixer below the presets:



34. ☐ Just below in the Stops options, change the opacity to 100% using the pop-out slider:

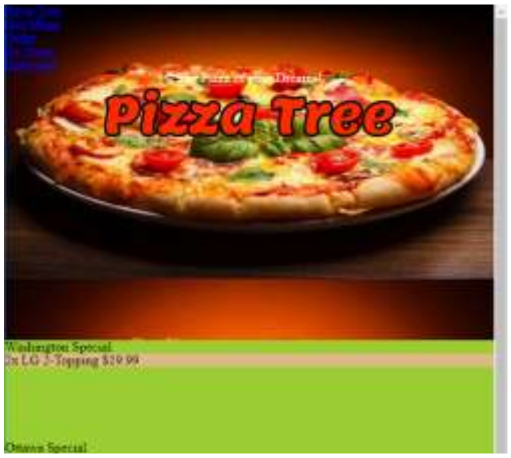


35. ☐ Copy the code for the gradient you just selected from the CSS pane. Switch back to your IDE and `nav.css`. Paste the background from the CSS Gradient Generator on top of the background rule within the nav selector (replace the old background property with the pasted rule).

Note—your generated BG will look slightly different but the effect should look roughly the same—it doesn't have to be a perfect match!

```
nav {
    /* Permalink - use to edit and share this gradient:
http://colorzilla.com/gradient-editor/#000000+0,000000+100&1+0,0+100 */
    background: -moz-linear-gradient(top, rgba(0,0,0,1) 0%, rgba(0,0,0,0) 100%);
/* FF3.6-15 */
    background: -webkit-linear-gradient(top, rgba(0,0,0,1) 0%,rgba(0,0,0,0)
100%); /* Chrome10-25,Safari5.1-6 */
    background: linear-gradient(to bottom, rgba(0,0,0,1) 0%,rgba(0,0,0,0) 100%);
/* W3C, IE10+, FF16+, Chrome26+, Opera12+, Safari7+ */
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#000000',
endColorstr='#00000000',GradientType=0 ); /* IE6-9 */
    width:100%;
}
```

36. ☐ Save `nav.css` and preview `index.html` in your browser:



Congratulations! You have modularized your CSS into external stylesheets that will make managing and testing your code much easier, and harmonized the box model between all browsers with a style reset. We are getting closer to achieving the look in the finished case study, but much work remains.

STOP

This is the end of the exercise.

Overview

In this exercise, we will leverage more advanced selectors to avoid additional DOM pollution when making selections for style. Screenshots taken for this exercise are in 910px viewport width.

Part 1: Preparing a 3-column layout

1. ☐ Open the starting point for Ex 3.1 in your browser first, followed by your IDE. You may wish to close previous exercise files. This is where we left off after Ex 2.1:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.1\index.html
```

2. ☐ Open the CSS files referenced in the `<link>` element within your IDE:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.1\css\default.css
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.1\css\nav.css
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.1\css\feature.css
```

3. ☐ Within the `/css` folder of `Ex3.1`, create the following new CSS document:

```
specials.css
```

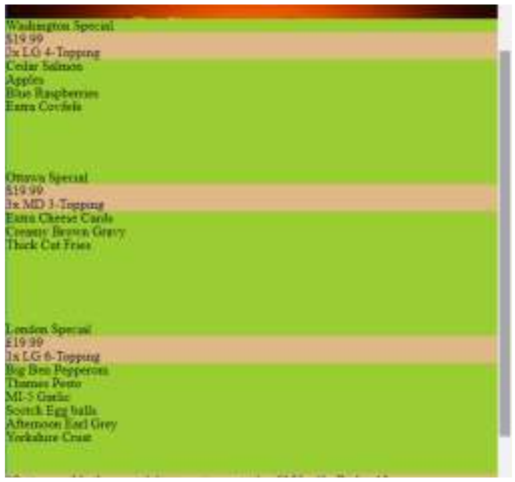
4. ☐ Switch to `index.html` in your IDE and add the link for this new stylesheet:

```
<link rel="stylesheet" href="css/nav.css" />
<link rel="stylesheet" href="css/feature.css" />
<link rel="stylesheet" href="css/specials.css" />
```

5. ☐ Switch to `default.css` in your IDE and cut the style for the section selector. Switch to the new `specials.css` document you created in Step 3, and paste the section selector you just copied in this empty file.

6. ☐ Change the height in the section selector from 120px to 180px. We added some more toppings to our specials last night, but they didn't fit. Adding to the px height is not an ideal solution—we'll fix this with flexbox later.

7. ☐ Save all open files and test `index.html` in your browser:



Washington Special
$19.99
2x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
Extra Cordials

Ottawa Special
$19.99
3x MD 3-Topping
Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

London Special
£19.99
3x LG 6-Topping
Big Ben Pepperoni
Thames Pesto
MI-5 Garlic
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

8. ☐ Our finished case study displays the specials in a single row. We offer three specials, so switch back to your IDE and change the width of each section selector in `specials.css` to 33%. Don't worry about the missing 1% for now.

9. ☐ Save `specials.css` and test `index.html` in your browser.

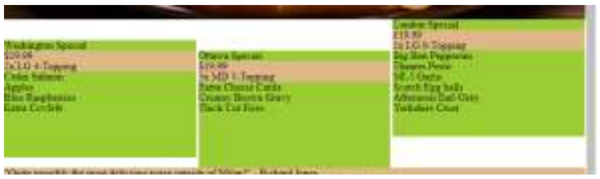
❓ *Why aren't the sections in a single row now that they are only 33% of viewport width?*

Answer: *The Meyer reset converted sections to block-level elements, which have a line break before and after.*

10. ☐ Switch back to your IDE and `specials.css`. Change the display property of the section selector to `inline-block` (not just inline or our width value won't be respected!):

```
section {
    height:180px;
    background:yellowgreen;
    width:33%;
    display:inline-block;
}
```

11. ☐ Save `specials.css` and test `index.html` in your browser:



Washington Special
$20.98
2x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
[illegible]

Ottawa Special
$19.99
3x MD 1-Topping
Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

London Special
£19.99
1x LG 6-Topping
Big Ben Pepperonis
Thames Pesto
[illegible]
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

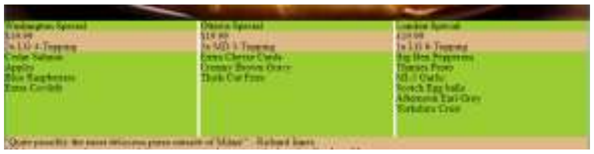
[illegible]

12. ☐ CSS has a wonderful property: `vertical-align`. It almost never works as intended, which has led to numerous additions to CSS such as flexbox and grid. In this case however, `vertical-align` to the rescue!

    Switch back to your IDE and `specials.css` and add the `vertical-align` property with a value of `top`:

```
section {
    height:180px;
    background:yellowgreen;
    width:33%;
    display:inline-block;
    vertical-align:top;
}
```

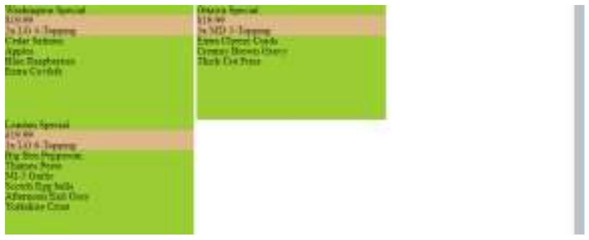
13. ☐ Save `specials.css` and test `index.html` in your browser:



14. ☐ Remember that missing 1%? Let's assign it to the middle special, which happens to the second section element on the page. Return to your IDE and `specials.css`, and after the `section` selector, create a selector for the second section that changes the width from 33% to 34%:

```
section:nth-of-type(2) {
   width:34%;
}
```

15. ☐ Save `specials.css` and test `index.html` in your browser:



This was not the desired effect! By setting the combined width of child elements to 100%, there isn't enough room to display within the parent article element, despite the Meyer Reset having eliminated all margins, borders, and padding. Using more advanced positioning techniques will solve this problem. You were asked to attempt this fix as you are likely to run into this in your professional careers and now you know what *not* to attempt.

For the curious, relative units of measure such as % accept decimal values, and both 33.1% and 33.2% will get you closer to a solution, but 33.3% will wrap the third section.

16. ☐ Switch back to your IDE and `specials.css` and remove the width property from the second `section` selector. Keep the selector: we will use this later.

17. ☐ Fix the text touching the edge of the specials containers by assigning 8px of padding around all edges to all specials:

```
section {
    ...
    vertical-align:top;
    padding:8px;
}
```

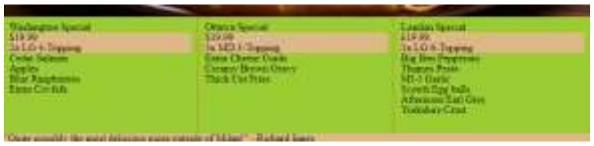
18. ☐ Set the background of the containing article to the same green assigned to sections (add an `article` selector), and the illusion of 100% width is complete:

```
article {
    background:yellowgreen;
}
```

19. ☐ Add the red border from the completed case study to the right edge of the first two sections, and not the last section. There are a few different ways to accomplish this with selectors, here is but one solution:

```
section:not(:last-of-type) {
    border-right:1px dotted red;
}
```

20. ☐ Save `specials.css` and test `index.html` in your browser:



Washington Special
$19.99
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

Ottawa Special
$19.99
[illegible]
Extra Cheese Curds
[illegible]
Thick Cut Fries

London Special
$19.99
[illegible]
[illegible]
[illegible]
[illegible]
Scotch Egg balls
[illegible]
[illegible]

[illegible] — Richard [illegible]

Part 2. Creating a tooltip via callout

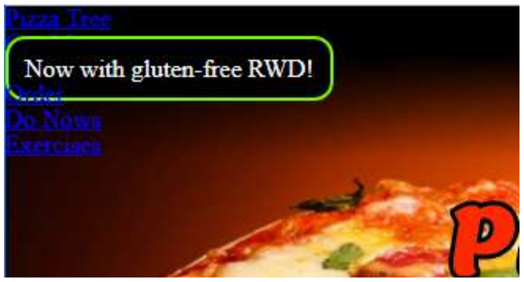
21. ☐ Switch over to `index.html` in your IDE. Near line 20, find the Menu list item, and after the `<a>` element, add a `<span>` element for a callout as in the following:

```
<li><a href="#">Our Menu</a><span class="callout">Now
with gluten-free RWD!</span></li>
```

22. ☐ Switch back to `default.css` in your IDE and start styling the callout as per the case study. We require a green border, black background, white text, and additional styling to create the callout:

```
.callout {
    width:200px;
    color:white;
    background: black;
    border: 2px solid lawngreen;
    border-radius:12px;
    padding:10px;
}
```

23. ☐ Save all open files and preview `index.html` in your browser:



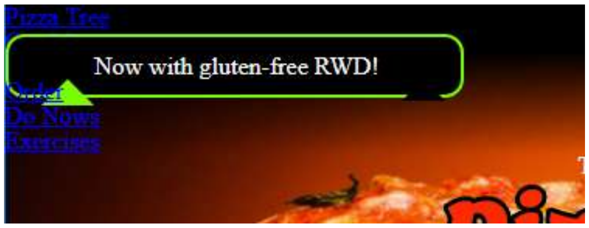
24. ☐ Switch back to your IDE and `nav.css` and add a shared selector for `:before` and `:after` on the `.callout` class. This selector will create a pseudo element border style for the callout tail by creating the illusion of a triangle:

```
.callout:before, .callout:after  {
    border: solid transparent;
    content: "";
}
```

25. ☐ Continue by individually styling the `:before` and `:after` pseudo selectors, making the green border (triangle) slightly thicker than the black border (triangle) for the appearance of a green border around the entire callout pseudo element:

```
.callout:before {
    border-width: 17px;
    border-bottom-color: lawngreen;
    margin-left: 11px;
}
.callout:after {
    border-width: 14px;
    border-bottom-color: black;
    margin-left: 14px;
}
```

26. ☐ Save `nav.css` and preview `index.html` in your browser:



✓ *We're almost there! If only we could reliably position the triangles created with `:before` and `:after` pseudo elements… that's coming up in the next chapter.*

Congratulations! You have completed the exercise.

This is the end of the exercise.

Overview

In this exercise, we will continue leveraging pseudo selectors and selector chains, and deploy the `calc()` and `filter()` functions. Screenshots taken for this exercise are in 910px viewport width.

Part 1: Preparing the login view

1. ☐ Open the starting point for Ex 3.2 in both your browser first, followed by your IDE. You may wish to close previous exercise files. We are working on a new file, `login.html`:

   ```
   C:\inetpub\wwwroot\pizzatree\Ex\Ex3.2\login.html
   ```

   Working within `login.html`, add a link to a new CSS stylesheet:

   ```
   <link rel="stylesheet" href="css/nav.css" />
   <link rel="stylesheet" href="css/feature.css" />
   <link rel="stylesheet" href="css/login.css" />
   ```

2. ☐ Create this new CSS stylesheet within the `/css` folder of the Ex 3.2 project:

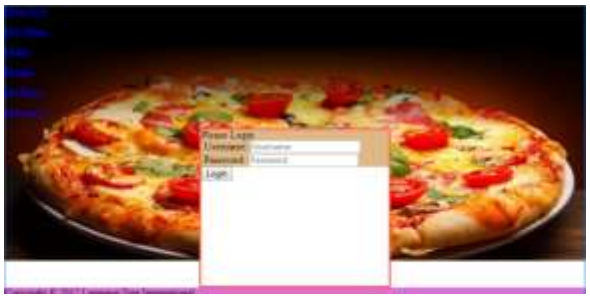
```
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.2\css\login.css
```

3. ☐ Open your new `login.css` stylesheet in your IDE. Before we can calculate the horizontal and vertical centers for aligning our login prompt, we need to setup some initial selectors for our form elements: a `div` with an `id` of `login` to contain our form, a form, some paragraphs containing form element pairs, and the label/input pairs themselves.

   Start by styling the `<div id="login">` with the appropriate selector within `login.css`. Set the height to 300px, the width to 250px and the margins to auto. Finish by adding a tomato, 4px, solid border, and set the background color to white:

```
div#login {
    margin:auto;
    width:300px;
    height:250px;
    border:4px solid tomato
}
```

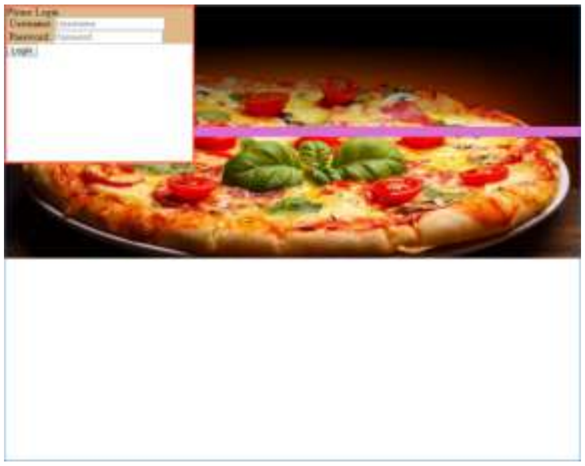
4. ☐ Save `login.css` and preview `login.html` in your browser:



5. ☐ There appears to be artificial vertical centering due to the height of navigation elements. For now, return to `login.css` within your IDE and within the `#login` selector set the position property value of the login prompt to absolute and the top property to 0. This will help illustrate the need for proper horizontal and vertical alignment functions:

```
position:absolute;
top:0;
```

6. ☐ Save `login.css` and preview `login.html` in your browser:

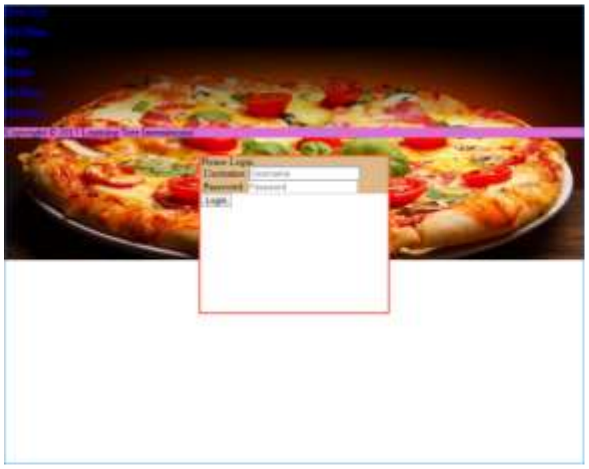


Note the footer has moved to just beneath the navigation menu and is displayed underneath the login prompt. This is because our position rule has removed the login prompt from the natural flow of the linear layout, and DOM elements have filled in the space previously occupied by the login prompt.

7. ☐ Using the `calc()` function, set the login prompt to center both horizontally and vertically by adjusting the top and left margin. Use one-half the height and width we declared as part of the `calc()` function (recall the space requirement on both side of the subtraction operator):

```
margin-top:calc(50vh - 125px);
margin-left:calc(50vw - 150px);
```

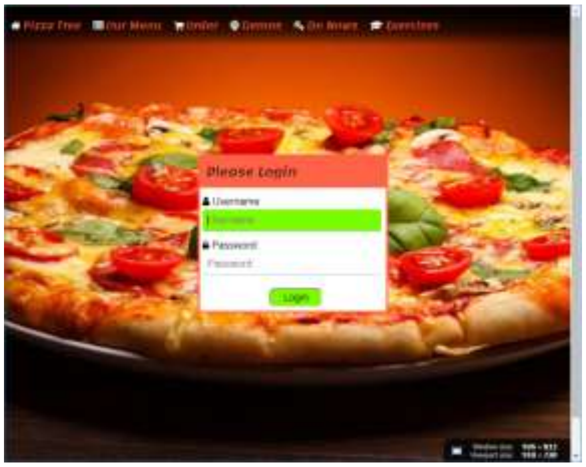
8. ☐ Save `login.css` and preview `login.html` in your browser:



9. ☐ Switch back to `login.css` in your IDE and style the `<p>` element containing the text `"Please Login"` to match the Case Study:
   - Add the appropriate selector and set the background to `tomato`
   - Add 10px of padding and after centering the text
   - Set the font-size to 25% larger than default (1.25em or 125%):

```
div#login p:first-of-type {
    background:tomato;
    padding:10px;
    text-align:center;
    font-size:1.25em;
}
```

10. ☐ Our (completed) Case Study login prompt has the following appearance:

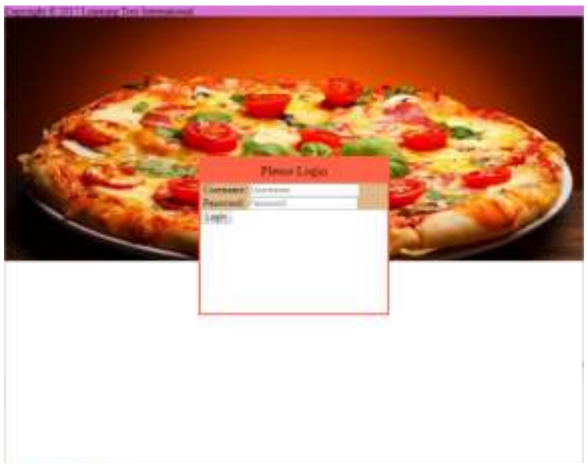


11. ☐ Our login prompt will center in our screen regardless of content thanks to the `calc()` function. We can prove this by removing the `nav` from our current exercise to more closely resemble the horizontal navigation geometry we will create later.

    Open `login.html` in your IDE and comment out the `<nav>` element using HTML comments:

```
<!--
    <nav>…..</nav>
-->
```

12. ☐ Save all open files in your IDE and preview `login.html` in your browser. Even without the `<nav>` element, your login prompt centers to expectations:



Part 2

13. ☐ CSS3 functions are wonderful shortcuts to amazing results in the viewport. We will now leverage the `filter()` function by applying a black and white filter to the image of our young Mario Brothers in Milan.

Switch back to your IDE and open `index.html` from the Ex 3.2 project folder:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.2\index.html
```

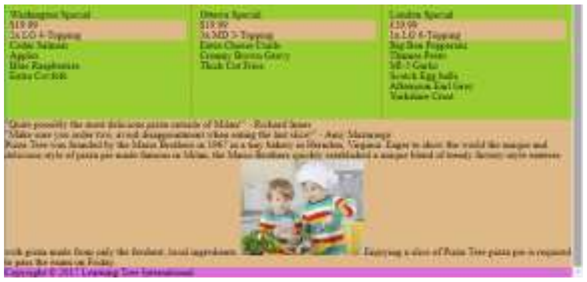
14. ☐ Scroll down to the text describing the hard-working Mario Brothers in the second article at around line 72. Between the second and third sentence, add the image of our two young superstar chefs. The image is located in the /img folder:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.2\img\mario_brothers.jpg
```

The code to add the image:

```
... established a unique blend of trendy factory-
style eateries with pizza made from only the
freshest, local ingredients. <img
src="img/mario_brothers.jpg" />Enjoying a slice of
Pizza Tree pizza pie is required to pass the exam
on Friday.
```

15. ☐ Save `index.html` and preview the same file in your browser:



Washington Special
$19.99
2x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
[illegible]

Ottawa Special
$19.99
2x MD 3-Topping
Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

London Special
£19.99
1x LG 6-Topping
Big Ben Pepperoni
[illegible]
MI-5 Garlic
Scotch Egg balls
[illegible] Earl Grey
Yorkshire Crust

"Quite possibly the most delicious pizza outside of Milan" - Richard [illegible]
"Make sure you order two, avoid disappointment when eating the last slice" - [illegible]
Pizza Tree was founded by the Mario Brothers in 1967 as a tiny bakery in Herndon, Virginia. Eager to share the world the unique and delicious style of pizza pie made famous in Milan, the Mario Brothers quickly established a unique blend of trendy factory-style eateries with pizza made from only the freshest, local ingredients. Enjoying a slice of Pizza Tree pizza pie is required to pass the exam on Friday.
Copyright © 2017 Learning Tree International

16. ☐ We don't have to re-render this image in Photoshop to achieve our desired black and white appearance—CSS3 Functions will do this for us.

