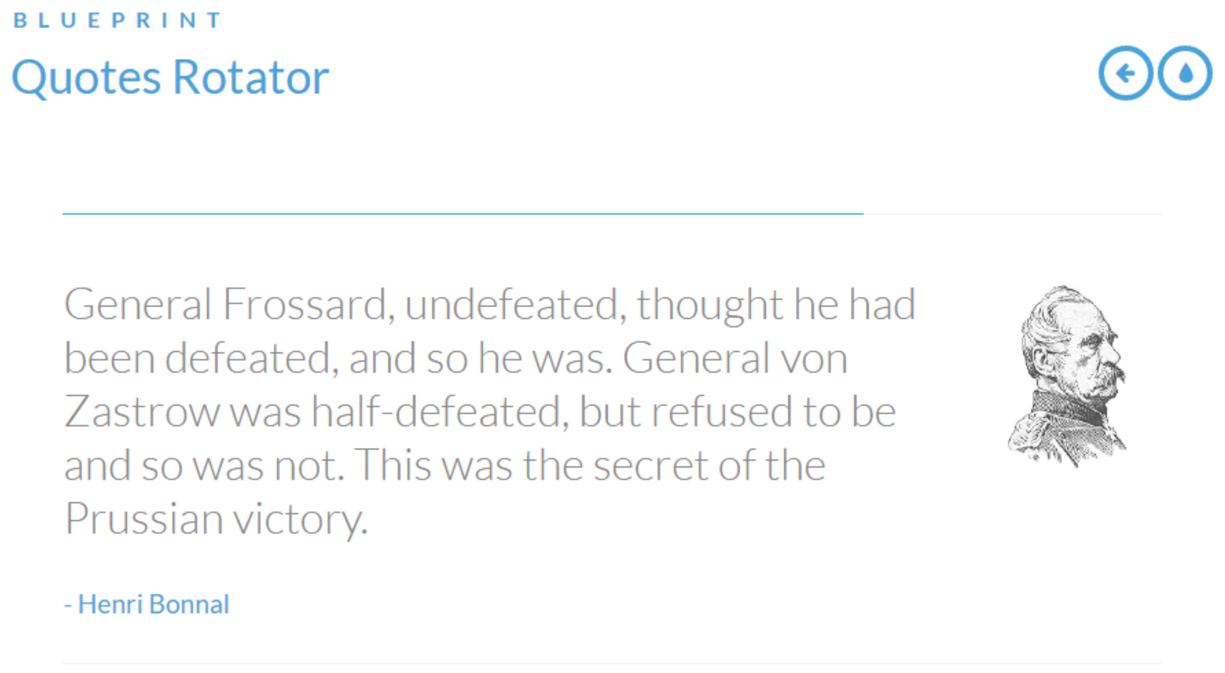
Quotes Rotator inClass Project

Based on a codrops.com Blueprint tutorial

In this inClass tutorial project you will be creating a carousel app that displays quotes which rotate (by fading out and in) on a timed basis shown via a progress bar. When the last quote is done, the app will cycle back to the first quote and cycle through the quotes again. The finished app will look like this:



BLUEPRINT

Quotes Rotator

General Frossard, undefeated, thought he had been defeated, and so he was. General von Zastrow was half-defeated, but refused to be and so was not. This was the secret of the Prussian victory.

- Henri Bonnal

Note that only one quote is shown at a time with a progress bar above the quote showing how much time is remaining until the next quote is displayed. Again, when the progress bar interval completes, the visible quote will fade out and the next quote will fade into view.

This kind of app could be used in many different ways such as displaying products, etc... As such, this would be a good candidate to develop as a jQuery plugin so others could adapt it to their needs. Remember that jQuery plugins allow us to create new methods that are not a part of the core jQuery library, but can be "plugged-in" to the library in a web application. Also, be aware that this type of app could be just one component of a larger web page!

Learning Objectives:

- CSS basics review – attributes and selection techniques
- CSS positioning for layout: relative and absolute positioning, float

- Advanced CSS topics such as: local storage, icon fonts, and ::before and ::after pseudo elements
- Responsive design using media queries
- jQuery plugin design and development
- JavaScript's prototype property and custom object definition

Starting Resources:

You have been provided a folder of starting files for this project under your lesson in E360. It contains the complete index.html file for the app as well as two CSS files (inside the css folder), fontawesome font-related files (fonts folder), images for the project (images folder), and two JavaScript files (js folder).

Download the QuotesRotator_startFiles folder with its files and store it somewhere on your local machine so you can work with it.

If you don't already have the Brackets editor installed on your machine, go to brackets.io to download and install it. It is a free editor that was specifically made for working with HTML, CSS, and JavaScript and we will be using it during this class. *If you have another editor that you really like to use for web apps, you can feel free to use it as well as long as I can open your files*.

Once you have Brackets installed, you should be able to *right-click* on your QuotesRotator_startFiles folder and choose Open as Brackets Project in Windows. On a Mac, Brackets does not provide this feature that I know of so you may need to open the files more manually.

Let's get started…

When you are provided with starting HTML, you should look it over doing some detective work to answer the following questions:

- What does the **structure** of the document look like? You are looking for things like the types of tags and their meaning (semantics), the order of the tags, and how they are nested.

  Reminder: **tags** are formally called HTML **elements** and these are represented as ***HTMLElement*** objects in JavaScript.

