Breaking down sites

Let's break down this site:

This site has a lot to break down but it is still in a nice, clean (uncrowded) design. We'll start by applying our basic box breakdown. Here is what I see:

- There is a background image behind most of this page
- That background image has had a black overlay applied to dim it
    - The image may have been made this way, or some HTML and CSS could be doing this effect
- There is a nav bar with the links set to the right
- There is a site title or small logo in the upper left corner
- The main text is centered in the middle of the screen
- There is a box with 3 content areas near the bottom of the screen
- Those 3 content areas are spaced evenly across the space, and their content is left-justified

That's the basic structure I see, and you may see more or less detail that what I've broken down. This breakdown is the easy part, now comes the hard part - figuring out what tags and CSS properties we should use to achieve this layout.

Here is the skeletal code for this project:

Class 11 Example.zip

Download this file, then in Workshop click **Choose File** button and select this zip file, then click **Upload**. After the upload completes you should see a new project folder **Class 11 Example**. Expand this project folder.

Let's now consider the background image:

- Download this image
- In Class 11 Example folder create the folder **images**, and upload the image there. Rename this image to **background.jpg**.
- Select the **index.html** file.

Notice The <link> tag in index.html for the fontawsome css. This is for **fontawesome**, which is a CSS file from the fontawesome site that provides the **icons** we'll use for those in the upper right corner of the screenshot, and the boxes near the bottom. These icons are not images, but **glyphs** in a font. More on our use of this to come.

- In **index.html**, let's add a tag for this home page. As in our sample project, we'll use an article tag for this purpose. Give the article tag an id of **home**.
- In **standard.css**, add a selector block for **#home**. In this block add the CSS to make this image full size:

```
#home {
background-image: url('../images/background.jpg');
background-size: cover;
background-repeat: no-repeat;
width: 100%;
height: 100vh;
}
```

The width and height will cause the #home article to exactly fill the screen, which in turn causes the

background image with it's size of cover to completely cover the screen as well. Preview your index.html file, and depending on your screen resolution and aspect ratio you may notice that the right side of the background image is somewhat cut off. This is because we have not yet set the position we want the background-image to occupy. Our CSS has the background-size: cover so that the image will cover the entire space available, but by default it will be aligned at the top and left edge. Look at our background image - where is content we absolutely do not want to be cropped due to the cover size? The man at the top of the image is already partially cropped in the photo, and so aligning this photo to the top of the space is the correct decision in order to not further crop that top area. The rest of the original photo is well centered horizontally, so it is best to center the position of this background image in the center horizontally as well. To do this we need to add to #home selector block:

```
background-position: center top;
```

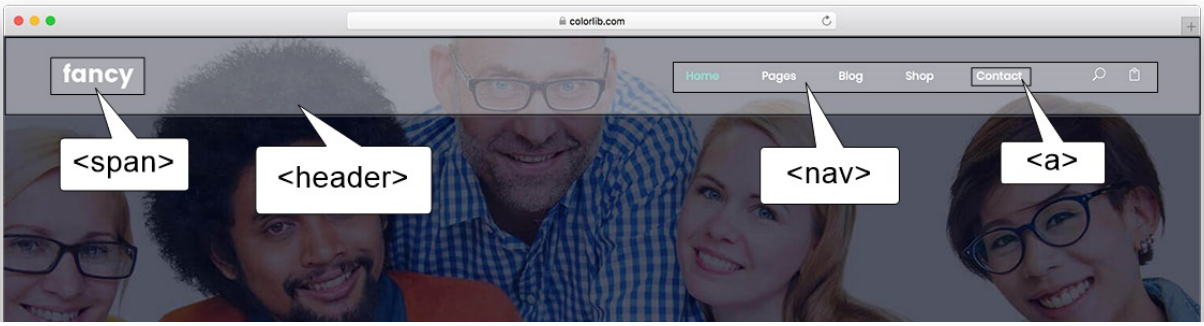
The background-position has two values the first is the position horizontally and the second is vertically.

Header & Nav

At the top of the screen we have the site nav, so not surprisingly we'll be using a <nav> tag for what we see here. But consider everything we see there. There are really three separate groupings within this top area: The **fancy** to the left, the main **links** near the right side, and separated from those links are the two **icons**. What HTML might comprise this layout?