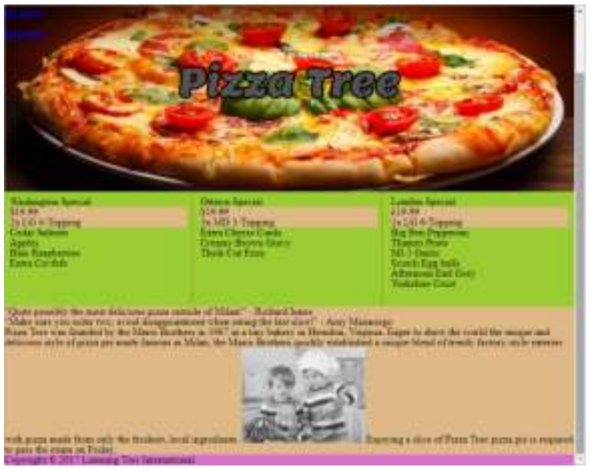
Open `default.css` located within the `/css` folder of the Ex 3.2 project:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex3.2\css\default.css
```

17. ☐ Add a selector for our new image placed in `index.html` and add a filter function that sets a grayscale value of 100%:

```
img {
    filter:grayscale(100%)
}
```

18. ☐ Save `default.css` and preview `index.html` in your browser:

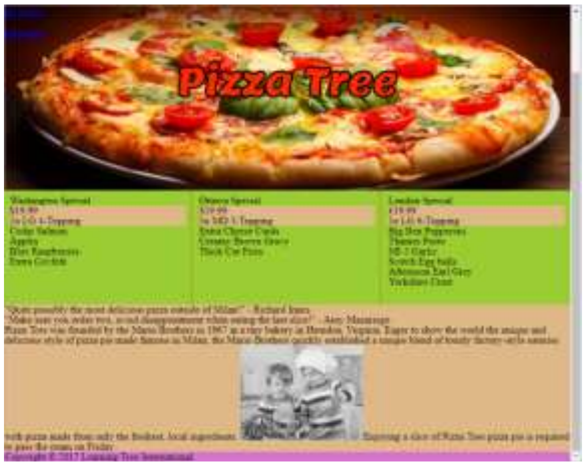


Well done! The Mario Brothers have just travelled back in time—but OOPS! Our Pizza Tree logo also happens to be an image, so it too inherited the grayscale filter.

19. ☐ Switch back to your IDE and `default.css` and fix the application of the grayscale filter to the Pizza Tree image by setting a more specific selector to the image selector we created in Step 18 that only targets the Mario Brothers image. There is no single solution, here is but one example:

```
p img:nth-of-type(1) {
    filter:grayscale(100%)
}
```

20. ☐ Save `default.css` and preview `index.html` in your browser:



Washington Special
$19.99
1x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
Extra Cornichons

Ottawa Special
$19.99
1x MD 3-Topping
Extra Cheese Curds
Creamy Beaver Gravy
Thick Cut Fries

London Special
£19.99
1x LG 6-Topping
Big Ben Pepperoni
Thames Prawn
HP-2 Garlic
Scotch Egg Balls
Afternoon Earl Grey
Yorkshire Crust

"Quite possibly the most delicious pizza outside of Milan!" - Richard Jones
"Make sure you order two, avoid disappointment when eating the last slice!" - Amy Mazzaringo
Pizza Tree was founded by the Mario Brothers in 1967 as a tiny bakery in Herndon, Virginia. Eager to show the world the unique and delicious style of pizza pie made famous in Milan, the Mario Brothers quickly established a unique blend of family factory-style [illegible]

with pizza made from only the freshest, local ingredients. Enjoying a slice of Pizza Tree pizza pie is a required to pass the exam on Friday

Copyright © 2017 Learning Tree International

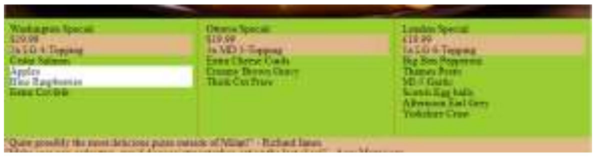
Part 3

21. ☐ Selector chaining, as we saw in our last exercise, is a great way to condense rules for quick results in the viewport.

Switch back to your IDE and `default.css`. Style the Washington Special toppings so that only Apples and Blue Raspberries have a white list-item background, without polluting the DOM markup with classes and IDs. You will need to create a selector that skips the first and last list items, and doesn't accidentally style the list-items within the `<nav>` item, nor does it style the list-items within the other Ottawa and London specials:

```
section:first-child li:not(:first-of-type):not(:last-of-type) {
    background:white;
}
```

22. ☐ Save `default.css` and preview `index.html` in your browser:



Washington Special
$19.99
1x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
[illegible] Cookies

Ottawa Special
$19.99
1x MD 3-Topping
Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

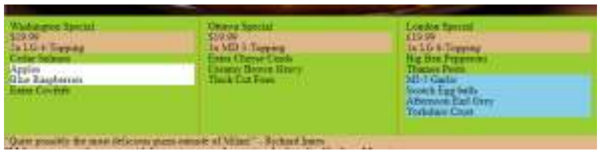
London Special
£19.99
1x LG 6-Topping
Big Ben Pepperoni
Thames Pesto
MI-5 Garlic
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

"Quite possibly the most delicious pizza outside of Milan!" – Richard James

23. ☐ Switch back to your IDE and `default.css` and continue styling the London Special so that only the last four toppings have a `skyblue` background. You will need to use a range selector chain and make sure only London is selected:

```
section:last-child li:nth-child(n+3):nth-child(-n+6) {
    background:skyblue;
}
```

24. ☐ Save `default.css` and preview `index.html` in your browser:



Washington Special
$19.99
1x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
[illegible] Cookies

Ottawa Special
$19.99
1x MD 3-Topping
Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

London Special
£19.99
1x LG 6-Topping
Big Ben Pepperoni
Thames Pesto
MI-5 Garlic
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

"Quite possibly the most delicious pizza outside of Milan!" – Richard James

Congratulations! You have used some advanced pseudo selector chains to select only the elements required for style without polluting the DOM markup with unnecessary classes and `ids`, and leveraged native CSS3 `calc()` and `filter()` functions to achieve dynamic viewport results.

STOP

This is the end of the exercise.

Overview

In this exercise, we will continue styling our navigation menu and upgrade our Specials to use flexbox vs the percentages we used in Hands-On Exercise 3.1. A small Pizza Tree logo has been added as the first list item that will serve as the mobile view header logo, and an HTML entity has been added as the last list item that will serve as the mobile menu trigger. Screenshots taken for this Exercise are in 910px viewport width.

Part 1: Re-structing the rendered navigation menu

1. ☐ Open the following files in your IDE. You may wish to close previous exercise files.

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.1\index.html
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.1\css\nav.css
```

2. ☐ In `index.html`, temporarily disable the callout we created in a previous exercise by commenting out the `<span>` element within the third list item near line 21:

```
<li><a href="#">Our Menu</a>
<!--<span class="callout">Now with gluten-free RWD!</span>-->
</li>
```

3. ☐ Switch to `nav.css`. Add a selector that floats the list items within the navigation menu to the left and provides 10px of left and right padding to prevent the words from floating into each other. Make your selector specific to avoid clashing with other list items. Add this selector under the first nav selector that terminates near line 8:

```
nav ul li {
    float:left;
    padding:0 10px;
}
```

4. ☐ Our goal is to replicate the transparent gradient background from the finished Case Study, which has a height of 4em. Our nav already has this background, which we created previously, but default user-agent styles mean the list items are only as tall as the unordered list, which in turn is only as tall as the nav. Style the unordered list to a height of 4em by adding a new unordered list selector. As a best practice, place less-specific styles before more-specific styles to make your code easier to read:

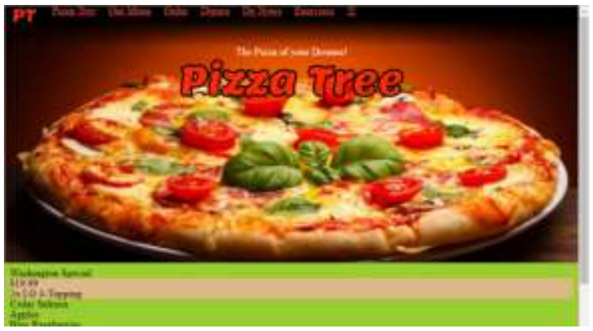
```
nav ul {
    height:4em;
}
```

Yes, we could have added a height property to the `nav` selector, but we'll need a `ul` selector later in the exercise, so we added it here with the same result.

5. ☐ Style the navigation anchors so we can see them. Set the height to match the height of the `<ul>` element and the width to 100% so the `<a>` elements fill the list items. `<a>` elements are inline elements by default, so we cannot assign dimension without converting them to block-level elements:

```
nav ul li a {
    display:inline-block;
    width:100%;
    height:4em;
    color:tomato;
}
```

6. ☐ Save `nav.css` and preview `index.html` in your browser:



Now that our `<a>` elements are set to inline-block, they can have dimension, which means our cursor will switch to a hand icon when any portion of the anchor is hovered over, not just the text. The anchors are now the same size as the list items, minus any padding within the list items. Move your cursor directly under the navigation link text to see this in action.

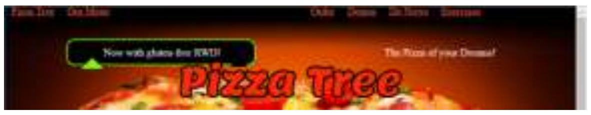
Press `<F><12>` in your browser to pop open the debugger and inspect the anchor elements to get a true visualization of their dimension.

7. ☐ We don't want to see the Pizza Tree logo or the menu logo when browsing on desktop devices, so disable them by setting their display properties to none using an appropriate pseudo selector for the list items in question (chain a `:not` selector or use a group selector):

```
nav ul li:last-of-type, nav ul li:first-of-type {
    display:none;
}
```

8. ☐ It's time to fix the borders of our callout using absolute positioning. Switch back to `index.html` and remove the comments you added in Step 2.

9. ☐ Save `index.html` and preview in a browser:

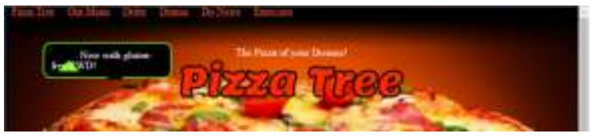


Now that our menu items are floating left, the callout is pushing subsequent menu items to the right.

10. ☐ Switch back to your IDE and `nav.css`. Prepend to the `.callout` selector properties for absolute position: 3.5em from the top, and 60px from the left:

```
.callout {
    position: absolute;
    top:3.5em;
    left:60px;
    ...
}
```

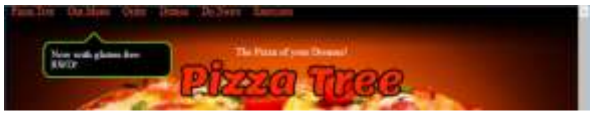
11. ☐ Save `nav.css` and preview `index.html`. The callout position should have changed, which directly affected the position of the pseudo element callout tails:



12. ☐ Switch back to your IDE and `nav.css`. Prepend the same absolute position to the .callout:before, .callout:after pseudo element selector, and set the bottom to 100% and left to 50%:

```
.callout:before, .callout:after  {
    position: absolute;
    bottom: 100%;
    left: 25%;
    border: solid transparent;
    content: "";
}
```

13. ☐ Save `nav.css` and preview `index.html` in your browser to reveal the completed callout:



Part 2: Restructuring Specials with flexbox

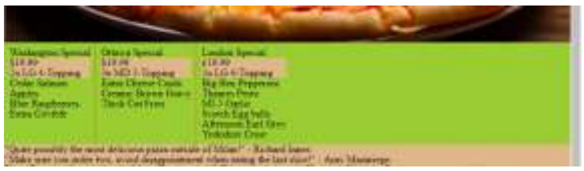
14. ☐ Return to your IDE and open `specials.css` from the `/css` folder of the Ex 4.1 project:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.1\css\specials.css
```

15. ☐ Within `specials.css`, set the article selector position to relative to serve as an anchor for descendant element positioning, and set the display to flex:

```
article#specials {
    background:yellowgreen;
    position:relative;
    display:flex;
}
```

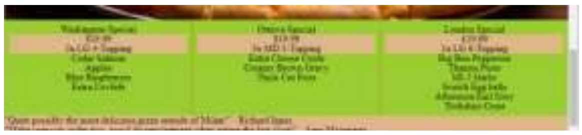
16. ☐ Save `specials.css` and preview `index.html` in your browser to see the proper way to create columns with CSS (you did a great job in a previous exercise but now you know how to convert multi-column layouts to flexbox!):



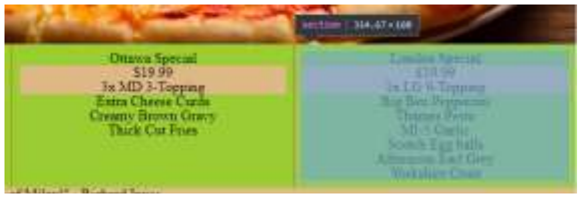
17. ☐ We have three child `<section>` elements, each representing a Special, and each requiring a flex property to occupy the requisite space. Return to `specials.css`, and append to the section selector properties to set `flex` to `1` and center the text:

```
section {
    background:yellowgreen;
    padding:8px;
    text-align:center;
    flex:1;
}
```

18. ☐ Save `default.css` and preview `index.html` in your browser:



19. ☐ Using your browser debugger, inspect each section to verify they are all the same height thanks to flexbox.

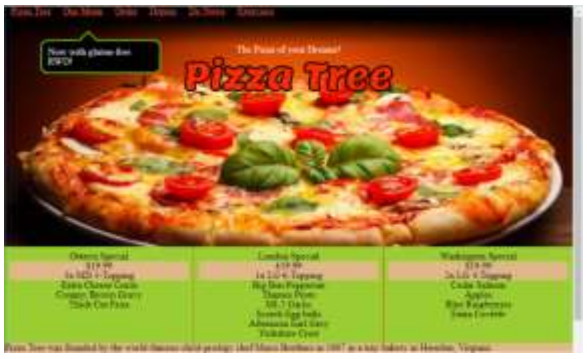


20. ☐ Flexible child elements can have their viewport display order updated by CSS. Return to `specials.css` in your IDE and, at the end of your file, add a selector that targets the first Special and sets its display order to `3`.

This will have the effect of making the first Special in the linear layout the last Special in the rendered DOM. Don't forget to update the right border of your new last Special by giving it a left border instead!

```
section:first-of-type {
    order:3;
    border-right:none;
    border-left:1px solid red;
}
```

21. ☐ Save `specials.css` and preview `index.html` in your browser:

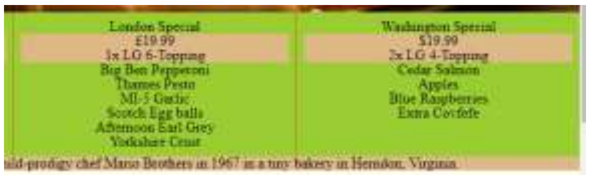


OOPS—now there is a 2px dotted border between the London and Washington Specials. The DOM re-order via flexbox has broken our border logic, which was not smart enough to adjust for the new DOM order on the fly.

22. ☐ Fix the double border by removing the pseudo selector decorator from the section selector, as we are no longer required to apply the right border to everything but the last element:

```
section {  /* removed :not(:last-of-type)  */
    border-right:1px dotted red;
}
```

23. ☐ Save `specials.css` and preview `index.html` in your browser:



London Special
£19.99
1x LG 6-Topping
Big Ben Pepperoni
Thames Pesto
MI-5 Garlic
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

Washington Special
$19.99
2x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
Extra Covfefe

ild-prodigy chef Mario Brothers in 1967 in a tiny bakery in Herndon, Virginia.

Congratulations! Creating multi-column, equal-height layouts has never been easier thanks to the addition of flexbox to CSS3.

Have you tested your solution in different viewports by varying the width of your browser? Yuck—it looks terrible in mobile (narrow) viewports! We will address responsive issues next.

This is the end of the exercise.

Overview

In this exercise, we will make our work responsive—our website needs to adapt to varying viewport widths. Note that some of your starting CSS files for this exercise have been slightly updated to catch up to this point in our case study for brevity.

Screenshots taken for this exercise are in 910px viewport width for desktop view and 420px viewport width for mobile view.

Part 1: Responsive navigation

1. ☐ Open the following files in your IDE:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.2\index.html
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.2\css\nav.css
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.2\css\specials.css
```

2. ☐ Create a new stylesheet named `responsive.css` within the `/css` folder of your Ex 4.2 project files:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex4.2\css\responsive.css
```

3. ☐ Add the appropriate link to `index.html` for this new stylesheet:

```
<link rel="stylesheet" href="css/testimonials.css" />
<link rel="stylesheet" href="css/responsive.css" />
<title>Pizza Tree - Badda Boom!</title>
```

4. ☐ Our `index.html` document needs to adapt to different viewports but following instructions on how to open on mobile devices. Directly after the first `<meta>` element near line 4, add a `<meta>` element with a viewport name attribute value and content set to adapt to the device width at an initial scale of 100%:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

5. ☐ Open your new `responsive.css` stylesheet. We will be adding two `@media` queries to fit the collapse requirements of our Case Study. Always start with the smallest viewport and work your way up to the largest viewport in your responsive stylesheet as part of a mobile first strategy.

   Add an `@media` query for screens that will apply to viewports less than or equal to 760px wide:

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

6. ☐ Add a selector within your new `@media` query for an `id` of `top` that sets the background size to cover the available width and height, maintaining perspective and only constraining when both have reached 100%. Adjust the image to be horizontally centered and raised by 250px:

```
#top {
    background-size:cover;
    background-position:center -250px;
}
```

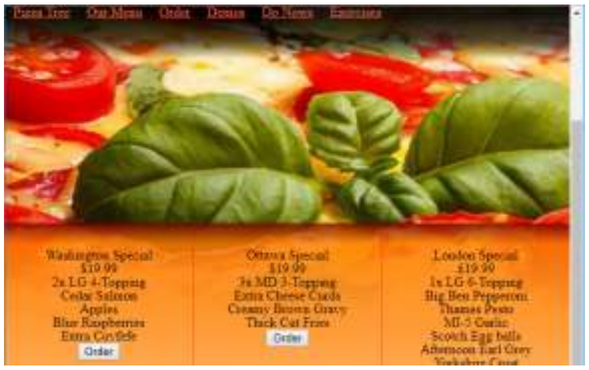
7. ☐ Hide the callout in mobile view (within the `@media` query):

```
.callout {
    display:none;
}
```

8. ☐ Our navigation menu should not scroll out of view on mobile devices, so fix it in place with the appropriate display property:

```
nav {
    position:fixed;
}
```

9. ☐ Save all open files and preview `index.html` in your browser. Narrow your viewport to less than 760px and scroll to ensure your menu is now “sticky”:



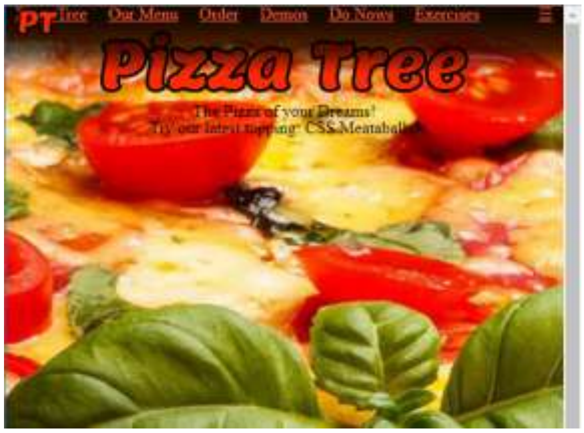
10. ☐ Return to `responsive.css` and set the last list item (our menu toggle) to display in our mobile viewport, and anchor it to the top right of our screen, letting the `<nav>` element background show through:

```
nav ul li:last-of-type {
    display:block;
    position:absolute;
    right:0; top:0;
    background:transparent;
}
```

11. ☐ On mobile devices, we want the Pizza Tree icon to be our corporate brand on the top left of our navigation. Add a mobile selector for the first list item and change its display from the previously assigned `none` to `block`. Anchor it to the top left, and let the `<nav>` element background show through:

```
nav ul li:first-of-type {
    display:block;
    position:absolute;
    left:0; top:0;
    background:transparent;
}
```

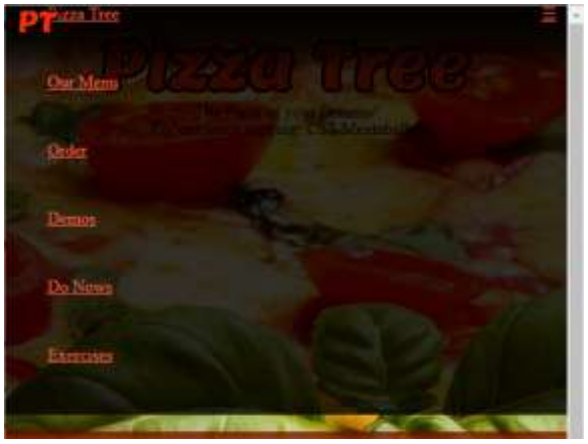
12. ☐ Save your work in `responsive.css` and preview `index.html` in your browser, making sure you test different viewport widths:


PT
Tree
Our Menu
Order
Demos
Do Nows
Exercises
Pizza Tree
The Pizza of your Dreams!
Try our latest topping: CSS Meatballs!

13. ☐ Our navigation links are still showing as floated list items in mobile view, which is not the desired behavior. Add a selector for all list items but the first and last to reset the float to none and add a translucent background to accompany some extra left and right padding of 40px:

```
nav li:not(:first-of-type):not(:last-of-type) {
    float:none;
    background:rgba(0,0,0,.8);
    padding:0 40px;
}
```

14. ☐ Save and preview `index.html` in your browser:



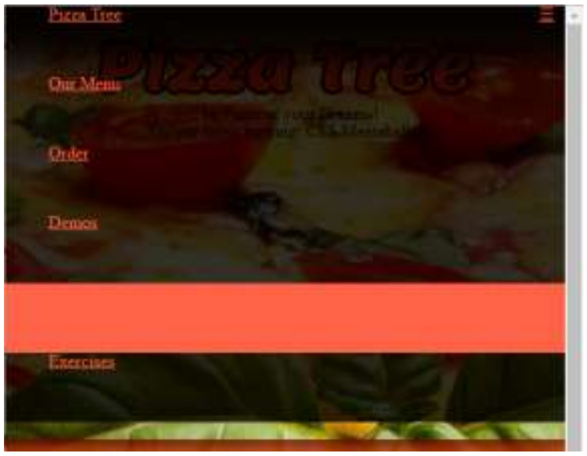
15. ☐ This menu will eventually only be displayed when the menu button is toggled, so for now, turn the Pizza Tree logo off on mobile until we have finished styling our menu:

```
nav ul li:first-of-type {
    display:none;
    position:absolute…
```

16. ☐ Add a pseudo selector on list items in the hover state that sets the background color to `tomato`. Be sure to select only menu items and not the menu toggle!