As with most things in development, there are many different ways to implement this. We'll break this down into three tags:

- A <header> tag to contain everything
- A tag inside the header to contain **fancy**
- A <nav> tag inside the header to contain the links and the icons, each of which is inside an <a> tag



So, above our <article> tag we should now add the basic HTML for this top area:

```
<header>
<span>fancy</span>
<nav>
<a href="#">Home</a>
<a href="#">Pages</a>
<a href="#">Blog</a>
<a href="#">Shop</a>
```

```
<a href="#">Contact</a>
<a href="#"><i class="fas fa-search"></i></a>
<a href="#"><i class="far fa-clipboard"></i></a>
</nav>
</header>
```

The last two <a> tags are our icons. They were located by doing:

- Go to fontawesome.com and click **Icons**

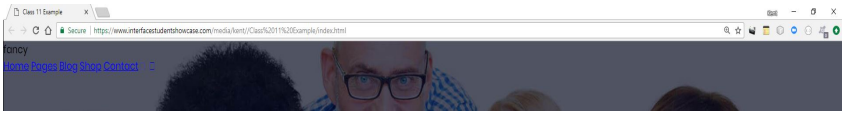
- Search for **search**
- Click on the first icon returned
- Near the bottom of the screen you should see an HTML box with: ****
- Click the clipboard icon next to this html to copy it to the clipboard
- Paste this as the content where you want it to go (already done in the above HTML)
- Go back to the search screen and search for **clipboard**
- Click the second icon returned
- The displayed HTML should be ****
- Copy it's HTML and paste it where you want it to go (already done in the above HTML)

Now for the CSS for this HTML. We'll start by providing some default font information for the entire page:

```
html, body {
margin: 0;
font-family: Poppins, sans-serif;
font-size: 2vh;
}
```

Where do we want the header? Fixed At the top of the screen, so we'll use position: fixed for this:

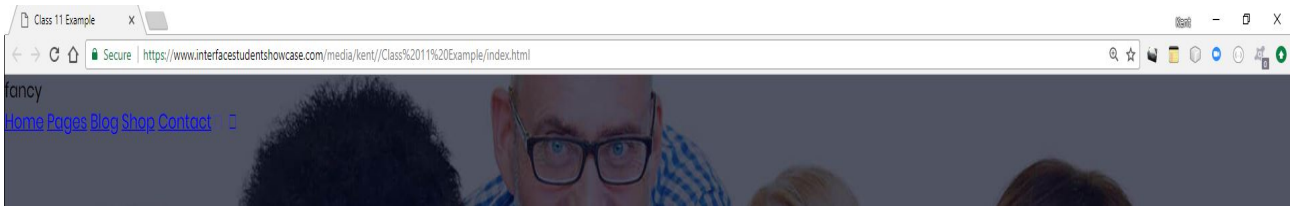
```
header {
position: fixed;
top: 0;
left: 0;
width: 100%;
}
```



Preview your index.html file so far.

How are we going to get the fancy to the left and the other links to the right?

Header layout



How do we get fancy to be on the left and the links to be on the right?

float fancy to the left and nav to the right

Set the header to be a flexbox and then use justify-content: space-between

Header breakdown

We'll go with the flexbox approach:

To the header selector block in standard.css add:

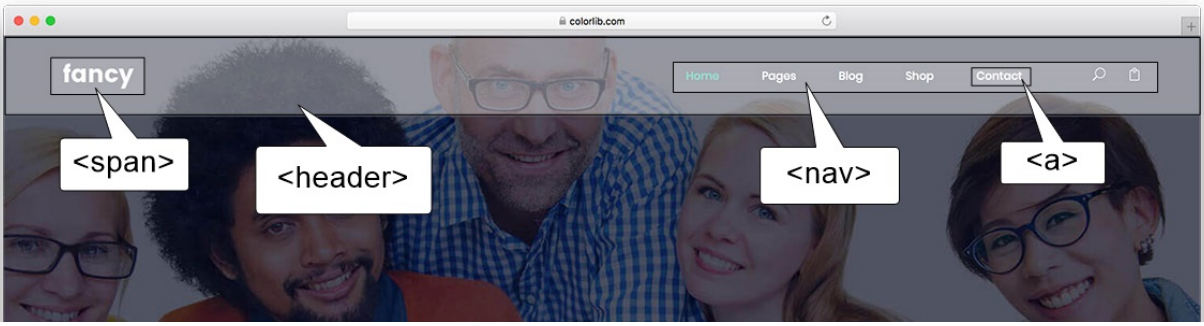
```
display: flex;
justify-content: space-between;
```

This will get us closer, but we you'll notice when you preview that fancy is right up against the left side of the screen, and the rightmost icon is up against the right side of the screen. We need to add some empty space - What choices do we have? To separate content like this our two basic options are margin and padding. We'll choose margin by adding some new selector blocks:

```
header > span {
margin-left: 3vw;
}
header > nav > a:last-child {
margin-right: 3vw;
}
```

Instead of using the compound child selector > we could also have added either an id or a class attribute to the appropriate tags and selected those here instead. Either approach would be fine. Where did the **3vw** values come from? Simple dead-reckoning: Look at the space as a relative proportion of the page in the direction you are trying to space, which is horizontally (width) in the case. I judged the spacing to be about 3% of the screen width. Feel free to adjust this based on your own perception.

Last, we need to add a little separation between the last text link and the first icon link. Remember this is what we're going for:



Again, many choices of how to accomplish, for example we could add a margin-right to Contact or a margin-left on the search icon. We'll do the latter:

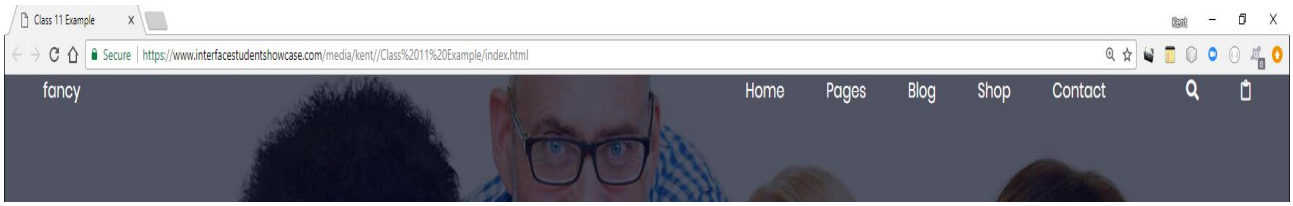
```
header > nav > a:nth-last-child(2) {
margin-left: 3vw;
}
```

We also need to make all of this text and icons white. Since most of the text on this page is white, we'll just add **color: white;** add the to the html, body selector block. This will color everything white except the links, since the browser will color those blue by default and the white will not be inherited. So we need to add a selector block for this purpose:

```
nav > a {
color: white;
```

```
text-decoration: none;
margin-right: 3vw;
}
```

The margin-right was added to separate the links and roughly match the sample page. Your header should now look like this:



What we are trying to achieve is this:

We're getting close, but the biggest mismatch right now is the vertical spacing. Our version (top) is too close to the top of the screen, we need to move it down some. How much? Imagine the row of our link text. As a proportion of the size of that text, how much empty space is above it? I'd guess about 2 rows of text, which is 2em. But how can we use this to space this text down? You should know what I'm about to say on this topic - there are many ways to do this, and I'll choose one we haven't seen so far: using line-height and vertical-align: middle.

Examine the screenshot from the sample (bottom) and picture that this part of the screen is the header. What we want to accomplish is centering all this text vertically within that space. vertical-align: middle would do this if the line-height for the text matched the height of the area as shown. Since we're currently guessing that the space above the links is 2em, and the height of the text is 1em, that means the total height is 5em. So let's set the line-height of all the a tags to be 5em. In the nav > a selector block add:

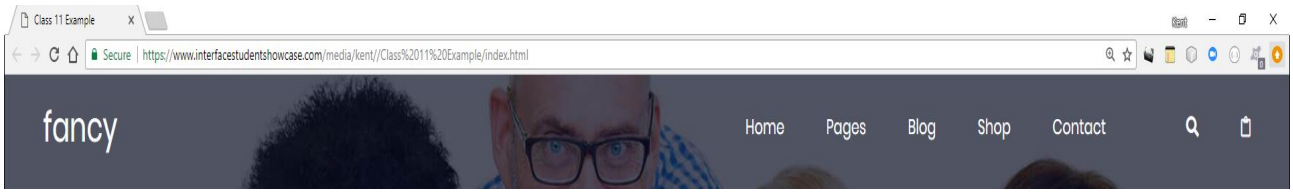
```
line-height: 5em;
vertical-align: middle;
```