```
nav li:not(:last-of-type):hover {
    background:tomato;
}
```

17. ☐ Save your work and preview `index.html` in your browser, being sure to mouse over menu items in mobile view:



18. ☐ The hover effect has neutralized our menu text. Switch back to your IDE and `responsive.css`, and fix this by changing the menu text color on hover. This is easier said than done; you are changing the anchor color, not the list item color, which means adding another selector in hover state:

```
nav li:not(:last-of-type):hover a {
    color:black;
}
```

Part 2: Responsive flexbox

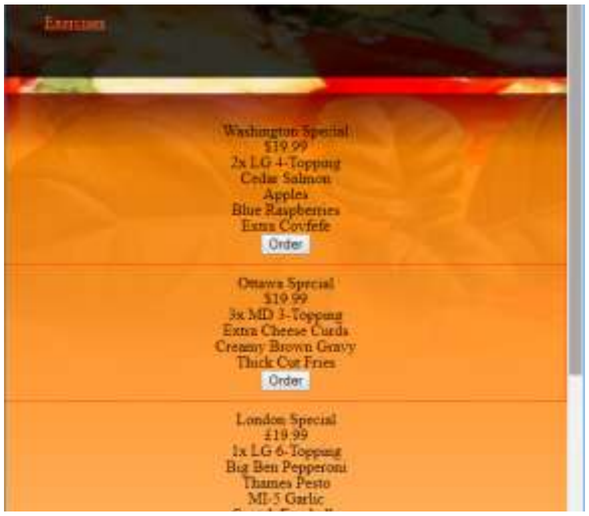
19. ☐ Our Specials are currently flexing in three columns, but that doesn't look good on mobile devices as there isn't enough room on smaller screens. Reset the mobile flex rules for our `<div>` element with an `id` of `specials` to wrap rows by adding a selector:

```
#specials {
    flex-flow:row wrap;
}
```

20. ☐ The child `<div>` elements within each Specials need to be converted to 100% wide flex items, and since we're stacking our Specials in mobile view, swap the right border for the bottom border in your new `div` selector:

```
#specials div {
    text-align:center;
    flex:1 100%;
    border-bottom:1px dotted red;
}
```

21. ☐ Save and preview `index.html` in your browser:



Washington Special
$19.99
2x LG 4-Topping
Cedar Salmon
Apples
Blue Raspberries
Extra Covfefe

Ottawa Special
$19.99
3x MD 3-Topping
Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

London Special
£19.99
1x LG 6-Topping
Big Ben Pepperoni
Thames Pesto
MI-5 Garlic

22. ☐ Once we have our menu toggle working, the focal point of the mobile view will be a delicious pizza and the Pizza Tree title image in our first view. It is not, however, currently visible, nor does it respond to larger viewports.

Disable the navigation menu by updating its display to none within `responsive.css` in your IDE:

```
nav {
    position:fixed;
    display:none;
}
```

23. ☐ Add selectors for the heading level 2 and 3 elements in the feature div to change size based on the width of the viewport:

```
div#feature h2 {
    font-size:5vw
}
div#feature h3 {
    font-size:4vw
}
```

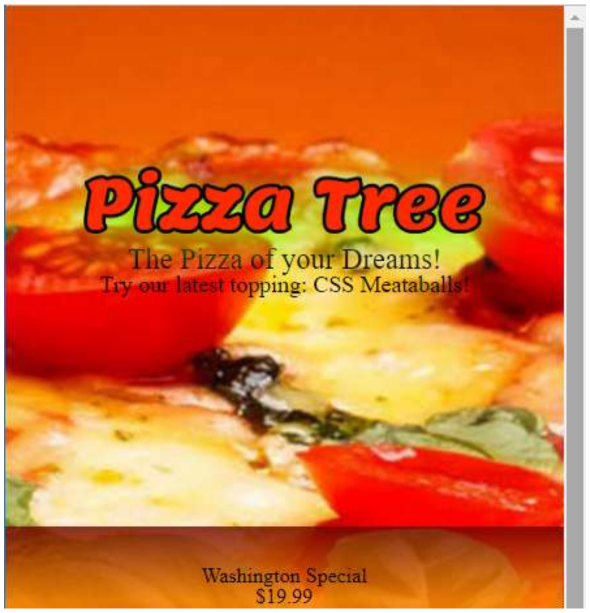
24. ☐ Add a selector that shows off our logo by setting the size of our main Pizza Tree image to 75% of its full width and applies a CSS3 filter() function to drop a lawngreen shadow—that's right, we're dropping a shadow from a transparent image:

```
div#feature img {
    width:75%;
    filter: drop-shadow(6px 6px 16px lawngreen);
}
```

25. ☐ Move the Featured Content down by adding a selector for our `<div id="feature">` element (before all other `#feature` selectors) with a top padding property of 100px:

```
div#feature {
    position:relative;
    top:100px;
}
```

26. ☐ Save your work and preview `index.html` in your browser:


Pizza Tree
The Pizza of your Dreams!
Try our latest topping: CSS Meataballs!
Washington Special
$19.99

Congratulations! Your menu and homepage is now responsive. We still have to make the Menu and Order form responsive, and our menu needs help from JavaScript to be truly responsive, but this will be accomplished in coming chapters. Updating the Typography of our Case Study comes first.

STOP

This is the end of the exercise.

Overview

In this exercise, we will initiate typography norms across our case study by leveraging Type CDNs. We will also take advantage of the many type and font properties to customize the look and feel of our navigation, specials and testimonials.

Screenshots taken for this Exercise are in 910px viewport width for desktop view, 420px viewport width for mobile view.

Part 1: General Case Study Fonts

1. ☐ Open the following files in your IDE:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex5.1\index.html
C:\inetpub\wwwroot\pizzatree\Ex\Ex5.1\css\nav.css
C:\inetpub\wwwroot\pizzatree\Ex\Ex5.1\css\responsive.css
```

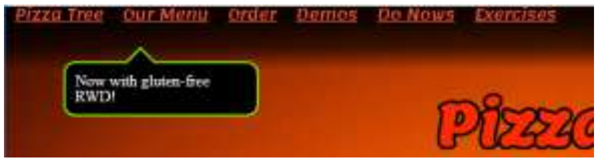
2. ☐ Within `index.html` and directly after the `<link>` element that assigns our favicon near line 6, add stylesheet links for Lato and Lemonada (available through Google CDN) and the Font Awesome icon stylesheet (available as both CDN and direct file download—we have opted for the download for variety):

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Lemonada" />
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Lato" />
<link rel="stylesheet"
href="css/font-awesome-4.7.0/css/font-awesome.css" />
```

3. ☐ Switch to `nav.css` in your IDE and add a font-family property to the `nav ul li a` contextual selector near line 21 that uses the Lemonada font and provides a fallback to cursive:

```
font-family:Lemonada, cursive;
```

4. ☐ Save all open files and preview `index.html` in a browser to ensure the Lemonada font is loading:



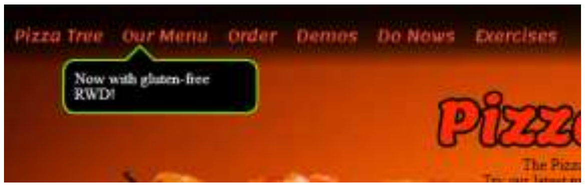
5. ☐ Switch back to your IDE and `nav.css` and within the same `nav ul li a` contextual selector, style the text to appear with the horizontal center by matching the line-height to the element height, centering the text, and removing the underline:

```
line-height:4em;
text-align:center;
text-decoration:none;
```

6. ☐ The screenshot in Step 4 shows the callout is not quite pointing to the Our Menu text—fix this by adjusting the callout left position in the `:before` and `:after` grouped pseudo selector near line 41 of `default.css`—you'll have to open this file from the `/css` folder in the Ex 5.1 project files and edit it appropriately (try 25% but this is splitting hairs):

```
.callout:before, .callout:after  {
position: absolute;
bottom: 100%;
    left: 25%;
    border: solid transparent;
    content: "";
}
```

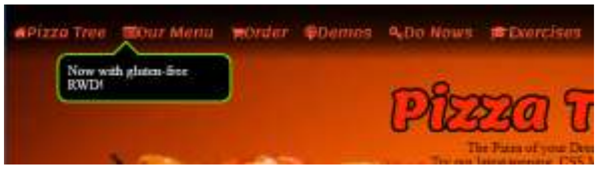
7. ☐ Save all open files and preview `index.html` in your browser, taking note of the active mouse geometry of your `<a>` elements within the navigation menu:



8. ☐ Now we'll add icons from the Font Awesome repository. Switch back to your IDE and `index.html`. For each list item in the navigation menu except the first and last ones, add `<i>` elements with classes that map to Font Awesome CSS entities using the class `fa` for every class, along with a custom icon class unique to each element:

```
<li>
   <a href="index.html">
      <i class="fa fa-home"></i>Pizza Tree
   </a>
</li>
<li>
   <a href="menu.html">
      <i class="fa fa-list-alt"></i>Our Menu
   </a>
   <span class="callout">Now with gluten-free RWD!</span>
</li>
<li>
   <a href="order.html">
      <i class="fa fa-shopping-cart"></i>Order
   </a>
</li>
<li>
   <a href="demos.html">
      <i class="fa fa-podcast"></i>Demos
   </a>
</li>
<li>
   <a href="donows.html">
      <i class="fa fa-key"></i>Do Nows
   </a>
</li>
<li>
   <a href="exercises.html">
      <i class="fa fa-mortar-board"></i>Exercises
   </a>
</li>
```

9. ☐ Save your work and preview `index.html` in a browser:



10. ☐ Switch back to your IDE and `nav.css` and change the new icon color to white. Give each icon some right padding to avoid the crushed effect with the navigation menu text:

```
nav ul li a i {
    color:snow;
    padding:0 4px 0 0
}
```

11. ☐ Finally, change the callout typeface (within `default.css`) to Lato, align the text within the callout to the center, and change the font-size to 1.2em, as Lato is considerably smaller than the default font size:

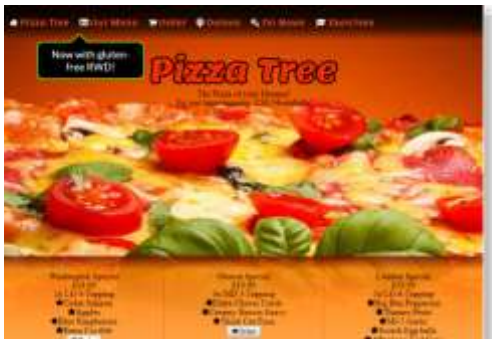
```
.callout {
    ...padding:10px;
    font-family:Lato;
    font-size:1.2em;
    line-height:1.2em;
    text-align:center;
}
```

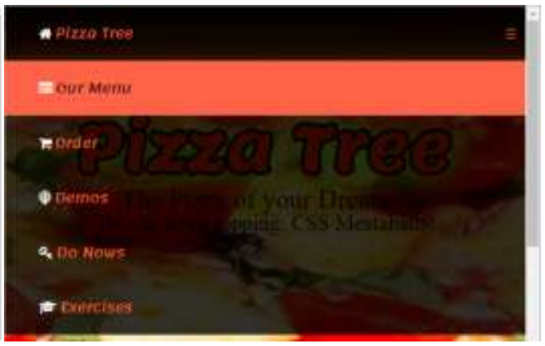
12. ☐ Switch to `responsive.css` and find the `nav` selector. Remove the property that is hiding the navigation menu from mobile view (`display:none`).

13. ☐ Under the last `nav` selector near line 34, add another `nav` selector for child anchors (be specific) that left-justifies the text:

```
nav ul li a {
    text-align:left;
}
```

14. ☐ Save all open files and preview `index.html` in your browser in both viewports:





Part 2: Styling the Feature and Specials Text

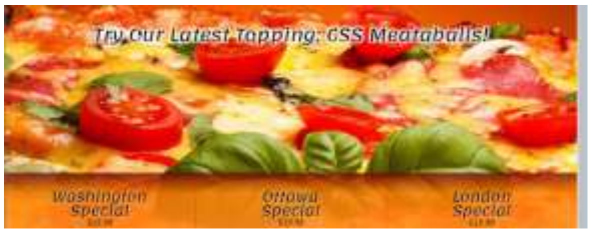
15. ☐ Switch back to your IDE and open the `feature.css` stylesheet. Add a selector that targets level 3 headings. Add 40px to the top padding, set the font-size to 4vw, and put a white text-shadow of 3px on all four sides:

```
div#feature h3 {
    padding-top:40px;
    font-size:3vw;
    text-shadow:3px 3px 3px white,
                -3px 3px 3px white,
                3px -3px 3px white,
                -3px -3px 3px white;
}
```

16. ☐ Open `specials.css` and add a new selector at the bottom of the stylesheet. Style the level 3 headings with a Capitalized Lemonada font (add a cursive fallback), set the font size to 1.5em, add a 1px white drop shadow to the bottom right, and set the white space property to pre-wrap, which will create a line break in the viewport where a soft break exists in the HTML code:

```
#specials div h3 {
    font-family:Lemonada, cursive;
    font-size:1.5em;
    white-space: pre-wrap;
    text-shadow:1px 1px 2px white;
    text-transform:capitalize;
}
```

17. ☐ Save all open files and preview `index.html` in a browser:



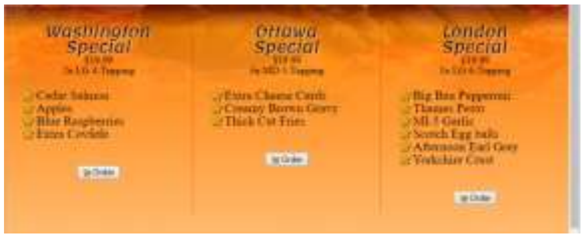
18. ☐ Return to your IDE and in `specials.css`, add a new contextual selector to style the unordered list with left justification, top and bottom padding of 20px, a 20px bottom margin, and a font-size of 1.25em:

```
#specials div ul {
    padding:20px 0;
    text-align:left;
    width:12em;
    margin:0 auto 20px auto;
    font-size:1.25em
}
```

19. ☐ Style the star icons within `<i>` elements gold with a black drop shadow to make them stand out:

```
#specials div i {
    padding-right:4px;
    color:goldenrod;
    text-shadow:1px 1px 1px black;
}
```

20. ☐ Save `specials.css` and preview `index.html` in a browser:

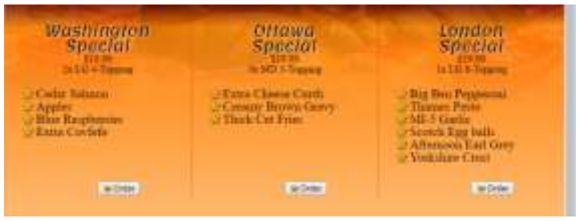


Note the shopping carts in the order buttons inherited the same goldenrod color!

21. ☐ Our buttons don't look right—flexbox has created equal height columns, but the buttons aren't aligned on the same axis. Fix this by switching back to your IDE and within `specials.css`, push them to the bottom of each Special by styling a button selector with absolute position, 0px from the bottom:

```
#specials button {
    position:absolute;
    bottom:0;
}
```

22. ☐ Save and preview `index.htm` in a browser:



Cedar Salmon
Apples
Blue Raspberries
Extra Cowbells

Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

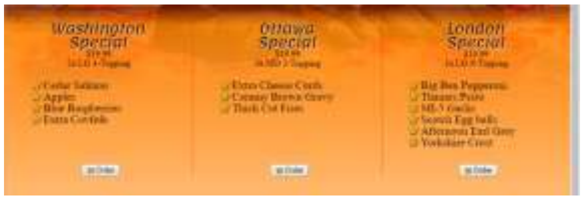
Big Ben Pepperoni
Thames Pesto
MI-5 Garlic
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

This looks much better, but the buttons are no longer horizontally centered within each Special. Absolute position elements are centered from their left edge if no left or right property is assigned. We can calculate how far to push the button over with a CSS3 function.

23. ☐ Switch back to your IDE and within `specials.css`, update the button selector to add a `right` property that uses the `calc()` function to subtract 2.75em from 50% of the parent element width:

```
#specials button {
    position:absolute;
    bottom:0;
    right:calc(50% - 2.75em);
}
```

24. ☐ Save and preview `index.html` in a browser:



Cedar Salmon
Apples
Blue Raspberries
Extra Cowbells

Extra Cheese Curds
Creamy Brown Gravy
Thick Cut Fries

Big Ben Pepperoni
Thames Pesto
MI-5 Garlic
Scotch Egg balls
Afternoon Earl Grey
Yorkshire Crust

25. ☐ Moving along to the testimonials, switch back to your IDE and open `testimonials.css`. Each aside in the HTML DOM contains an `<i class=”fa fa-quote-left”>` element for styling with a Font-Awesome “Quote” icon. Style this icon for effect by making it much larger (4em) than the testimonial text. Make it transparent by adding an rgba color set to 0.2 opacity to the new selector:

```
aside i.fa-quote-left {
    font-size: 4em;
    color:rgba(0,0,0,.2)
}
```

26. ☐ Update the paragraph text for the testimonial with a paragraph selector that uses the same Lemonada font-family used previously:

```
aside p {
    font-family: Lemonada, cursive
}
```

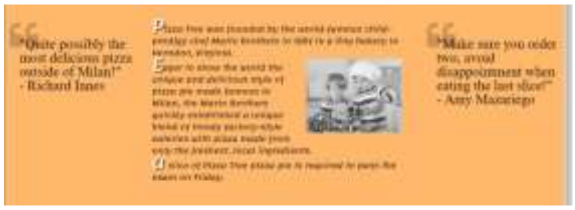
27. ☐ Add a special touch to the first letter of each `<p>` element within the Company History by changing their font size to 2.25em, the color to white, and a black text drop shadow of 2px:

```
aside p:first-letter {
   font-size:2.25em;
   color:white;
   text-shadow:2px 2px 2px black;
}
```

28. ☐ Finally, move the testimonial text into the blockquote icon by nudging the top margin -30px and add 20px to the left margin. Finish styling the text by setting the font size to 1.75em:

```
blockquote {
   margin:-30px 0 0 20px;
   font-size:1.75em;
}
```

29. ☐ Save all open files and preview `index.html` in your browser in both viewports:



"Quite possibly the most delicious pizza outside of Milan!"
- Richard Innes

[illegible]

"Make sure you order two, avoid disappointment when eating the last slice!"
- Amy Mazariego

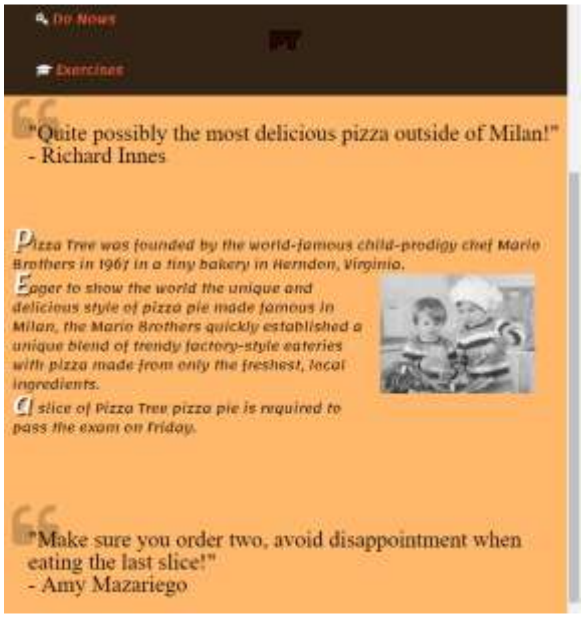
30. ☐ We need to clean up the mobile testimonial view. Switch back to `responsive.css` in your IDE and add a selector for `#testimonials` that assigns a flex flow property that wraps rows, much like we did for our specials:

```
#testimonials {
    flex-flow:row wrap;
}
```

31. ☐ Sandwich the Company History between two testimonials in mobile single-column view by changing the display order of our flexbox items as in the following code:

```
aside:nth-of-type(1) {
    order:2;
    flex:1 100%;
    line-height:1.5em;
}
aside:nth-of-type(2) {
    order:1;
    flex:1 100%
}
aside:nth-of-type(3) {
    order:3;
    flex:1 100%
}
```

32. ☐ Save and preview `index.html` in your mobile browser viewport:



"Quite possibly the most delicious pizza outside of Milan!"
- Richard Innes

Pizza Tree was founded by the world-famous child-prodigy chef Mario Brothers in 1967 in a tiny bakery in Herndon, Virginia.
Eager to show the world the unique and delicious style of pizza pie made famous in Milan, the Mario Brothers quickly established a unique blend of trendy factory-style eateries with pizza made from only the freshest, local ingredients.
A slice of Pizza Tree pizza pie is required to pass the exam on Friday.

"Make sure you order two, avoid disappointment when eating the last slice!"
- Amy Mazariego

Congratulations! Your homepage typography is now completely custom and responsive. It’s time to move on to styling forms and data tables in our next chapter.

This is the end of the exercise.

Overview

In this exercise, we will style the order form. Our starting point is a valid HTML5 form with everything required to support an accessibility strategy, but it has no physical formatting and oddly enough looks much better in linear layout as seen by a screen reader than it does rendered in a modern browser. We will fix this and make the form responsive to prevent it collapsing into a mess in mobile viewports.

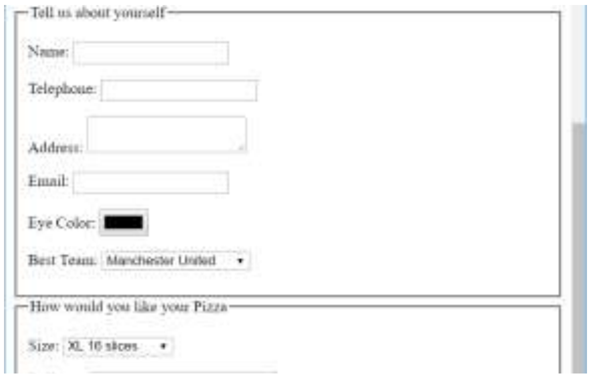
Screenshots taken for this Exercise are in 1200px viewport width for desktop view, 420px viewport width for mobile view.

Part 1: Desktop viewport

1. ☐ Open the following files in your IDE and Chrome (we need Chrome for this exercise):

   `C:\inetpub\wwwroot\pizzatree\Ex\Ex6.1\order.html`

2. ☐ In Chrome, after admiring the beauty of our order form, use the Web Developer extension to disable all styles. What you are looking at is an accessible form:



Tell us about yourself

Name:

Telephone:

Address:

Email:

Eye Color: ■

Best Team: Manchester United ▾

How would you like your Pizza

Size: XL 16 slices ▾

The `<fieldset>` element provides a container element for grouped `<label>` and `<input>` elements.

Each `<fieldset>` element has an accompanying `<legend>` element to describe the context of the grouped `<label>` and `<input>` elements.

All `<input>` elements have sibling `<label>` elements, which provide captions for data entry. All `<label>` elements feature a `for=""` attribute, which matches the `id=""` attribute of the `<input>`.

3. ☐ Switch back to your IDE and in the Ex 6.1 project files, create a new stylesheet for our form:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex6.1\css\form.css
```

4. ☐ Within `order.html` add a stylesheet link for our new form stylesheet, directly *before* the `responsive.css` stylesheet link tag:

```
<link rel="stylesheet" href="css/form.css" />
```

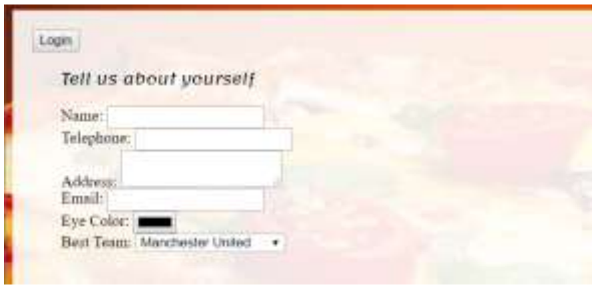
5. ☐ Open your new `form.css` stylesheet in your IDE. We must first style the `<form>` element to 1200px in width (this is an arbitrary number explained in Step 18), left justification, and centered using the automatic margins. Give your form a slightly transparent white background (90% white should do nicely) and assign a 20px padding for some breathing room:

```
form {
    text-align:left;
    width:1200px;
    margin:0 auto;
    background:rgba(255,255,255,.9);
    padding:20px;
}
```

6. ☐ Next, style the fieldset margins as 20px top, 40px bottom, 10px each for left and right, and add some padding of 20px. Create another selector for the legends and use our Lemonada font with the usual cursive fallback:

```
fieldset {
    margin:20px 10px 40px;
    font-family:Lemonada, cursive;
    padding:20px;
}
legend {
    font-family:Lemonada, cursive;
}
```

7. ☐ Save `form.css` and preview `order.html` in Chrome:



Login

Tell us about yourself

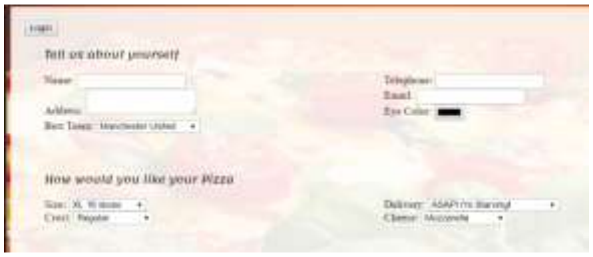
Name:
Telephone:
Address:
Email:
Eye Color: ■
Best Team: Manchester United ▾

8. ☐ There is too much white space. We can use that as a second column for our for elements. We'll use a `<p>` element as a physical container for each `<label>/<input>` contextual element pairing, and ensure two `<p>` elements can fit in the same row on desktop viewports.

Add a paragraph selector (specific to forms) that sets the width to 550px, and float to the left so we can wrap our odd numbered `<p>` elements:

```
form p {
    width:550px;
    float:left;
}
```

9. ☐ Save `form.css` and preview `order.html` in Chrome:



Login

Tell us about yourself

Name: Telephone:
Address: Email:
Best Team: Manchester United ▾ Eye Color: ■

How would you like your Pizza

Size: XL 16 inches ▾ Delivery: ASAP I'm Starving! ▾
Crust: Regular ▾ Cheese: Mozzarella ▾

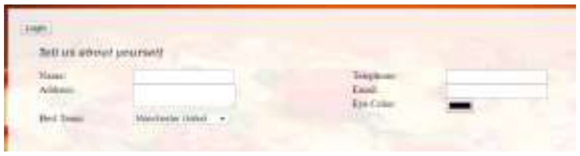
10. ☐ Switch back to your IDE and in `form.css`, add a group style that floats all labels, inputs, selects, and textareas to the left:

```
p label, p input, p select, p textarea {
    float:left;
}
```

11. ☐ Add another selector that clears the left edge of just the `<label>` elements, and sets their width to 150px:

```
p label {
    width:150px;
    clear:left;
}
```

12. ☐ Save your work and preview `order.html` in Chrome:



Login

Tell us about yourself

Name:

Address:

Best Team: Manchester United

Telephone:

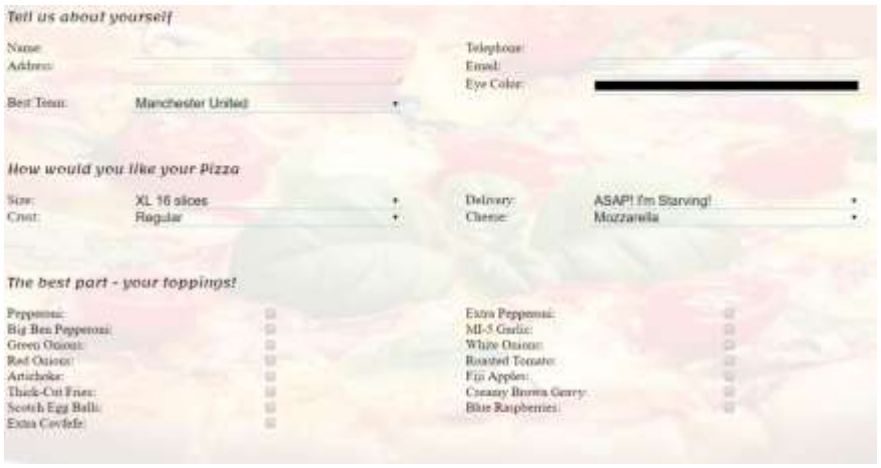
Email:

Eye Color:

13. ☐ Switch back to your IDE and `form.css`, and create a group style for inputs, selects, and textareas that does several things to achieve the look in the finished case study:
    - Set the border to `none`
    - Add a 1px skyblue bottom border
    - Make the background transparent
    - Assign 350px width
    - Set the font size and line-height to 1em

```
p input, p select, p textarea {
    border:none;
    border-bottom:1px solid skyblue;
    background:transparent;
    width: 325px;
    font-size:1em;
    line-height:1em;
}
```

14. ☐ Save your work and preview `order.html`:



Tell us about yourself

Name:		Telephone:	
Address:		Email:	
		Eye Color:	
Best Team:	Manchester United		

How would you like your Pizza

Size:	XL 16 slices	Delivery:	ASAP! I'm Starving!
Crust:	Regular	Cheese:	Mozzarella

The best part - your toppings!

Pepperoni:	Extra Pepperoni:
Big Ben Pepperoni:	MI-5 Garlic:
Green Onions:	White Onions:
Red Onions:	Roasted Tomato:
Artichoke:	Fiji Apples:
Thick-Cut Fries:	Creamy Brown Gravy:
Scotch Egg Balls:	Blue Raspberries:
Extra Covfefe:	

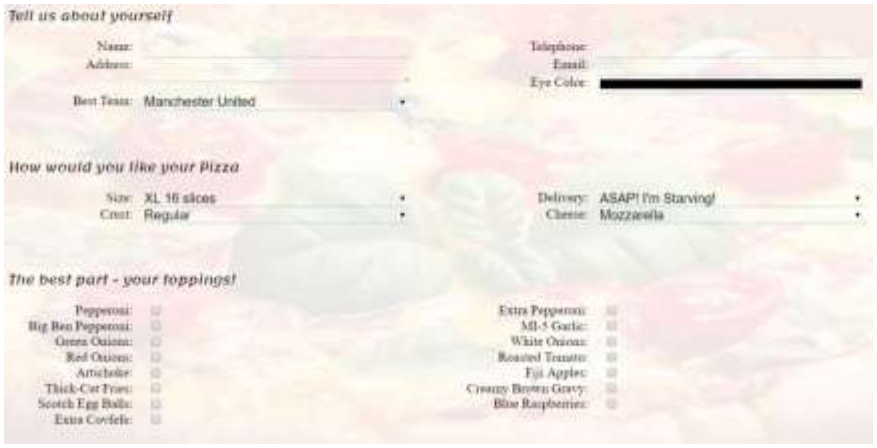
15. ☐ Align the existing label selector text to the right, and give it some right padding so it looks clean and clear, not crowded:

```
p label {
    width:150px;
    clear:left;
    text-align:right;
    padding-right:10px;
}
```

16. ☐ Finish our desktop viewport by adjusting the width of the checkboxes to 30px. They currently share the `<input>` element width of 350px, and checkboxes will center within any width, regardless of CSS text-align property:

```
p input[type="checkbox"] {
    width:30px;
}
```

17. ☐ Save your work and preview `order.html` in a browser:



Tell us about yourself

Name:		Telephone:	
Address:		Email:	
		Eye Color:	
Best Team:	Manchester United		

How would you like your Pizza

Size:	XL 16 slices	Delivery:	ASAP! I'm Starving!
Crust:	Regular	Cheese:	Mozzarella

The best part - your toppings!

Pepperoni:	☐	Extra Pepperoni:	☐
Big Ben Pepperoni:	☐	MI-5 Garlic:	☐
Green Onions:	☐	White Onions:	☐
Red Onions:	☐	Roasted Tomato:	☐
Artichoke:	☐	Fuji Apples:	☐
Thick-Cut Fries:	☐	Creamy Brown Gravy:	☐
Scotch Egg Balls:	☐	Blue Raspberries:	☐
Extra Covfefe:	☐		

Part 2: Responsive mobile viewport

18. ☐ Switch back to your IDE and open the `responsive.css` stylesheet for this exercise. The Pizza Tree Executive Team has decided anyone visiting on a tablet or smaller device should have the easiest form experience; therefore, we will collapse our form at any resolution under 1280px.

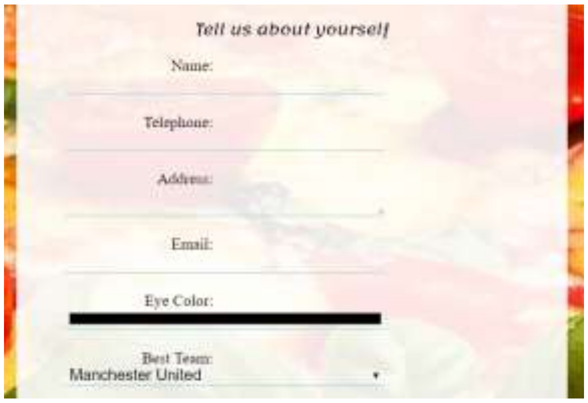
    Create a new `@media` query under the existing that accomplishes our goal. Within this new `@media` query, set the form width to 90% with centered legend content:

```
@media only screen and (max-width:1280px) {
    form {
        width:90%
    }
    legend {
        text-align:center;
    }
}
```

19. ☐ Our `<p>` elements have a fixed width for desktop viewports. Re-assign them to relative (100%), and add a 20px bottom border to help separate each label/input pair (add this and all remaining styles in our new `@media` query):

```
form p {
    width:100%;
    margin-bottom:20px;
}
```

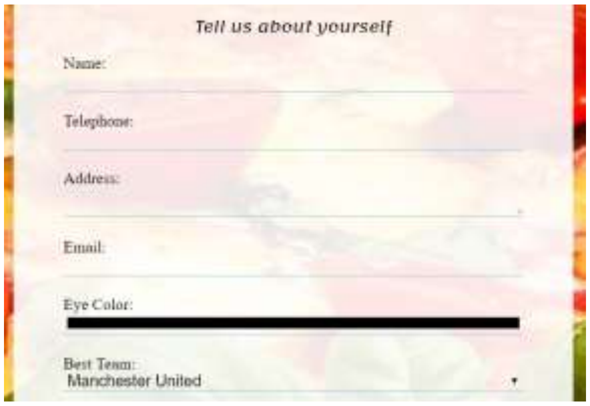
20. ☐ Save `responsive.css` and preview `order.html`:



21. ☐ We need to correct alignments and relative element sizing. Add a grouped selector that targets labels, inputs, textareas, and selects and stops the desktop float, applies left justification, and sets the width to 100%:

```
p label, p input, p textarea, p select {
    float:none;
    text-align:left;
    width:100%;
}
```

22. ☐ Save and preview `order.html` in Chrome:



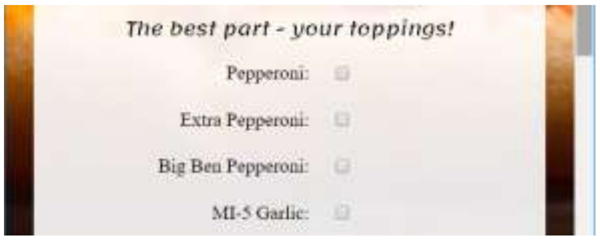
23. ☐ This looks much better, but scroll down to our toppings and we still have some leftover desktop properties to overrule. The toppings checkboxes are within `<p class=”topping”>` elements. Add a new selector for labels within this element that floats and clears left, sets the width to 55% and right-justifies text:

```
p.topping label {
    float:left;
    clear:left;
    width:55%;
    text-align:right;
}
```

24. ☐ Line the checkboxes up with their respective labels by adding a selector that floats the checkboxes left:

```
p.topping input {
    float:left;
}
```

25. ☐ Save and preview `orders.html` in Chrome:


The best part - your toppings!

Pepperoni: ☐

Extra Pepperoni: ☐

Big Ben Pepperoni: ☐

MI-5 Garlic: ☐

26. ☐ Switch back to your IDE and within `form.css`, style the buttons to match their appearance in the finished case study. Only peek at the finished selectors if you need help!

```
#order button {
    border-radius:8px;
    border-style:inset;
    padding:4px 20px;
    background:lawngreen;
    color:black;
    font-size:1em;
    position:relative;
    left:calc(100% - 10em)
}
#order button:hover {
    background:deepskyblue;
    cursor:pointer;
}
#order button:hover i {
    color:lawngreen
}
#order button[type="button"] {
    left:calc(100% - 7em)
}
```

Congratulations! Your order form is both accessible and responsive. We'll style our menu data table in the next exercise.

This is the end of the exercise.

Overview

In this exercise, we will style the menu table. Our starting point is a typical HTML4 form created by developers that did not prioritize accessibility, and our form predates responsive design, so it collapses into a giant unreadable mess on mobile.

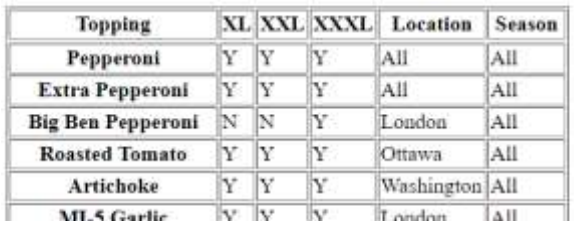
Screenshots taken for this Exercise are in 1200px viewport width for desktop view and 420px viewport width for mobile view.

Part 1: Desktop viewport

1. ☐ Open the following files in your IDE and Chrome:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex.6.2\menu.html
```

2. ☐ In Chrome, use the Web Developer extension to disable all styles. What you are looking at is a bare-bones HTML table:



Topping	XL	XXL	XXXL	Location	Season
Pepperoni	Y	Y	Y	All	All
Extra Pepperoni	Y	Y	Y	All	All
Big Ben Pepperoni	N	N	Y	London	All
Roasted Tomato	Y	Y	Y	Ottawa	All
Artichoke	Y	Y	Y	Washington	All
MI-5 Garlic	Y	Y	Y	London	All

3. ☐ Switch back to your IDE and in the Ex 6.2 project files, create a new stylesheet for our table:

```
C:\inetpub\wwwroot\pizzatree \Ex\Ex.6.2\css\table.css
```

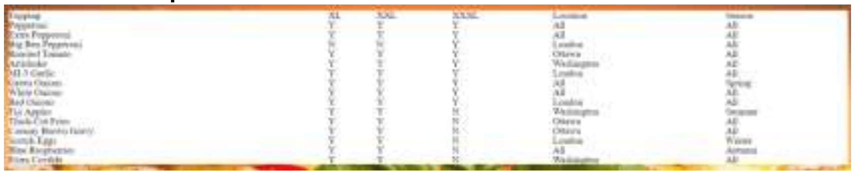
4. ☐ Within `menu.html` add a stylesheet link for our new form stylesheet before the link to `responsive.css`:

```
<link rel="stylesheet" href="css/table.css" />
```

5. ☐ Switch back to `table.css` and create the base style for the table element. Set the fixed maximum width to 1400px for non-mobile displays, make the background white, and set the top and bottom margins to 40px. Set the left and right margins to automatic, which will center the table in larger viewports, and set the text to left-justify, which will be inherited by child elements (the table cells):

```
table {
    width:1400px;
    background:white;
    margin:40px auto;
    text-align:left;
}
```

6. ☐ Save all open files and refresh `menu.html` in Chrome:



Topping	XL	XXL	XXXL	Location	Season
Pepperoni	Y	Y	Y	All	All
Extra Pepperoni	Y	Y	Y	All	All
Big Ben Pepperoni	N	N	Y	London	All
Roasted Tomato	Y	Y	Y	Ottawa	All
Artichoke	Y	Y	Y	Wellington	All
All-3 Garlic	Y	Y	Y	London	All
Green Onions	Y	Y	Y	All	Spring
White Onions	Y	Y	Y	All	All
Red Onions	Y	Y	Y	London	All
Fiji Apples	Y	Y	N	Wellington	Seasonal
Thick-Cut Fries	Y	Y	N	Ottawa	All
[illegible]	Y	Y	N	Ottawa	All
Scotch Eggs	Y	Y	N	London	Winter
Kiwi Raspberries	Y	Y	N	All	Autumn
[illegible]	Y	Y	N	Wellington	All

While the table is now slightly more pleasant on desktops, work remains to make the table not just accessible but responsive:

7. ☐ Switch back to your IDE and within `menu.html`, add a caption element to help screen readers with context for this menu table. Add your caption directly after the opening table element near line 45:

```
<table border="1">
    <caption>Our Menu</caption>
    <thead>
```

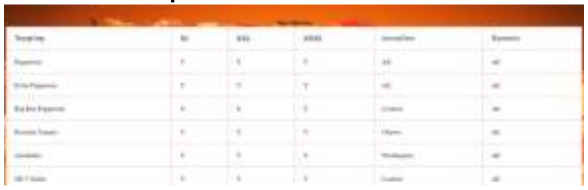
8. ☐ Within `table.css`, style the new caption as well as table header rows within the table header element with our Pizza Tree font:

```
caption,  thead th {
    font-family:"Lemonada", cursive
}
```

9. ☐ Style all `<tr>`, `<td>`, and `<th>` elements so they aren't crunched together. Add 20px of padding and assign a solid, 1px lightcoral border:

```
tr, th, td {
    padding:20px;
    border:1px solid lightcoral
}
```

10. ☐ Save all open files and refresh `menu.html` in Chrome:



Much better! Some zebra-striping would make text more visible across columns, and the non-heading text is too small for comfort.

11. ☐ Switch back to your IDE and within `table.css`, zebra stripe the table, being careful to only stripe the data rows and not the header (and potentially a future footer):

```
tbody tr:nth-child(odd){
    background:tomato
}
```

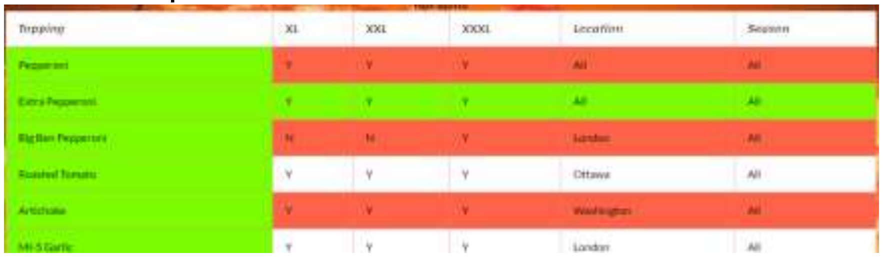
12. ☐ Style the text within the non-header data using the lato font, and slightly increase the font size:

```
tbody {
    font-size:1.1em;
    font-family:lato
}
```

13. ☐ Clarify the importance of the toppings as the key column by styling the background color to lawngreen for the first cell in every row, and apply a hover state to each row using the same background color:

```
tbody tr th,  tbody tr:hover {
    background:lawngreen
}
```

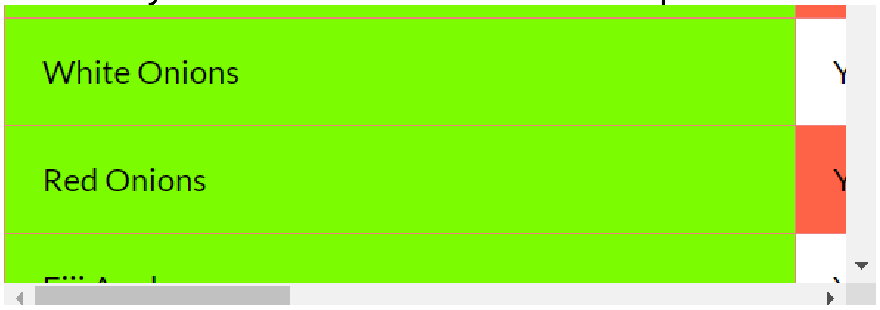
14. ☐ Save all open files and refresh `menu.html` in Chrome:



Topping	XL	XXL	XXXL	Location	Season
Pepperoni	Y	Y	Y	All	All
Extra Pepperoni	Y	Y	Y	All	All
Big Ben Pepperoni	N	N	Y	London	All
Roasted Tomato	Y	Y	Y	Ottawa	All
Artichoke	Y	Y	Y	Washington	All
MI-5 Garlic	Y	Y	Y	London	All

In this screenshot, the Extra Pepperoni row is in the hover state; any cell within the row should trigger the background of the entire row.

15. ☐ Preview your menu table in mobile viewport at around 420px:



White Onions

Red Onions

Yikes! Not only is this table not responsive, it's still not fully accessible.

16. ☐ Switch back to your IDE and within `menu.html`, help screen readers identify column headers by adding a column scope attribute to each `<th>` element within the `<thead>` element:

```
<th scope="col">Topping</th>
<th scope="col">XL</th>
<th scope="col">XXL</th>…
```

17. ☐ Still in `menu.html`, add a row scope attribute to each `<th>` element within the `<tbody>` element:

```
<th scope="row">Pepperoni</th>
<td>Y</td>
<td>Y</td>…
```

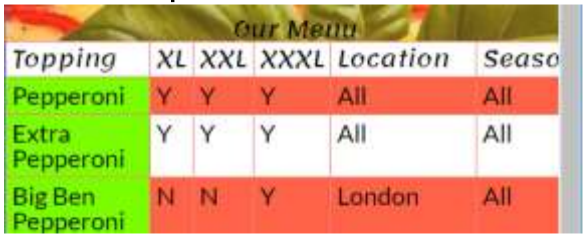
18. ☐ Open `responsive.css` from the Ex6.2 /css folder, and with the mobile media query near line 77, start styling our responsive table by setting the width to 90%:

```
table {
    width:90%;
}
```

19. ☐ We added padding to our desktop table earlier. We need to overrule this. Reduce the 20px down to 4px:

```
tr, th, td {
    padding:4px;
}
```

20. ☐ Save all open files and refresh `menu.html` in Chrome:



Our Menu

Topping	XL	XXL	XXXL	Location	Seaso
Pepperoni	Y	Y	Y	All	All
Extra Pepperoni	Y	Y	Y	All	All
Big Ben Pepperoni	N	N	Y	London	All

Despite the reduced padding and relative table width, there simply is not enough room aesthetically on mobile to display more than a few columns of data. We can fix this by converting our table to a single column, without touching the HTML.

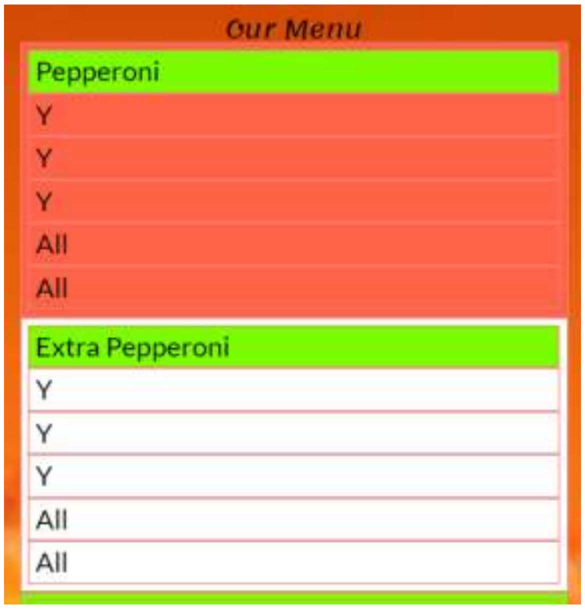
21. ☐ Remove the table header row that identifies the columns. We're only hiding with CSS; the original DOM is not being touched, which keeps it accessible:

```
table thead {
    display:none;
}
```

22. ☐ Style the table rows, headers and data cells as block-level elements (recall that block-level elements automatically expand in width to fit the geometry of their parent element):

```
table tr, table th, table td {
    display:block;
}
```

23. ☐ Save and refresh `menu.html` in Chrome:

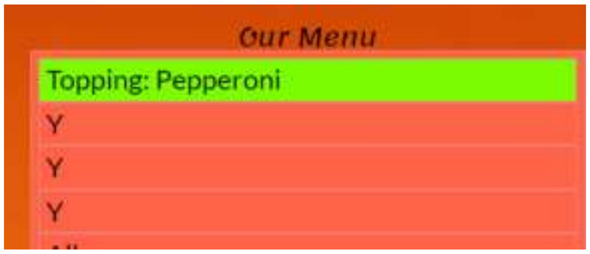


This is much easier to read on mobile devices, but now we're missing the important column headers.

24. ☐ Use CSS to add the column headers back by adding the header text as pseudo elements before the content of each table header. Start with Topping for the first header:

```
table th:nth-child(1):before{
    content: 'Topping: ';
}
```

25. ☐ Save and refresh `menu.html` in Chrome:



Our Menu

Topping: Pepperoni

Y

Y

Y

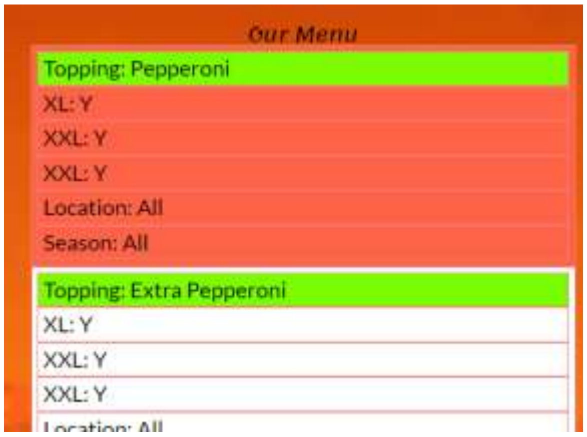
26. ☐ Now add the rest of the header text pseudo elements as in the following table:

Second `td`	**`'XL: '`**
Third `td`	**`'XXL: '`**
Fourth `td`	**`'XXXL: '`**
Fifth `td`	**`'Location: '`**
Sixth `td`	**`'Season: '`**

If you need help, here is the completed pseudo element CSS:

```
table th:nth-child(1):before{
    content: 'Topping: ';
}
table td:nth-child(2):before{
    content: 'XL: ';
}
table td:nth-child(3):before{
    content: 'XXL: ';
}
table td:nth-child(4):before{
    content: 'XXXL: ';
}
table td:nth-child(5):before{
    content: 'Location: ';
}
table td:nth-child(6):before{
    content: 'Season: ';
}
```

27. ☐ Save and refresh `menu.html` in Chrome:



Our Menu

Topping: Pepperoni
XL: Y
XXL: Y
XXL: Y
Location: All
Season: All

Topping: Extra Pepperoni
XL: Y
XXL: Y
XXL: Y
Location: All

Congratulations! Your menu table is both accessible and responsive.

STOP

This is the end of the exercise.

Overview

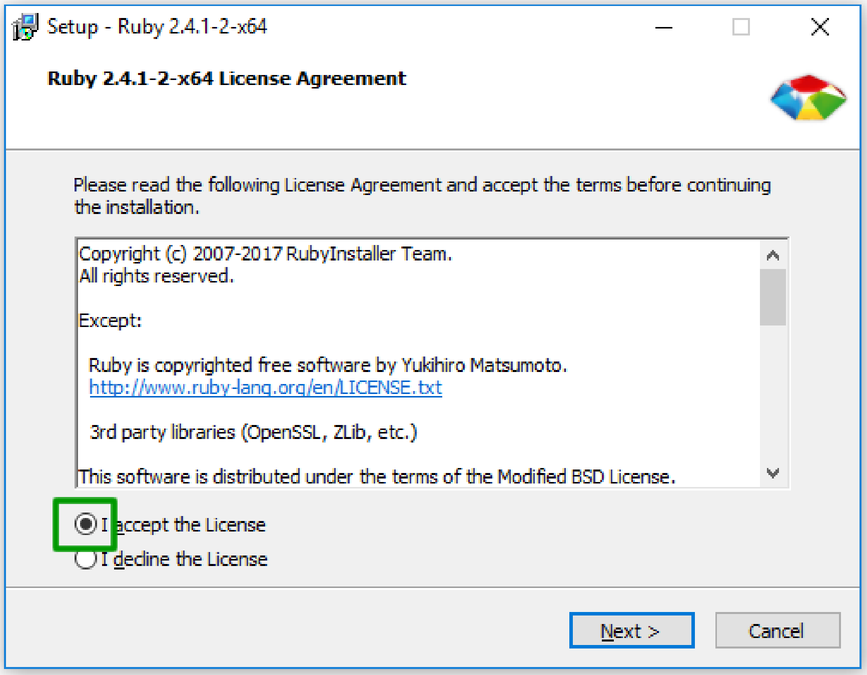
In this Exercise, we will install Ruby, and through Ruby install Sass. We will then create an HTML template and a SCSS template, and compile SCSS to CSS.

Warning! Be sure to follow the package installation steps exactly as directed, selecting only the options as seen in the screenshots, or your Ruby install will not complete.

1. ☐ Locate the `SOFTWARE` folder on your virtual machine desktop and double-click the following installation file:

```
rubyinstaller-2.4.1-2-x64.exe
```

2. ☐ From the License Agreement prompt, select **I accept the License** and click the **Next** button:



Ruby 2.4.1-2-x64 License Agreement

Please read the following License Agreement and accept the terms before continuing the installation.

Copyright (c) 2007-2017 RubyInstaller Team.
All rights reserved.

Except:

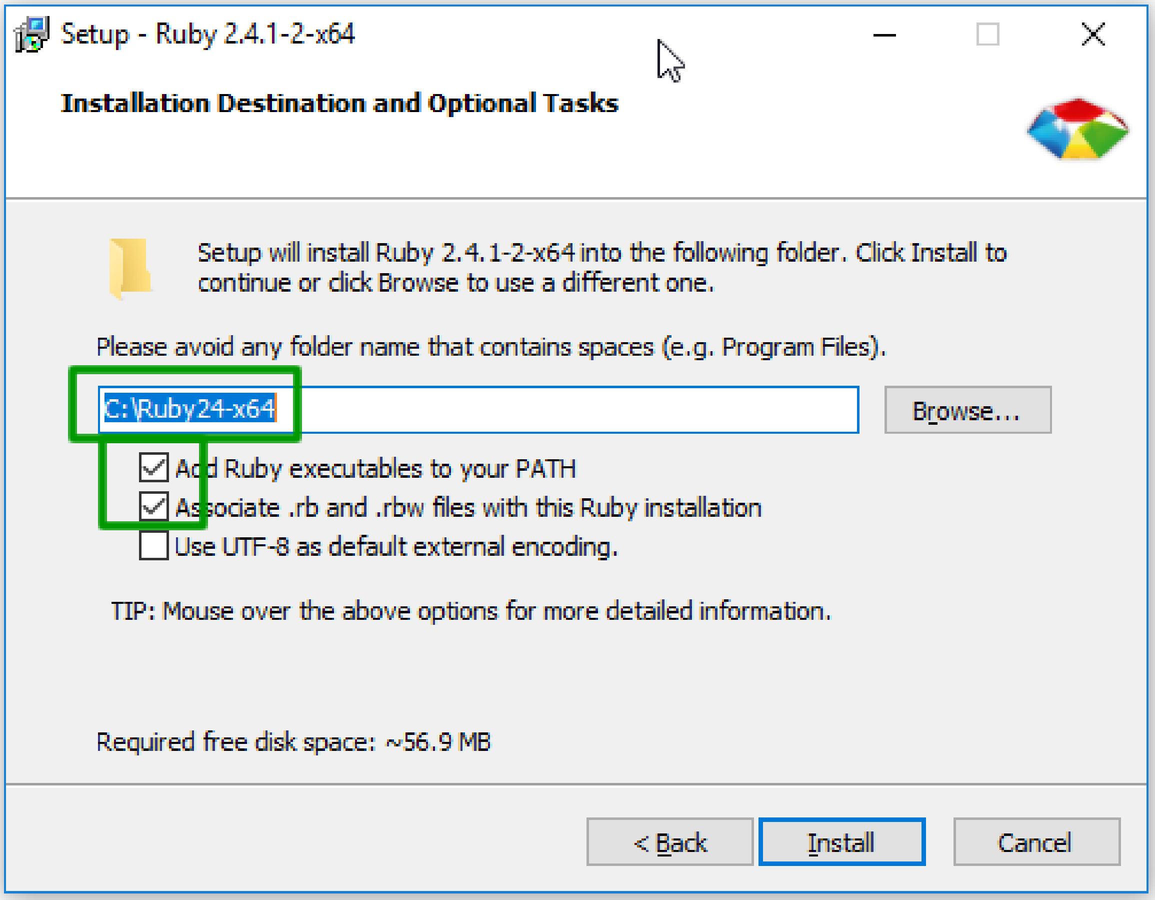
Ruby is copyrighted free software by Yukihiro Matsumoto.
http://www.ruby-lang.org/en/LICENSE.txt

3rd party libraries (OpenSSL, ZLib, etc.)

This software is distributed under the terms of the Modified BSD License.

[✓ I accept the License]
I decline the License

3. ☐ In the Installation Destination and Optional Tasks prompt, ensure the path is set to `C:\Ruby24-x64` and ensure only **Add Ruby executables to your PATH** and **Associate .rb. and .rbw files with this Ruby installation** are checked. Leave the path as `C:\Ruby24-x64`.



Setup will install Ruby 2.4.1-2-x64 into the following folder. Click Install to continue or click Browse to use a different one.

Please avoid any folder name that contains spaces (e.g. Program Files).

C:\Ruby24-x64

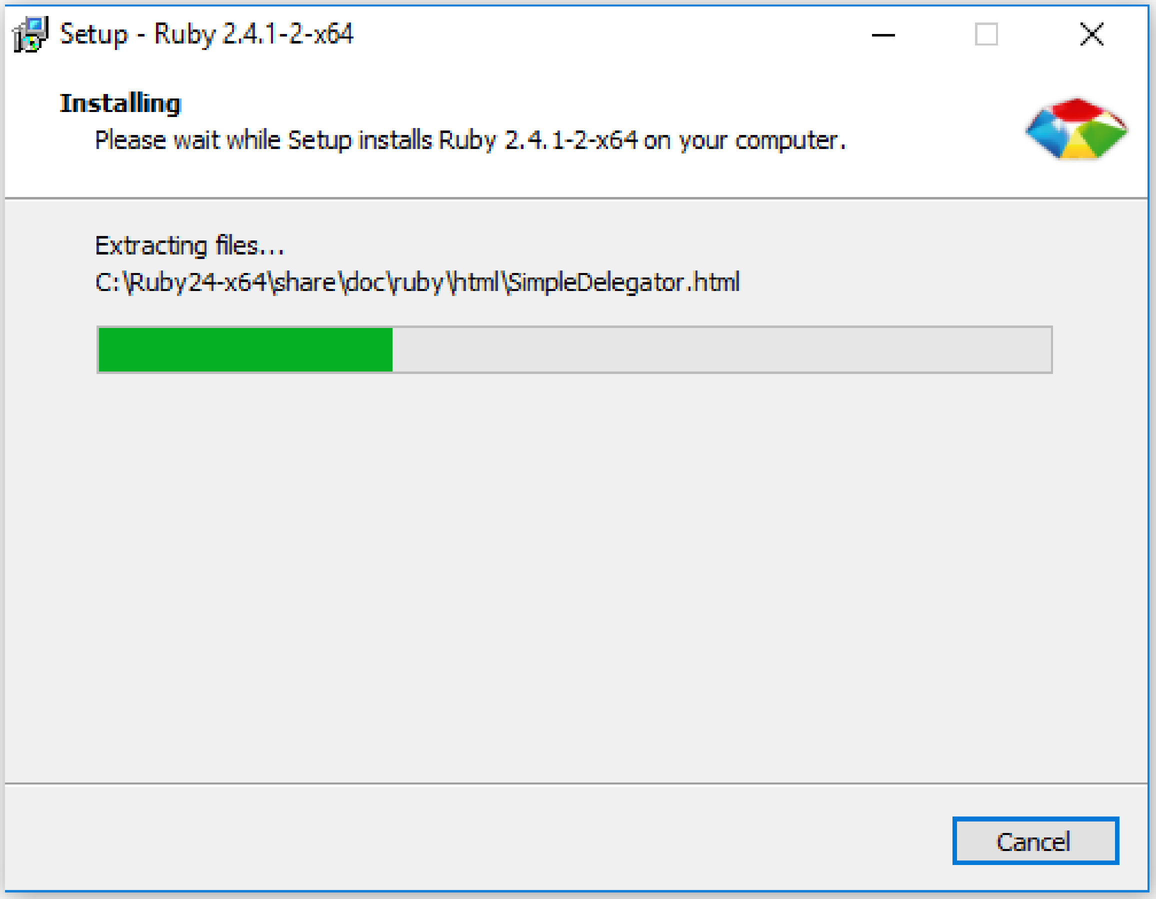
- ☑ Add Ruby executables to your PATH
- ☑ Associate .rb and .rbw files with this Ruby installation
- ☐ Use UTF-8 as default external encoding.

TIP: Mouse over the above options for more detailed information.

Required free disk space: ~56.9 MB

4. ☐ Click the **Install** button. You should see a progress bar indicating successful file operations.

Warning! Do not click Finish when prompted!



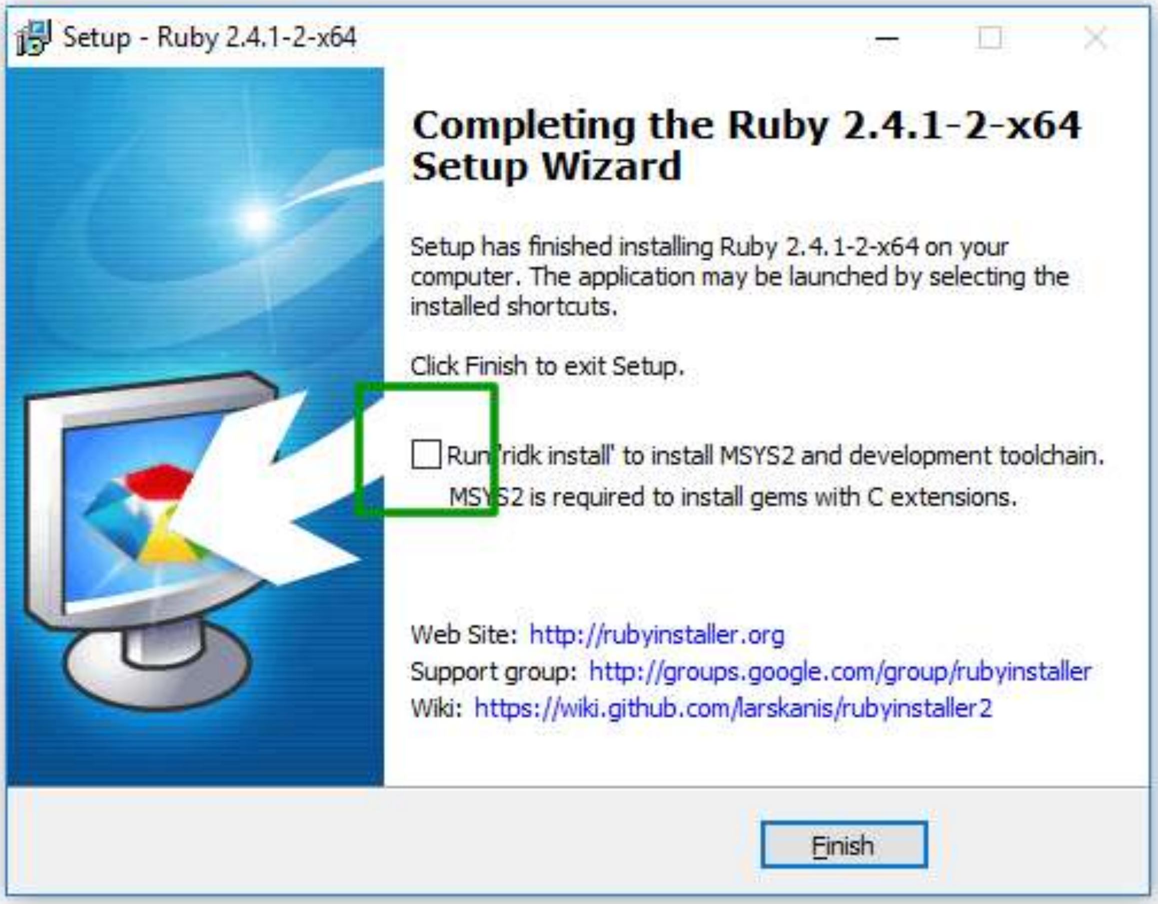
Setup - Ruby 2.4.1-2-x64

Installing

Please wait while Setup installs Ruby 2.4.1-2-x64 on your computer.

Extracting files...
C:\Ruby24-x64\share\doc\ruby\html\SimpleDelegator.html

Warning! Ensure the “Run ‘ridk install’ to install MSYS2 and development toolchain” option is *not* checked before clicking Finish!



Setup - Ruby 2.4.1-2-x64

Completing the Ruby 2.4.1-2-x64 Setup Wizard

Setup has finished installing Ruby 2.4.1-2-x64 on your computer. The application may be launched by selecting the installed shortcuts.

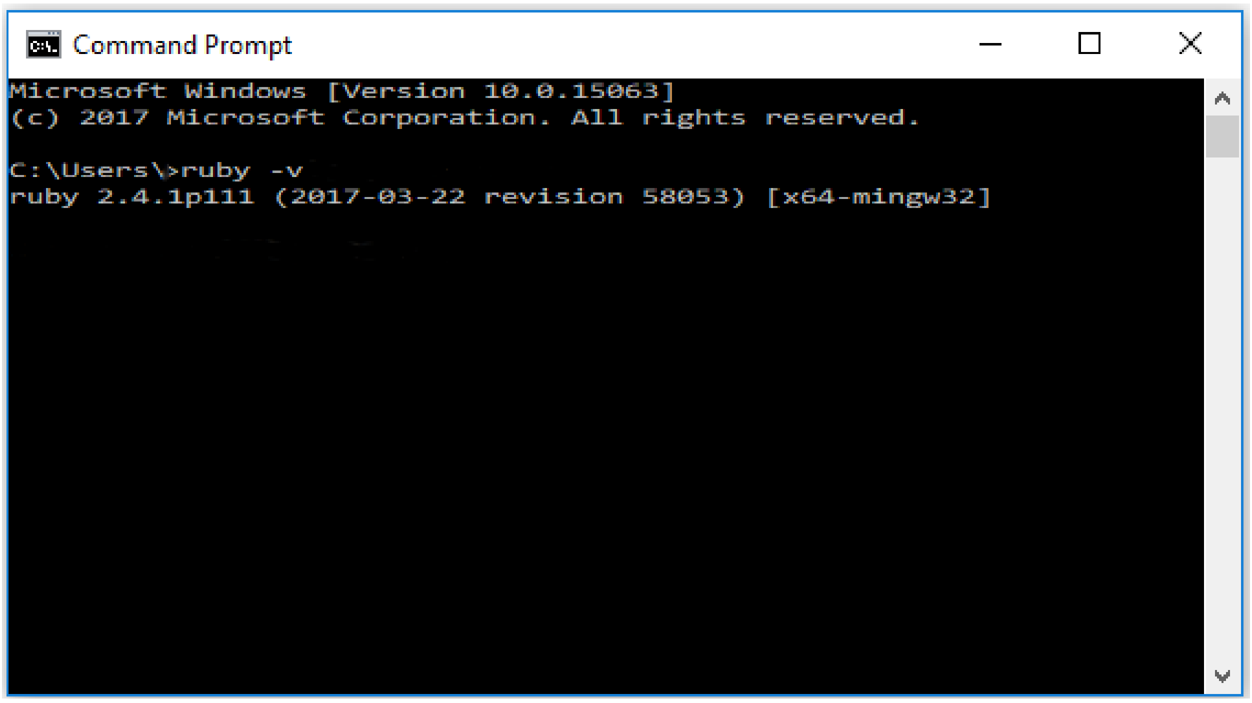
Click Finish to exit Setup.

☐ Run 'ridk install' to install MSYS2 and development toolchain. MSYS2 is required to install gems with C extensions.

Web Site: http://rubyinstaller.org
Support group: http://groups.google.com/group/rubyinstaller
Wiki: https://wiki.github.com/larskanis/rubyinstaller2

5. ☐ Ruby is now installed on your virtual machine. Let’s test your install by opening a command prompt (run as Administrator) and asking Ruby about itself:
   a. Press the magnifying glass icon next to the Windows button on the bottom left of your taskbar. Type in **`cmd`**, then press `<Enter>`. You should see a command prompt.
   b. At the command prompt, type: **`ruby -v`**

✓ *You should see the following response from Ruby, confirming Ruby 2.4.1 has successfully installed:*

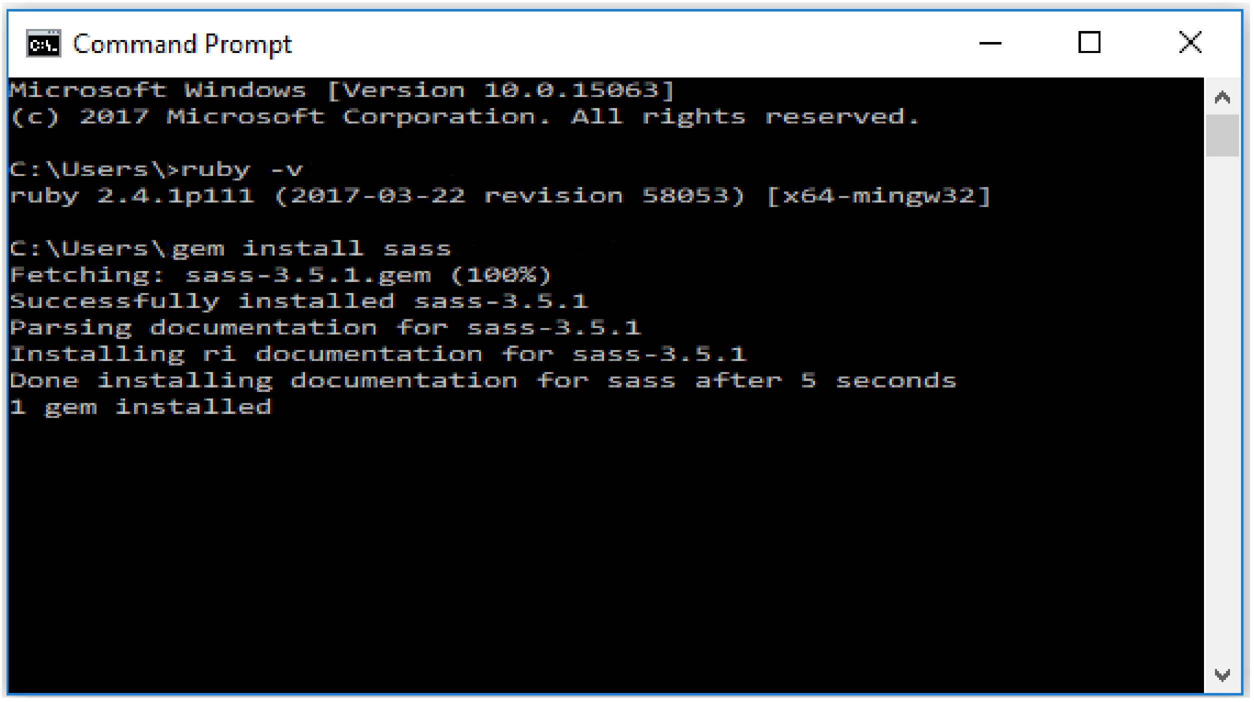