We'll want to do something similar for the **span** that surrounds the text **fancy.** So to the **header > span** selector block add:

```
 font-size: 2em;
line-height: 5rem;
vertical-align: middle;
```

This CSS also doubles the size of this font since the text should be bigger, and uses rem instead of em for the line-height. Our nav > a line-height could also be 5rem instead of 5em, and since it's font height is the default inherited from **html, body** it would be the same height. But in the header > span properties we have increased the font size to 2em, and so using line-height of 5em would really mean 10 (5 x 2) of the base font size!

If you've done everything correctly, the top of your page should look like this:



As always, if yours looks a lot different than this, double check all your work.

The Middle

Here is our sample page we are implementing:

Let's tackle the middle of this page next. Like our TwinFeats Software sample project, we already have an <article> tag to contain the home page content. We'll use a <main> tag inside this article for the middle content of the page:

```
<article id="home">
<main>
<h1>
Website Design, Brand Strategy,
<br/>
Digital Marketing with Stunning Results
</h1>
<a href="#">About us
</a>
<a href="#">Get a quote
</a>
</main>
</article>
```

Inside of this main tag is an <h1> tag to surround the text we want to make into large heading text, and that text is manually formatted using a
 tag to separate the two lines. This is done so that we have control over what text is on which line, although this approach also needs a bit of CSS support as we'll soon see. Remember that text content in our HTML is just inline content, and will be wrapped at the browsers discretion based on available space. This is not the effect we want, so

the combination of
 and the CSS described below will force the text into the format we want it to be in. Lastly, we have the two <a> tags for the links that will be styled using CSS to look like the buttons in the screenshot.

The primary text is about in the middle of the page both horizontally and vertically, and we've learned some techniques to accomplish this. We'll use positioning:

```
main {
font-size: 2em;
position: absolute;
left: 50%;
top: 50%;
transform: translate(-50%, -40%);
white-space: nowrap;
text-align: center;
}
```

This CSS:

- Doubles the h1 font size
- Positions the content in the middle of the page
- Prevents the text from auto-wrapping
- Aligns the text in the center horizontally

The **white-space: nowrap** is how we prevent text from wrapping to a new line between words. Normally, inline content like simple text automatically is wrapped by the browser based on space available and CSS, but this line of CSS prevents this default behavior.

Below this big text in the middle are two buttons. We've already placed <a> tags for these, and with the right CSS we can achieve the desired look:

```
main a {
font-size: 1rem;
text-transform: uppercase;
padding-top: 0.5em;
padding-bottom: 0.5em;
width: 20ch;
border: 3px solid transparent;
border-image: linear-gradient(90deg, blue, lightblue, turquoise);
border-image-slice: 30%;
display: inline-block;
color: inherit;
text-shadow: 1px 1px #303030;
text-decoration: none;
}
main a:hover {
border-width:0;
padding-top: calc(.5em + 3px);
padding-bottom: calc(.5em + 3px);
width: calc(20ch + 5px);
background: linear-gradient(90deg, blue, lightblue, turquoise);
}
```

CSS gradients are discussed at w3schools in a previous homework. But let's dig into the rest of this CSS.

There are some new CSS properties here, like text-shadow and text-transform. The most interesting part of this CSS is how to match the border, with its linear-gradient, with the background's linear-gradient. This is pretty tricky to accomplish due to the details behind linear-gradient, so what the above CSS does is simply avoid it. The non-hover just has a border, and the hover turns off the border with **border-width: 0** and then increases the top and bottom padding and the width using **calc** to include the size of the old border (3px). This way the entire size of these <a> tags (content size + padding + border) does not change.

Your page should now look like this:

The middle is now too high - the links we just added increased the total height of <main>, and so now centering it vertically has pushed the text higher than we want. We really want the <h1> text to be in the middle. How can we do this?

The centering of <main> is being done in part with this line of CSS in the main selector block:

```
transform: translate(-50%, -50%);
```

We want to change the second 50% (which is the Y translate) to account for the extra line of text we have added for the links. We have about two lines of text to account for (the margin below the <h1> and the height of the styled links), so we change the transform to:

```
transform: translate(-50%, calc(-50% + 2em));
```

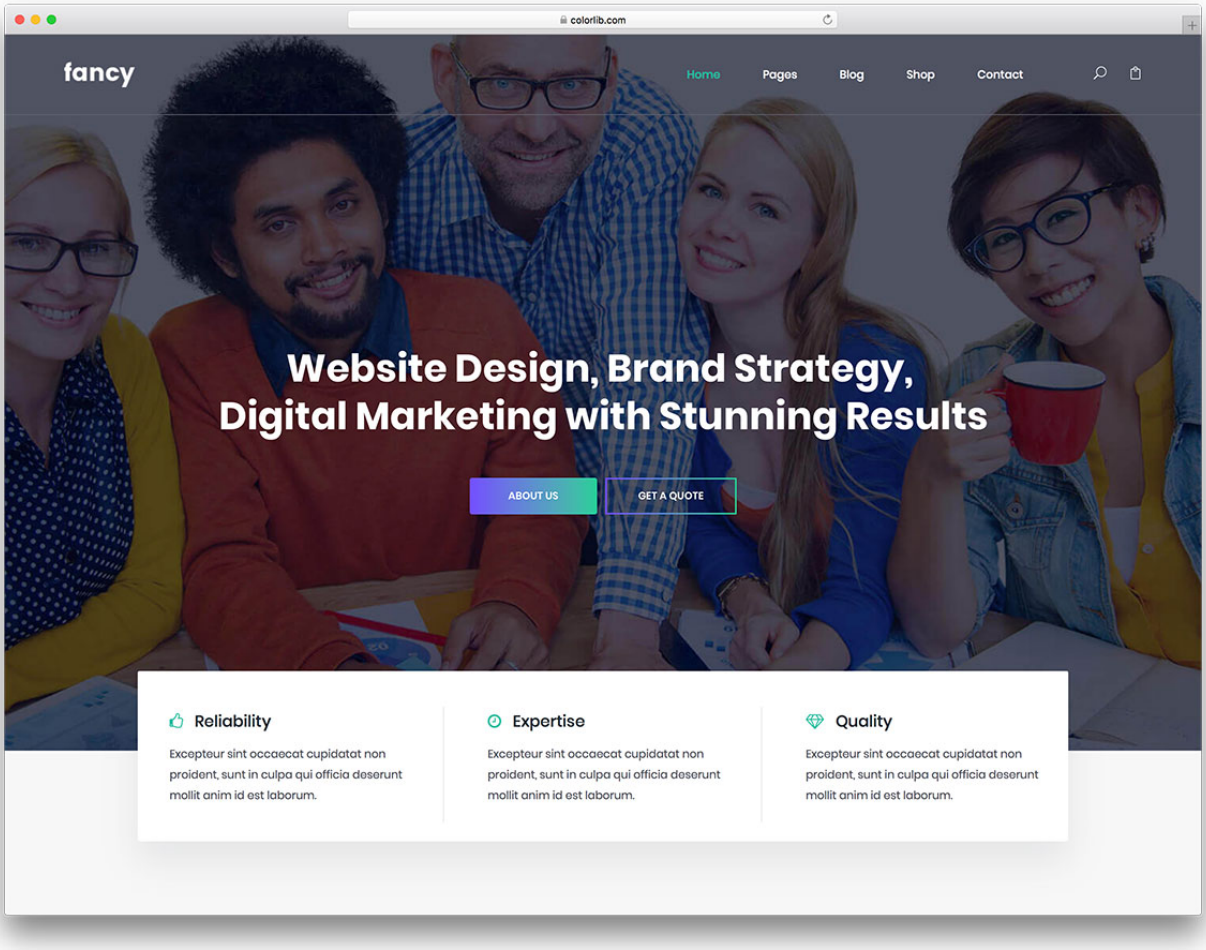
We end up with:

Excellent, this is very close to our target sample so far! Now we have to tackle the big white boxes at the bottom.

Accent box

```
.teaser > div:last-child
```

Once again, here is our target sample page:



This white box straddles the bottom of the page. The effect here acts as a teaser - it lets the user know that there is more content, and give you a little peek at what that content is without showing you it all. So question #1: How do we get this big white box to do this?