```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\>ruby -v
ruby 2.4.1p111 (2017-03-22 revision 58053) [x64-mingw32]
```

6. ☐ It's time to use Ruby to install Sass. If you've used npm and nodeJS, this will be very familiar. From the same command prompt, tell Ruby to search for, download and install a GEM package called Sass:

```
gem install sass
```

✓ *After a few seconds, Ruby should display a message that it is fetching sass-3.5.x.gem. If Windows Security alerts you to allow Ruby to run on Public Networks, click Allow. When it completes, you should see a summary of the install:*



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\>ruby -v
ruby 2.4.1p111 (2017-03-22 revision 58053) [x64-mingw32]

C:\Users\gem install sass
Fetching: sass-3.5.1.gem (100%)
Successfully installed sass-3.5.1
Parsing documentation for sass-3.5.1
Installing ri documentation for sass-3.5.1
Done installing documentation for sass after 5 seconds
1 gem installed
```

Congratulations! You've just installed Ruby and hooked the Sass GEM into Ruby's GEM portfolio.

7. ☐ Before proceeding with Sass, change the current directory in your command prompt to the Case Study directory (ask your Instructor for assistance if you have not previously used a command prompt):

```
\C:\inetpub\wwwroot\pizzatree
```

Leave this command prompt open as you proceed to the next step.

8. ☐ We can now set up our Sass environment by opening your IDE in Administrator Account Mode (Don't be shy asking your Instructor for guidance!) and creating a Sass test directory in the root Pizza Tree project folders. Name your new directory **`sass`.**

9. ☐ Within your new `/sass` directory, create the following template files:

`sass.html`	This file be our html template in which we'll see the results of our SCSS>CSS compilation.
`sassy.scss`	This is our SCSS template file, which we'll compile into CSS.
`sassy.css`	This CSS file will be managed by Sass.

10. ☐ Open your new `sass.html` file, which should be empty. In your IDE folder tree, goto `/Demos` and open `sass_starter_template.html`. Copy the full content of this file and paste it into our empty `sass.html` document. Here is the code:

```
<html>

    <head>
        <title>Welcome to SASSY CSS!</title>
    </head>

    <body>
        <div id="container">
        <p>I like to eat at Pizza Tree</p>
        </div>
        <div id="container2">
        <p>I like to get free pizza from Pizza Tree</p>
        </div>
    </body>

</html>
```

11. ☐ Within your newly-templated `sass.html` document (the one you created in the `/sass` folder) add a link to an external CSS document called `sassy.css` within the `<head>` element, under the `<title>` element.

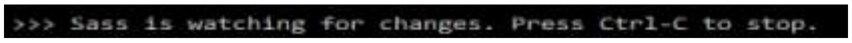
```
<link rel="stylesheet" href="sassy.css" />
```

12. ☐ Switch back to your command prompt and dive deeper into the folder structure by entering the new `/sass` folder, and tell Sass to watch the SCSS file for changes and compile into CSS when those changes occur.

```
sass -–watch sassy.scss:sassy.css
```

Note the double dash before the command to watch.

✓ *You should see a message from Sass:*



```
>>> Sass is watching for changes. Press Ctrl-C to stop.
```

If instead, you see an error message, you likely told Sass to watch the wrong directory! Fix this and try again.

13. ☐ Now it's time to write your SCSS. Open the SCSS file you created (`sassy.scss`) in an IDE. We'll start by creating some variables that can easily be shared by different selectors in our SCSS file:

```
$pM: 10px 4px;
```

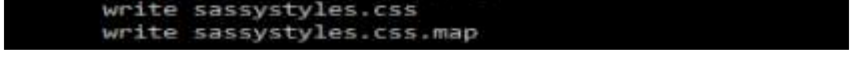
This will be our paragraph margin variable. Continue creating paragraph variables for color, height, width and border

```
$pC: tomato;
$pH: 215px;
$pW: 225px;
```

14. ☐ Under your new SCSS variables, add a paragraph selector that assigns the values of margin, color, height, width and border to the corresponding variables you created in the last step:

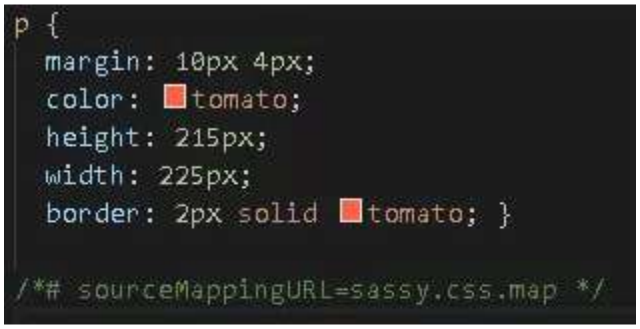
```
p {
   margin: $pM;
     color: $pC;
     height: $pH;
     width: $pW;
     border: 2px solid $pC; /* Use a variable here too! */
}
```

15. ☐ Save your SCSS file, and switch back to the command prompt. You should see a message from Sass:



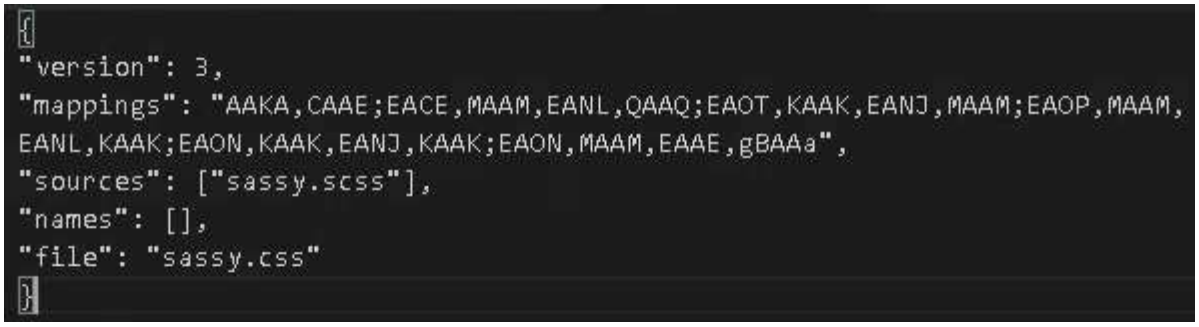
```
>>> Change detected to: sassystyles.scss
      write sassystyles.css
      write sassystyles.css.map
```

16. ☐ Switch to your IDE and examine the contents of your `/sass` folder. Open `sassy.css`, which was an empty file you created, and examine the CSS compiled by SCSS:



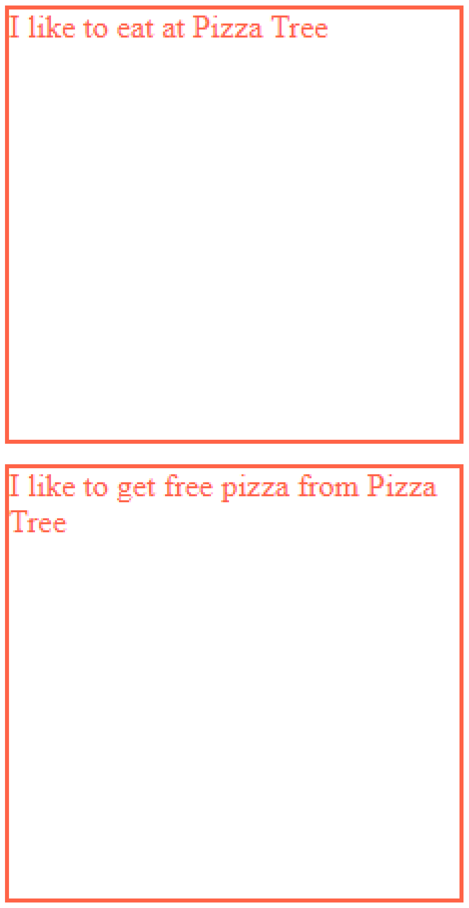
```
p {
  margin: 10px 4px;
  color: tomato;
  height: 215px;
  width: 225px;
  border: 2px solid tomato; }

/*# sourceMappingURL=sassy.css.map */
```

You should also see a `sassy.css.map` file. This is a very helpful JSON file your browser debugger uses to map compiled CSS line numbers back to corresponding line numbers in the original SCSS file:



```
{
"version": 3,
"mappings": "AAKA,CAAE;EACE,MAAM,EANL,QAAQ;EAOT,KAAK,EANJ,MAAM;EAOP,MAAM,
EANL,KAAK;EAON,KAAK,EANJ,KAAK;EAON,MAAM,EAAE,gBAAa",
"sources": ["sassy.scss"],
"names": [],
"file": "sassy.css"
}
```

17. ☐ Open the `sass.html` template file in a browser to see how your SCSS was compiled into fully compliant CSS. You should see a paragraph that looks like the following screenshot (notice that the second paragraph wraps—we'll fix that soon):


I like to eat at Pizza Tree

I like to get free pizza from Pizza Tree

18. ☐ Now let's create a reusable `@mixin` within `sassy.scss` by creating a common border style. An `@mixin` requires an identifier, in the same way a function requires an identifier. In fact, many people casually refer to `@mixin`s as functions. Declare this `@mixin` directly under your variables and above your standard CSS selectors:

```
@mixin tomatoBorder {
    border: 2px solid $pC
}
```

19. ☐ Edit the paragraph selector to include the new border `@mixin` instead of a hard-coded border (changes in bold):

```
p {
    margin: $pM;
    color: $pC;
    height:$pH;
    width: $pW;
    @include tomatoBorder;
}
```

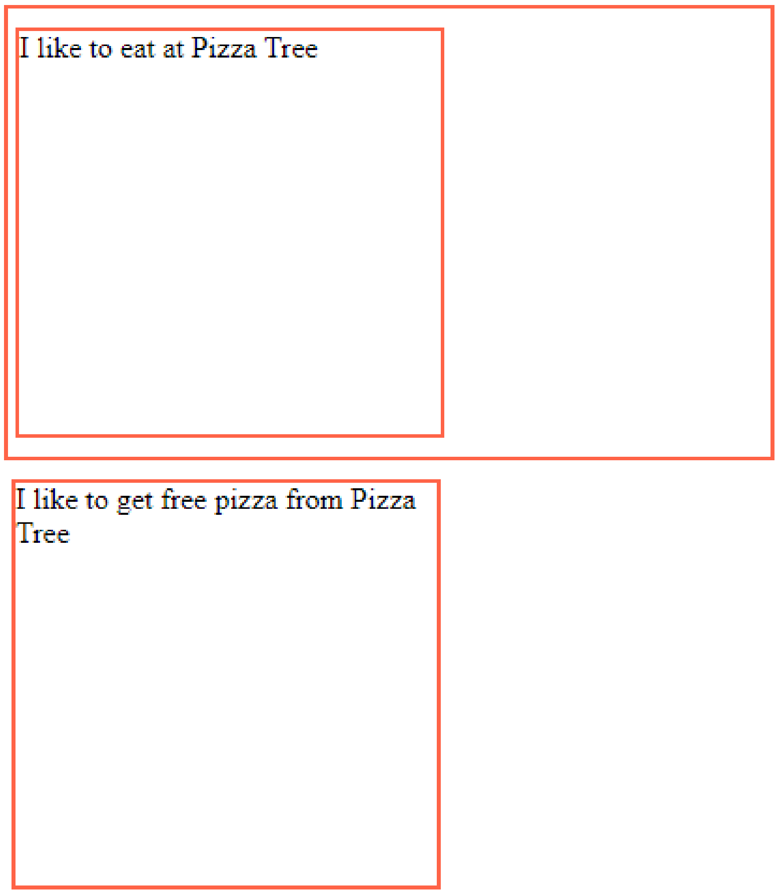
20. ☐ Save and preview in a browser. Still no changes, and your compiled CSS file looks the same, but you've created SCSS that contains easily re-usable styles. Directly under the previous `@mixin`, create another reusable style by creating an `@mixin` for the `div` elements that includes the same margin as used for paragraph elements:

```
@mixin divStyle {
    margin:$pM;
}
```

21. ☐ Below the new `@mixin` from Step 20 and above the paragraph selector, create a selector for any `div` element with an `id` of container that uses the mixins for both `divStyle` and `tomatoBorder`:

```
div#container {
  @include divStyle;
  @include tomatoBorder;
}
```

22. ☐ Save and preview in a browser. The first div element now has the border assigned to the paragraph `@mixin`, and you didn't have to duplicate styles.