Our rule is if something seems significantly out of place from where the browser would naturally put it, it is probably positioned. First the HTML, to be placed right after the <main> block:

```
<div class="teaser">
<div>
<h2>
<i class="far fa-thumbs-up">
</i> Reliability
</h2>
Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
</div>
<div>
<h2>
<i class="far fa-clock">
</i> Expertise
</h2>
```

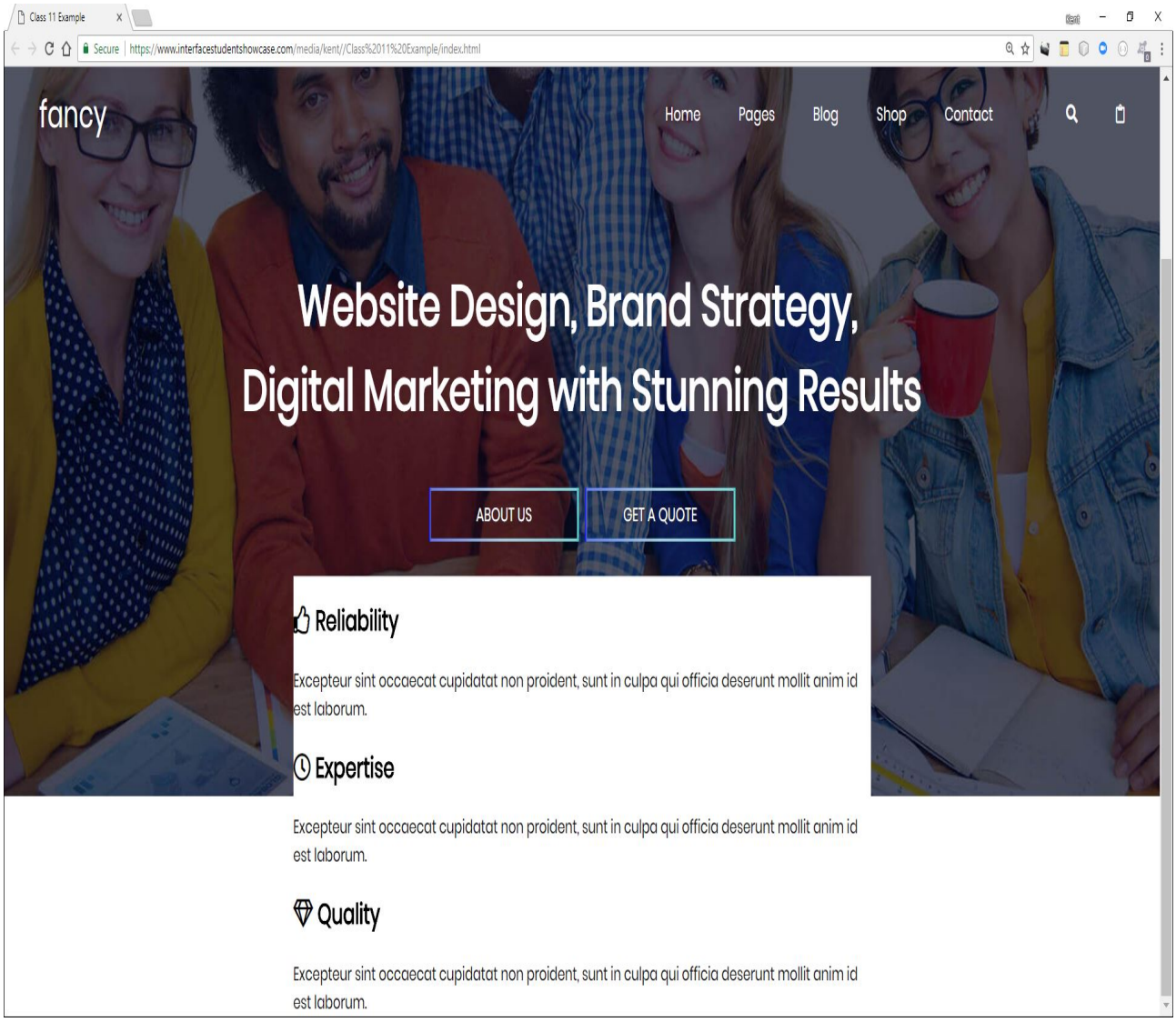
```
Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
</div>
<div>
<h2>
<i class="far fa-gem">
</i> Quality
</h2>
Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
</div>
</div>
```

Note the use of a class instead of an id for this outer div. We could use an id, but if we imagine implementing the other pages on this site and not just the home page, perhaps we'll put a teaser like this between every pair of pages, so I've chosen a class here. We'll place this with a technique very similar to our position method for centering content, except we aren't centering this content vertically. Remember how positioned centering works: Place the content at 50% (or 50vw/vh depending) and then use a transform to move it backwards by half of it's width/height. We want to do the same thing, but instead of starting at 50vh we start at 100vh:

```
.teaser {
position: absolute;
top: 100vh;
left: 50%;
transform: translate(-50%, -50%);
background-color: white;
color: black;
}
```

Make sure you think this positioning through, how similar it is to centering, and how it works.

Your page, scrolled down to the bottom, should look a lot like this:



What happened, why is this vertical and not horizontal?

Our HTML is currently using tags for each of the three content areas, and div is a block level tag, that's why. We want to lay these out across the page. Many ways to do this: We could float each of them (not a good use for float), we could turn the divs into inline-blocks so they lay out as inline content across the page, or we could use a flexbox. We'll use flexbox, which means our .teaser div needs to be the flex container in order to layout its direct children (the other divs) as flex items. Just add **display: flex;** to the list of .teaser properties.

Closer! If you compare this to our sample, our teaser box is not nearly wide enough. We need to set a width for this box: just add **width: 80%;** to the .teaser CSS properties.

We also need to add some vertical line separators and extra space between these three divs. The lines are just border-right on the first two divs, and none on the last one:

```
.teaser > div {
padding-left: 3em;
padding-right: 3em;
border-right: 1px solid #404040;
}
.teaser > div:last-child {
border-right: 0;
}
```

Preview this and you'll see:

Hmmmm - the lines go all the way from top to bottom of the .teaser div, but in our sample they don't. This is surprisingly interesting problem to solve, and there are many, many ways of doing it. We'll learn something new to do this.

In CSS, we've seen pseudo-classes like :hover and :nth-child. There are also the pseudo-elements **::after** and **::before**. These place text at the start or end of the matched elements, and then apply CSS properties to that text, treating the text as if it was in a . By generating content inside of each of the three divs in this way, we can then apply CSS so that the border-right is not on the in our HTML, but in the generating pseudo-element. This is a very strange concept, so here is what we are going to do in picture form:

We'll use CSS to make the generated ::after content to be shorter than the div that that contains it, so that the border-right applied to the generated ::after content doesn't take up all the vertical space. The CSS looks like this:

```
.teaser > div::after {
border-right: 1px solid #404040;
content: "";
position: absolute;
width: 100%;
height: 70%;
left: 0;
top: 15%;
}
.teaser > div:last-child::after {
border-right: 0;
}
```

We also need to remove the **border-right** on **.teaser > div** that we added previously, since we're doing that in the :: after now. Note that the previous

```
.teaser > div:last-child {
```

Is now

```
.teaser > div:last-child::after {
```

Lastly, to the **.teaser > div** selector block we need to add **position: relative;** so that this div tag becomes the position parent for the :: after content, which is being positioned absolutely. **THIS IS CRITICAL** for this technique, so be sure you understand this: When you use **position: absolute** or **position: fixed**, the top/left/bottom/right CSS property values are relative to the nearest ancestor tag that is being positioned any way other than static (the default). This is why the parent tag for the ::after content, which is .teaser > div, must have a position property, and since we don't want to move that div tag, we simply use position: relative with a left/right/top/bottom properties.

If everything is correct, your page should look like:

So very close now! Our icons need to be a different color to match our sample, and we need a little space between them and their following text:

```
.teaser > div > h2 > i {
font-size: 0.8em;
color: turquoise;
margin-right: 0.25em;
}
```

And we're done! Your page should look like (scrolled all the way to the top):

Review this ENTIRE lesson! Be sure you understand every concept that went into this page breakdown and the HTML and CSS that we used to build it - and why! There are many other approaches we could have taken, but the decisions we made are logical and certainly a good solution. If there is anything you don't understand, be sure to ask about it! This is probably the single more important lesson in this course, as it combines nearly everything you've learned and shown you how to logically approach breaking down a page design step-by-step, and come up with how to use the tools and techniques you've learned to develop this page from the ground up. This is the heart and soul of web development, so review this lesson - then review it again!

Submit

Submit