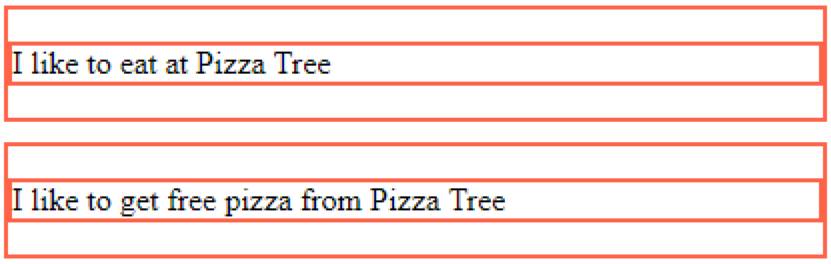

I like to eat at Pizza Tree

I like to get free pizza from Pizza Tree

23. ☐ Now let's nest paragraph element selectors within div element selectors to avoid having to write long, repetitive composite selectors. Remove the paragraph selector and replace the `div#container` selector you wrote in the last step with the following:

```
div#container {
    @include divStyle;
@include tomatoBorder ;
    p {
        @include tomatoBorder ;
    }
}
div#container2 {
    @include divStyle;
@include tomatoBorder ;
    p {
        @include tomatoBorder;
    }
}
```

24. ☐ Save and preview. Both div elements and both paragraph elements now share a common border:



I like to eat at Pizza Tree

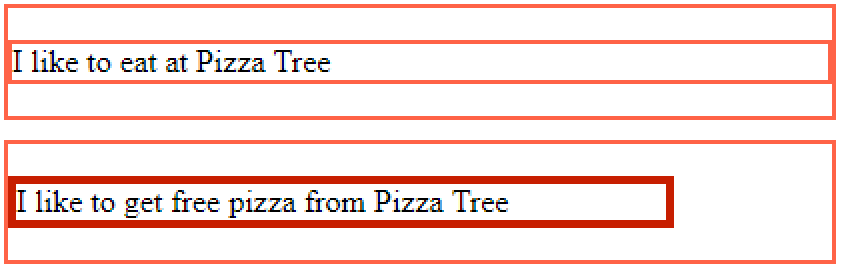
I like to get free pizza from Pizza Tree

25. ☐ Switch back to sassy.css in your IDE. SCSS has compiled the nested styles into duplicated composite styles, one each for `div#container p` and `div#container2 p`. Imagine how much more difficult it would be to manage a deeply nested HTML tree of thousands of elements with heavy CSS composite styles, when you can simply reuse what you need with SCSS and compile into CSS!

26. ☐ Integrate operators and functions into your SCSS by adding properties for width and border to the second div selector. Switch back to `sassy.scss` in your IDE. Add 100 pixels to the paragraph width variable. Use a SCSS function to darken the paragraph border color by 25% as follows (code to add in bold text):

```
div#container2 {
    @include divStyle;
    @include tomatoBorder;
    p {
        @include tomatoBorder;
        width: $pW + 100;
        border: 4px solid darken($pC, 25%);
    }
}
```

27. ☐ Save your work and preview. Your second paragraph should now be a darker shade of tomato and 335px wide as in the following screenshot:



I like to eat at Pizza Tree

I like to get free pizza from Pizza Tree

28. ☐ Examine the compiled CSS document for changes made by the last SCSS watch.

Congratulations! You have successfully installed Ruby and Sass, and created an SCSS document leveraging variables, `@mixins`, nested selectors, operators, and functions. You have also used Sass's watch command to compile SCSS to CSS.

STOP

This is the end of the exercise.

Overview

In this exercise, we will invoke Bootstrap classes to style an HTML document. Our starting point is a fresh, clean HTML document with the CSS Reset applied.

Screenshots taken for this exercise are in 1200px viewport width for desktop view, 420px viewport width for mobile view.

Part 1: Setting up Bootstrap with Basic Components

1. ☐ Open the following files in your IDE and Chrome:

```
C:\inetpub\wwwroot\pizzatree\Ex\Ex.7.2\bootstrap.html
```

2. ☐ In your IDE within `bootstrap.html` and directly under the CSS reset `<link>` element, add CDN references for Bootstrap 4.0.0 beta using the following CSS and JS resources (copy from the text file located in Exercise directory):

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
beta/css/bootstrap.min.css" integrity="sha384-
/Y6pD6FV/Vv2HJnA6t+vslU6fwYXjCFtcEpHbNJ0lyAFsXTsjBbfaDjzALeQsN6M"
crossorigin="anonymous">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
beta/js/bootstrap.min.js" integrity="sha384-
h0AbiXch4ZDo7tp9hKZ4TsHbi047NrKGLO3SEJAg45jXxnGIfYzk4Si90RDIqNm1"
crossorigin="anonymous"></script>
```

Bootstrap 4 added the `integrity` and `crossorigin` attributes to validate CDN source authenticity and work across CORS restrictions.

3. ☐ Bootstrap requires a master container class, which acts as the ancestor element to all descendant Bootstrap classes. Add a `<div>` element with a `class="container"` to the HTML document:

```
<div class="container">
</div>
```

All content must now be placed within this container element to leverage Bootstrap functionality.

Add another `<div>` element to act as our first Bootstrap row, and within this row add your first Bootstrap column:

```
    <div class="row">
        <div class="col">

        </div>
    </div>
```

4. ☐ Within the class value that includes `col` you created in the previous step, add the class `jumbotron`:

```
  <div class="col jumbotron">
```

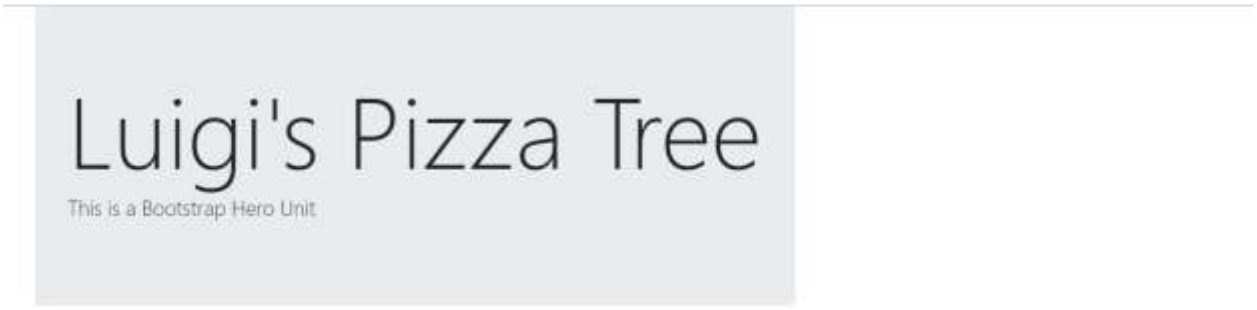
5. ☐ Add some content to the `<div>` element that is now a Bootstrap Jumbotron by including your name followed by the words Pizza Tree:

```
<div class="col jumbotron">
    Luigi's Pizza Tree
</div>
```

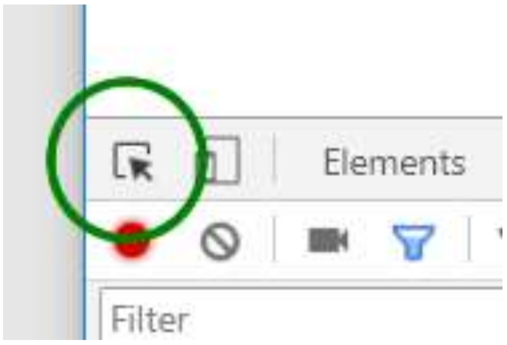
6. ☐ Add a `<p>` element within the Jumbotron, after the text you added in the previous step. Give your paragraph a `class="lead"` to make it stand out:

```
<p class="lead">This is a Bootstrap Hero Unit</p>
```

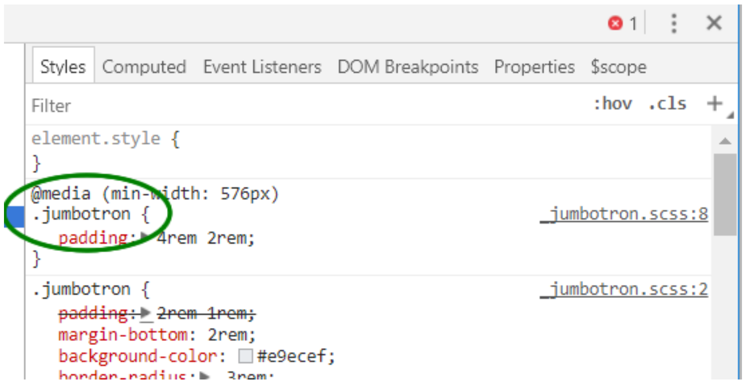
7. ☐ Save `bootstrap.html` and preview your work in Chrome:


Luigi's Pizza Tree
This is a Bootstrap Hero Unit

8. ☐ Still within Chrome, press `<F12>` to open the browser debugger so you can inspect the Bootstrap elements you just created. Press the Inspect Element button located on the top left of the debugger tools:



Move your cursor over the viewport and click any element to reveal the CSS applied in the browser debugger's Styles pane. In the following screenshot, the Jumbotron is being inspected:



Styles Computed Event Listeners DOM Breakpoints Properties $scope
Filter :hov .cls +

```
element.style {
}
@media (min-width: 576px)
.jumbotron {                                  _jumbotron.scss:8
   padding: ▶ 4rem 2rem;
}
.jumbotron {                                  _jumbotron.scss:2
   ~~padding: ▶ 2rem 1rem;~~
   margin-bottom: 2rem;
   background-color: ☐#e9ecef;
   border-radius: ▶ .3rem;
```

9. ☐ Currently, our Jumbotron is not a full-width element. Within the `element.style {}` Filter in the Styles tab, single left-click to add CSS until your Jumbotron is 100% wide:

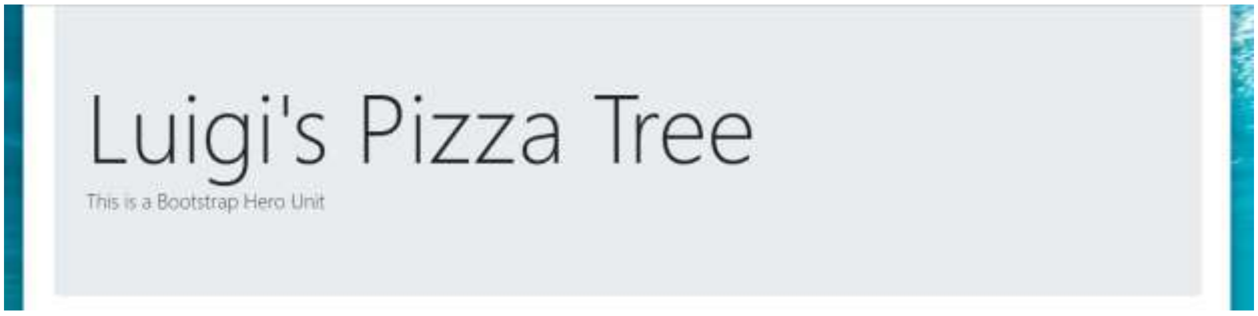
```
width:100%
```

10. ☐ Continue to inspect elements and adjust `element.style` Filters to see what CSS is added when Bootstrap classes are applied to HTML elements. Don't spend more than two or three minutes; we have more to do in this exercise!

11. ☐ Switch back to your IDE and add a `<style>` element around line 10 in `bootstrap.html` that corrects the full-width issue with our Jumbotron class:

```
<style>
.jumbotron {
    width:100%
}
</style>
```

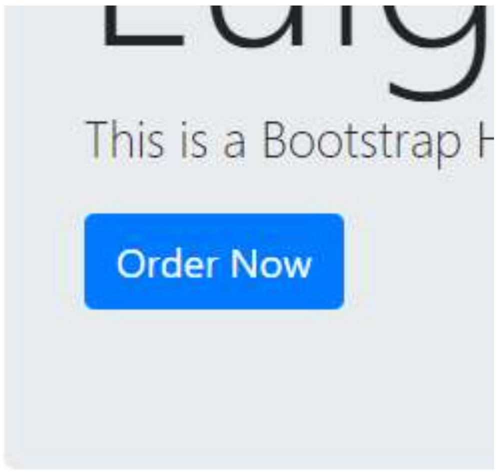
12. ☐ Save and preview in Chrome. You have just started customizing Bootstrap:



13. ☐ Add a button to order pizza directly under the Hero Paragraph and within your Jumbotron. Bootstrap will convert any `<a>` or `<button>` element when a `class="btn"` is applied. Add additional classes to change the button color, size, and other options. For example, to make the button blue, add `btn-primary` to your class:

```
<button type="button" class="btn btn-primary">Order Now</button>
```

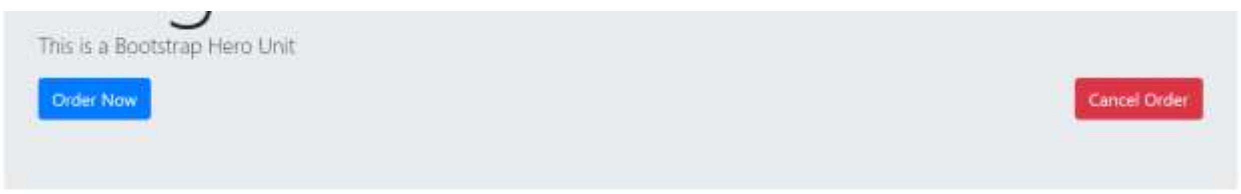
14. ☐ Save and preview your work. Hover over and click your button to see the different states applied to CSS:



15. ☐ Add another button directly after your first button to cancel orders, and make it both red with a btn-danger class and right-aligned with a float-right class:

```
<button type="button" class="btn btn-danger float-
right">Cancel Order</button>
```

16. ☐ Save and preview your work:



This is a Bootstrap Hero Unit

Part 2: Creating Responsive Multi-Column Layouts

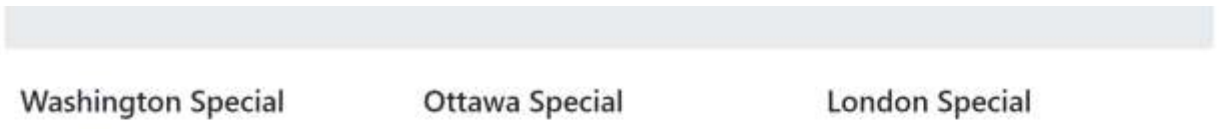
17. ☐ Switch back your IDE. We will now create a typical three-column layout for display deals by city, much like we did with our case study, but instead of using flexbox, we will use Bootstrap Columns.

Bootstrap rows span 12 columns by default. Bootstrap 4 now auto-assigns equal column widths when columns are created with the col class.

18. ☐ After the first row that contains our Jumbotron (make sure you're still in the root container class!), add a new Bootstrap row that contains three columns, each spanning 4 Bootstrap columns:

```
<div class="row">
    <div class="col">
        <h3>Washington Special</h3>
    </div>
    <div class="col">
        <h3>About Pizza Tree</h3>
    </div>
    <div class="col">
        <h3>London Special</h3>
    </div>
</div>
```

19. ☐ Save and preview your work:



20. ☐ It can be difficult to visualize the column sizes, so add some Bootstrap alert classes to each Bootstrap column. Set the first and last to alert alert-primary, set the middle column to alert alert-success:

```
<div class="row">
    <div class="col alert alert-primary">
        <h3>Washington Special</h3>
    </div>
    <div class="col alert alert-success">
        <h3>About Pizza Tree</h3>
    </div>
    <div class="col alert alert-primary">
        <h3>London Special</h3>
    </div>
</div>
```

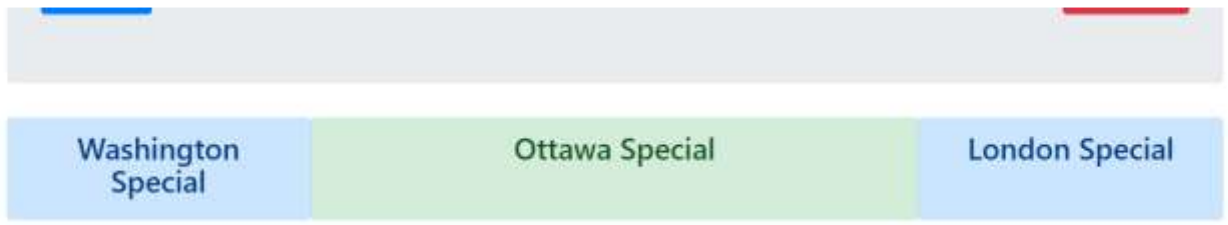
21. ☐ Center the text within each `<h3>` element by adding a `text-center` class:

```
<div class="row">
    <div class="col alert alert-primary">
        <h3 class="text-center">Washington Special</h3>
    </div>
    <div class="col alert alert-success">
        <h3 class="text-center">About Pizza Tree</h3>
    </div>
    <div class="col alert alert-primary">
        <h3 class="text-center">London Special</h3>
    </div>
</div>
```

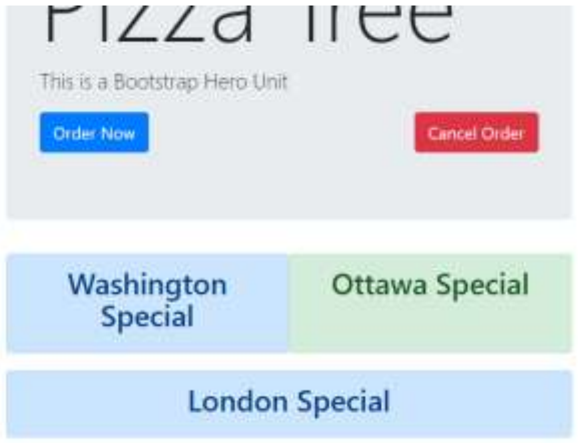
22. ☐ Expand the width of the middle Ottawa Special so it occupies 50% (1/2 of the available 12 cols) of the row width:

```
<div class="col col-6 alert alert-success">
    <h3 class="text-center">Ottawa Special</h3>
</div>
```

23. ☐ Save and preview your work:



So far so good; however, narrow your viewport and you'll see that while Bootstrap is indeed responsive, once your viewport resizes to <= 768px, we have an unanticipated effect that must be fixed:



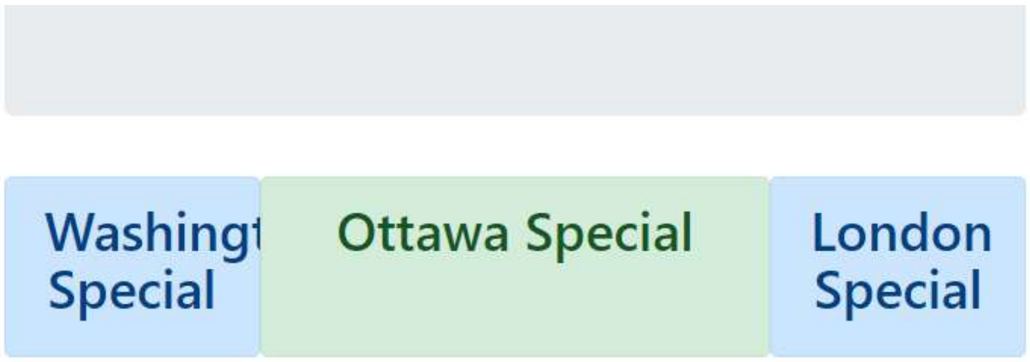
We have not explicitly assigned any column width to the first and last columns, and Bootstrap is wrapping the third column, which is now

100% width.

24. ☐ Add explicit column widths to the first and last columns, each set to `col-3`:

```
<div class="row">
   <div class="col col-3 alert alert-primary">
       <h3 class="text-center">Washington Special</h3>
   </div>
   <div class="col col-6 alert alert-success">
       <h3 class="text-center">Ottawa Special</h3>
   </div>
   <div class="col col-3 alert alert-primary">
       <h3 class="text-center">London Special</h3>
   </div>
</div>
```

25. ☐ Save and preview your work:



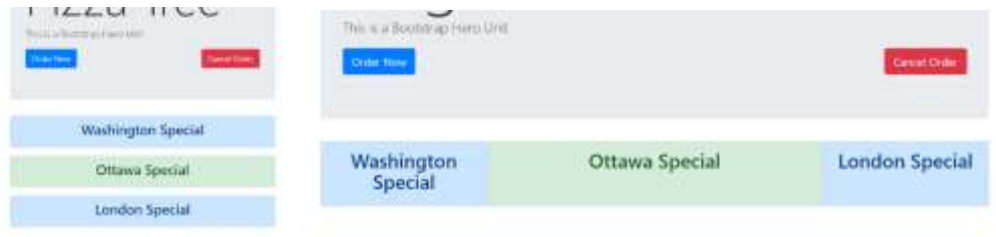
While our columns are now properly sized in smaller viewports, the padding applied to the root container class is causing loss of visible text in the Washington Specials at around 760px. It would be best to collapse this view at this width by adding viewport-specific column width classes to our three-column layout, based on the following Bootstrap Viewport definitions:

Extra Small	`xs`	viewport < 576px wide
Small	`sm`	viewport > 576px and <= 768px wide
Medium	`md`	viewport > 768px and <= 992px wide
Large	`lg`	viewport > 992px and <= 1200px wide
Extra Large	`xl`	viewport > 1200px wide

26. ☐ Multiple viewport settings can be applied to the same column by aggregating col classes and specifying the viewport width for each class. Adjust the column settings for each of our three columns to collapse all buttons to full-width when viewport width is less than medium:

```
<div class="row">
   <div class="col col-12 col-md-3 alert alert-primary">
       <h3 class="text-center">Washington Special</h3>
   </div>
   <div class="col col-12 col-md-6 alert alert-success">
       <h3 class="text-center">Ottawa Special</h3>
   </div>
   <div class="col col-12 col-md-3 alert alert-primary">
       <h3 class="text-center">London Special</h3>
   </div>
</div>
```

27. ☐ Save your work and preview your viewport in multiple widths:



Part 3: Interacting with the User

28. ☐ JavaScript alert, confirm, and prompt modal (blocking) functions are extremely basic and unappealing in modern UI. Bootstrap modals add a modern, polished look to any UI and require very little code to function. Bootstrap's JavaScript library will automatically handle the triggering and display of modals, hiding the underlying JavaScript mechanisms.

Switch back your IDE and within `bootstrap.html`, locate the first button we created (Order Now) near line 25.

Within this `<button>` element and after the class attributes, add the bootstrap attributes `data-toggle="modal"` (this identifies the button as a JavaScript trigger for a Bootstrap Modal Event) and data-target="#orderNow" (this identifies the button as a toggle for the orderNow class, which will be created shortly):

```
<button type="button" class="btn btn-primary"
data-toggle="modal" data-target="#orderNow">Order
Now</button>
```

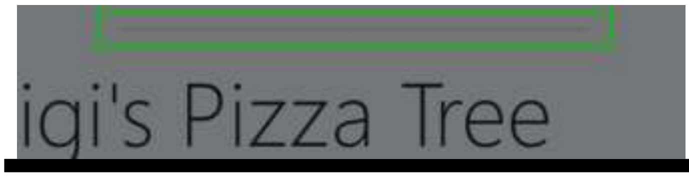
29. ☐ Create the Bootstrap Modal content to display when the data-toggle is clicked. Place this new `div` element OUTSIDE and UNDER the container div:

```
<div class="modal" id="orderNow">
    <div class="modal-dialog" >
        <div class="modal-content">
        </div>
   </div>
</div>
```

30. ☐ We need to include JavaScript references for jQuery and Popper before we reference the Bootstrap JavaScript library. At around line 8, after the CSS `<link>` but before the Bootstrap JavaScript reference, add the following CDN references:

```
<script src="https://code.jquery.com/jquery-
3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/G
pGFF93hXpG5KkN" crossorigin="anonymous"></script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper
.js/1.11.0/umd/popper.min.js" integrity="sha384-
b/U6ypiBEHpOf/4+1nzFpr53nxSS+GLCkfwBdFNTxtclqqenIS
fwAzpKaMNFNmj4" crossorigin="anonymous"></script>
```

31. ☐ Save your work and preview. Clicking the Order Now button will instantly apply a transparent grey overlay on top of the viewport, but there is no content yet:



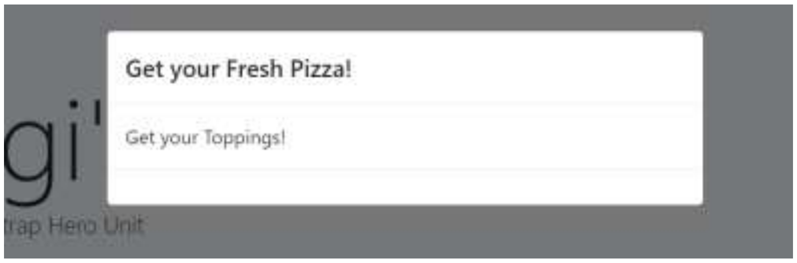
32. ☐ Bootstrap Modal content is comprised of a Header, Content and Footer class trio. Switch back to your IDE and within bootstrap.html, add the following modal-header class within the modal-content class:

```
<div class="modal-content">
    <div class="modal-header">
      <h5 class="modal-title">Get your Fresh Pizza!</h5>
    </div>
</div>
```

33. ☐ After the new modal-header class, add a content and footer class:

```
<div class="modal-header">
  <h5 class="modal-title">Get your Fresh Pizza!</h5>
</div>
<div class="modal-body">
  Get your Toppings!
</div>
<div class="modal-footer">

</div>
```

34. ☐ Save and preview in Chrome:



Get your Fresh Pizza!

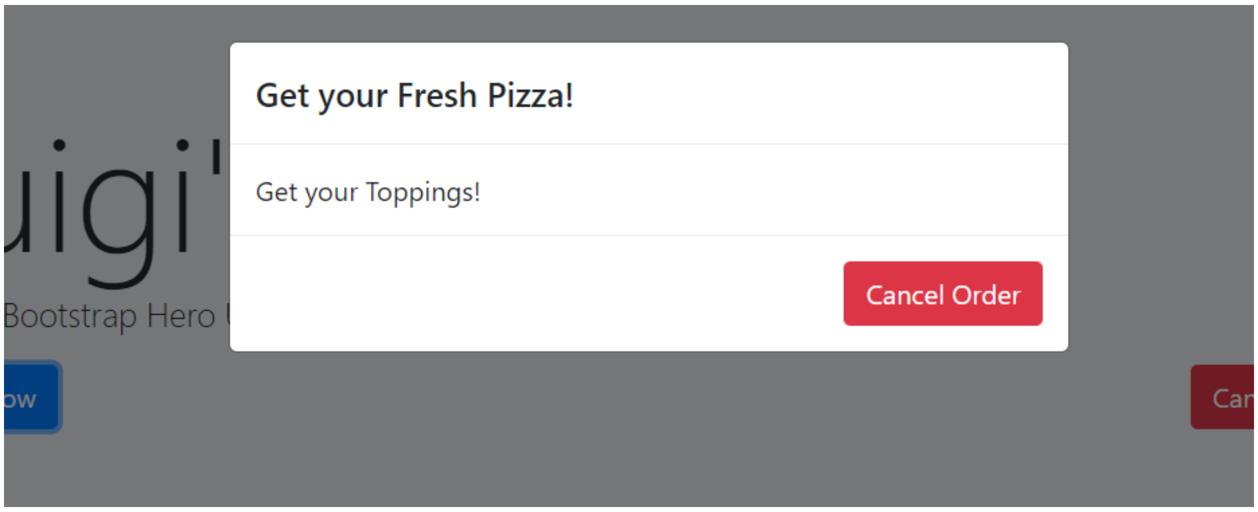
Get your Toppings!

The modal dialog works, but it cannot yet be dismissed. We can continue to apply this JavaScript functionality with Bootstrap classes in the next step.

35. ☐ Switch back to your IDE and within bootstrap.html, add a button to the modal-footer class that closes the modal class by adding a Bootstrap data-dismiss=”modal” attribute/value pair:

```
<div class="modal-body">
  Get your Toppings!
</div>
<div class="modal-footer">
  <button type="button" class="btn btn-danger" data-
dismiss="modal">Cancel Order</button>
</div>
```

36. ☐ Save and preview your fully functional modal dialog:



37. ☐ In an earlier exercise, we used a Ruby Gem to install Sass. Sass and Bootstrap work extremely well together, and using Sass with Bootstrap provides developers with additional control over their Bootstrap distributions.

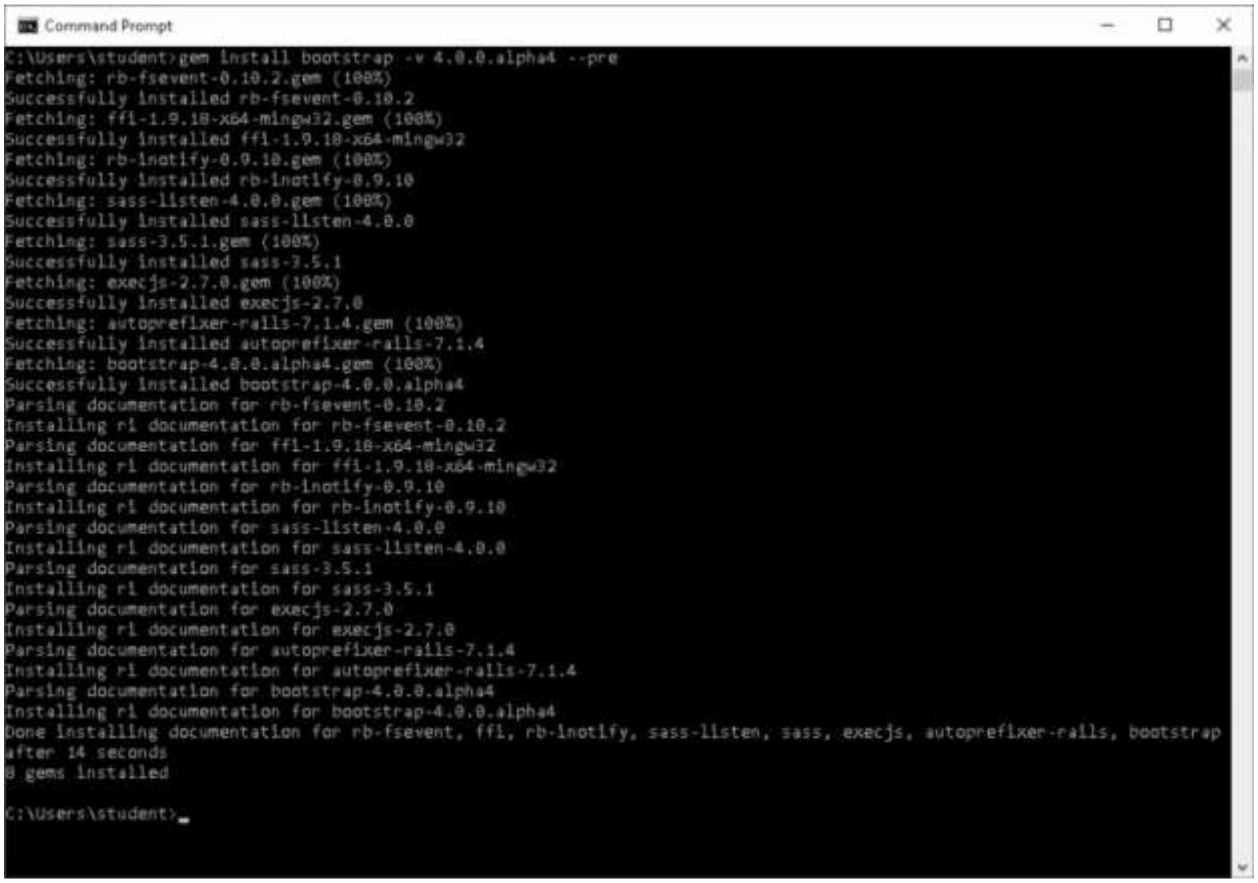
Let’s install the Bootstrap Gem so we can use Bootstrap Sass throughout our web application.

Open a command prompt by pressing the magnifying glass button on the bottom left of your taskbar, typing **`cmd:`** and pressing `<Enter>`.

At the command prompt, type the following:

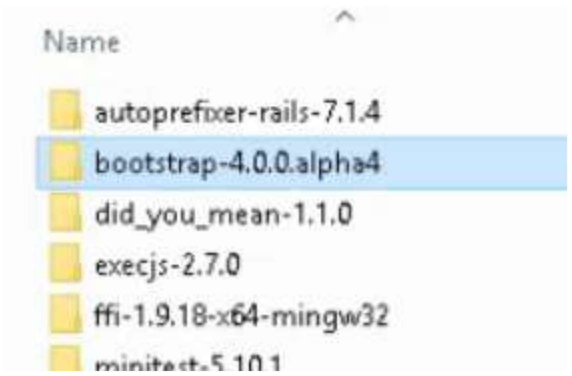
```
gem install bootstrap -v 4.0.0.alpha4 --pre
```

You should see the following response (if you did not complete Ex 7.1, this step has also installed Sass, and your screenshot will be nearly identical. If you did complete Ex 7.1, your screenshot will be slightly different as your command prompt will contain fewer installation log entries with Sass already installed):



```
C:\Users\student>gem install bootstrap -v 4.0.0.alpha4 --pre
Fetching: rb-fsevent-0.10.2.gem (100%)
Successfully installed rb-fsevent-0.10.2
Fetching: ffi-1.9.18-x64-mingw32.gem (100%)
Successfully installed ffi-1.9.18-x64-mingw32
Fetching: rb-inotify-0.9.10.gem (100%)
Successfully installed rb-inotify-0.9.10
Fetching: sass-listen-4.0.0.gem (100%)
Successfully installed sass-listen-4.0.0
Fetching: sass-3.5.1.gem (100%)
Successfully installed sass-3.5.1
Fetching: execjs-2.7.0.gem (100%)
Successfully installed execjs-2.7.0
Fetching: autoprefixer-rails-7.1.4.gem (100%)
Successfully installed autoprefixer-rails-7.1.4
Fetching: bootstrap-4.0.0.alpha4.gem (100%)
Successfully installed bootstrap-4.0.0.alpha4
Parsing documentation for rb-fsevent-0.10.2
Installing ri documentation for rb-fsevent-0.10.2
Parsing documentation for ffi-1.9.18-x64-mingw32
Installing ri documentation for ffi-1.9.18-x64-mingw32
Parsing documentation for rb-inotify-0.9.10
Installing ri documentation for rb-inotify-0.9.10
Parsing documentation for sass-listen-4.0.0
Installing ri documentation for sass-listen-4.0.0
Parsing documentation for sass-3.5.1
Installing ri documentation for sass-3.5.1
Parsing documentation for execjs-2.7.0
Installing ri documentation for execjs-2.7.0
Parsing documentation for autoprefixer-rails-7.1.4
Installing ri documentation for autoprefixer-rails-7.1.4
Parsing documentation for bootstrap-4.0.0.alpha4
Installing ri documentation for bootstrap-4.0.0.alpha4
Done installing documentation for rb-fsevent, ffi, rb-inotify, sass-listen, sass, execjs, autoprefixer-rails, bootstrap
after 14 seconds
8 gems installed

C:\Users\student>_
```

Within your GEM folder in `C:/Ruby24-x64`, you now have a full local bootstrap Sass distribution:



Congratulations! Manipulating Bootstrap via Sass and Ruby is not only an entire Exercise on its own, but is best left to an environment dedicated exclusively to Bootstrap. You are already well on your way to creating a custom Bootstrap template, but you have added the Bootstrap Gem to your Ruby Install, which allows you to continue your Sass development by customizing the base Bootstrap install with Sass.

STOP

This is the end of the exercise.

Overview

In this final exercise, we will convert our CSS menu to an accessible and responsive, collapsing menu without the requirement for a third-party JavaScript library such as jQuery.

Screenshots taken for this Exercise are in 1200px viewport width for desktop view, 420px viewport width for mobile view.

1. ☐ Open the following file in your IDE and Chrome:

   `C:\inetpub\wwwroot\pizzatree\Ex\Ex.8.1\index.html`

2. ☐ In your IDE, goto the bottom of `index.html` and add an empty `<script>` element just before the closing `<body>` element near line 100 (leave a blank line between tags):

   ```
   <script>

   </script>
   ```

3. ☐ Within the `<script>` body, use JavaScript to retrieve an element with an id attribute matching `menuToggle`. Our hamburger icon has the attribute `id="menuToggle"`.

   ```
   document.getElementById('menuToggle')
   ```

4. ☐ Bind a click event listener to the menuToggle:

```
document.getElementById('menuToggle').addEventListener('click',
```

5. ☐ When JavaScript hears the click on the menuToggle, invoke a function that accepts a parameter called `e` (short for event):

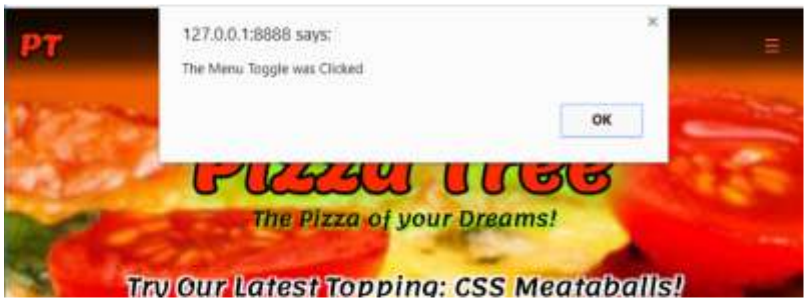
```
.addEventListener('click', function(e) {    }, false);
```

Is the `,false)` required? Yes. This tells our event binding mechanism not to use event propagation, a topic of discussion best suited to a dedicated JavaScript environment.

6. ☐ Within the body of the function, add a modal alert that confirms operation of the toggle:

```
<script>
document.getElementById('menuToggle').addEventListener('click',
function(e) {
    alert('The Menu Toggle was Clicked')
}, false);
</script>
```

7. ☐ Save `index.html` and preview in Chrome. Click the menu toggle and you will see your alert:


127.0.0.1:8888 says:

The Menu Toggle was Clicked

OK
PT
Pizza Tree
The Pizza of your Dreams!
Try Our Latest Topping: CSS Meataballs!

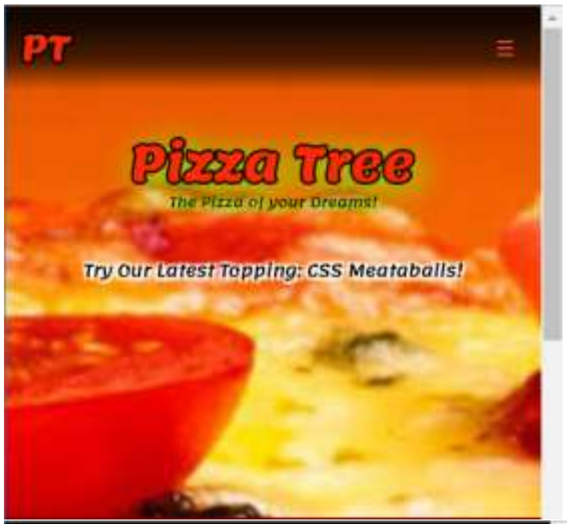
8. ☐ It’s time to display our menu instead of our modal message when the toggle is clicked.

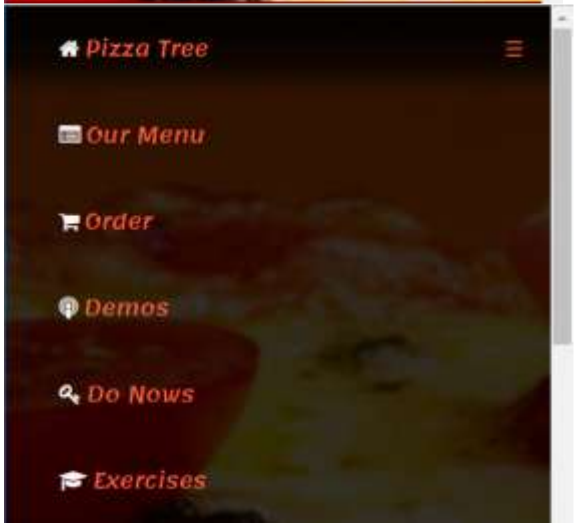
   First, we must detect the current state of menu display, which is based on the presence of the collapse class. We then set the collapse class to the opposite of the current display state, and modify the display property of our menu according to the status of the collapse class.

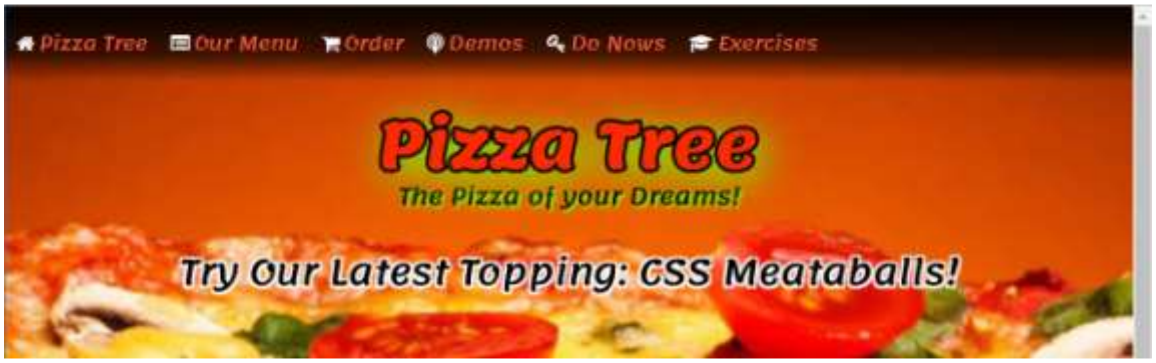
   Add the following JavaScript logic within the body of your function:

```
document.getElementById('menuToggle').addEventListener('click',
function(e) {
    if (e.target.parentNode.parentNode.className == 'menuToggle
collapse') {
        e.target.parentNode.parentNode.className = 'menuToggle';
        document.querySelector('div#feature header').style.display
= 'block'
    } else {
        e.target.parentNode.parentNode.className = 'menuToggle
collapse';
        document.querySelector('div#feature header').style.display
= 'none'
    }
}, false);
```

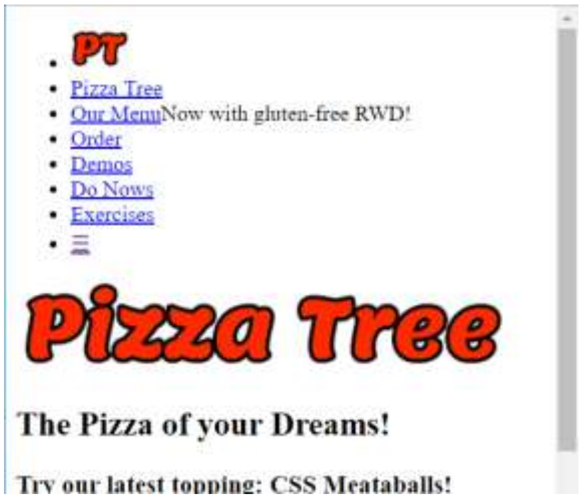
9. ☐ Save your work and preview in Chrome. Toggle the menu and confirm it is responsive by adjusting viewport widths:


PT
☰
Pizza Tree
The Pizza of your Dreams!
Try Our Latest Topping: CSS Meataballs!




Pizza Tree
Our Menu
Order
Demos
Do Nows
Exercises
Pizza Tree
The Pizza of your Dreams!
Try Our Latest Topping: CSS Meataballs!

10. ☐ Confirm your menu is accessible by removing the CSS using the Web Developer Extension:



- PT
- Pizza Tree
- Our MenuNow with gluten-free RWD!
- Order
- Demos
- Do Nows
- Exercises
- ☰

Pizza Tree

The Pizza of your Dreams!

Try our latest topping: CSS Meatballs!

The “Now with gluten-free RWD” and hamburger icon can be removed from the linear layout and accessibly added to the rendered content easily with jQuery. Trying to do so with POJO is an exercise in frustration!

Congratulations! You have completed converting the case study to an accessible and responsive website.

STOP

This is the end of the exercise.