Did you know that each HTML document is structured like a family tree? As the browser moves through the HTML it builds and stores this family tree showing the tag hierarchy (parents, children, grandchildren, etc…) in something called the DOM (Document Object Model). Every tag (HTMLelement object), every tag attribute (attribute object), and every chunk of text content (text nodes or objects) are part of the DOM.

- What **id**'s and **classes** do you see on the tags?

- Do you see any **patterns** in the structure of the tags that could be important to how you would access these element objects (tags) via your CSS and/or JavaScript code?

Looking through the provided HTML:

Note: I may not show the entire HTML file here so be sure to look through the provided index.html file in your code editor in its entirety…

```
<html lang="en" class="no-js">https://modernizr.com/
        <head>
                <title>Quotes Rotator</title>

                <link rel="stylesheet" href="css/default.css">
                <link rel="stylesheet" href="css/component.css">

                <script src="js/modernizr.custom.js"></script>
        </head>
```

Note that a class of "no-js" has been added to the HTML tag. This will be used in concert with our Modernizr JavaScript file to determine if the user's browser supports JavaScript vs having JavaScript disabled. If JavaScript is supported, the Modernizr code will change this class to "js". Now we can use this "no-js" class for selection in our CSS to apply certain styles if the user's browser has JavaScript disabled…

You can see the Modernizr JavaScript file I'm referring to being brought in via the script tag at the bottom of the head section (modernizr.custom.js). This file is provided as part of the project's start files and is used to determine if the user's browser supports certain features such as JavaScript and CSS transitions. It creates an object named **Modernizr** that will have a property named *csstransitions* which will be a Boolean variable. If it contains *true* then the browser supports CSS transitions. If it contains *false*, it doesn't. Simple and convenient huh? For more information about Modernizr, the feature detection JavaScript library for HTML5/CSS3, go to modernizr.com.

Why was the modernizr.custom.js file up in the head section vs at the bottom of the body where our imported JavaScript files should normally be placed? Well, we want the Modernizr feature detection code to run right away before the body content begins rendering so we know what the user's browser supports right away.

Did you notice that we are linking in two CSS files in the head section? We will get to those later…

Ok? Let's continue with the HTML structure of the body…

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

         <header class="clearfix">
                  <span>Blueprint</span>
                  <h1>Quotes Rotator</h1>
                  <nav>
                           <a href="http://tympanus.net/Blueprints/NestedAccordion"
class="icon-arrow-left" data-info="previous Blueprint">Previous Blueprint</a>

                           <a href="http://tympanus.net/codrops/?p=14591" class="icon-
drop" data-info="back to the Codrops article">back to the Codrops article</a>
                  </nav>
         </header>
```

Things to note in the above code:

- All of our page content is wrapped inside of div.container (a <div> with a class of "container")
- We have a header tag with a class of "clearfix" that has several elements inside it (a span tag, h1 tag, and nav tag).
- The links (<a> tags) have some interesting features: classes as well as a data-info attribute. What is that?

Well, HTML5 allows us to create custom attributes for tags that we can name as we would like as long as they begin with “data-“. These attributes can, and often are, used to store custom data (their value) right with that attribute object in the DOM (local storage). Cool huh?

So, in the first link we are storing the string value “previous Blueprint” in the data-info attribute of that <a> element object in the DOM. We can retrieve that value (based on the attribute name) in either our CSS or JavaScript code.

Ok, let’s take a look at the next part of the body’s structure which is the main content of the page!

```
<div class="main">
        <div id="quoteRotator" class="quoteRotator">
                <div class="quoteContent">
                        <img src="images/roon.png" alt="img01">
                        <blockquote>
                            <p>General Frossard, undefeated, thought he had been
defeated, and so he was.  General von Zastrow was half-defeated, but refused to be
and so was not.  This was the secret of the Prussian victory.</p>
                            <footer>Henri Bonnal</footer>
                        </blockquote>
                </div>
                <div class="quoteContent">
                        <img src="images/macmahon.png" alt="img02">
                        <blockquote>
                            <p>One day I would like to greet the Germans here; not even
a field mouse would come out alive.</p>
                            <footer>Marshal Patrice MacMahon</footer>
                        </blockquote>
                </div>
```

Note that more div.content tags follow at this point… Notice any patterns?

Things to note in the above code:

- Our main content is wrapped in a div with a class of “main” and then is immediately wrapped again in a second div that has an id and class with the name of “quoteRotator”. Why is this?

  We’ll be using these two “wrapper” div’s in the CSS to do a few things.

  Also, developers will sometimes use the same value for both an **id** and **class** name for a tag. This allows them flexibility in what they wish to use for selection later on.

- Did you notice that div#quoteRotator contains several <div>’s that follow a pattern?

  All of these <div>’s have the same class (“quoteContent”) and they each contain the *same tag structure*: an <img> tag followed by a <blockquote> tag which contains a paragraph and <footer> tag.

  Each of these .quotecontent div’s represent a quote (image, the quote text itself, and the quote’s author).

  Patterns like this can be extremely useful to us in how we structure our code logic!

  Also, notice on the finished page you saw earlier only one of these quotes is showing at a time. How is this the case when we see there are several of them in the HTML? Through CSS and JavaScript magic of course!

  If you preview the page right now, you will see that all of these quotes are shown. We will need to fix that up later… In Brackets click on the lightning bolt in the upper right corner of the window (Live Preview).

  Answer OK to any dialog that pops up and a Chrome browser window will launch showing your web page in its current state. It should look like the following right now (notice that some of the page content on the bottom is cut off and some strange artifacts are showing up in the scroll bar…):

BLUEPRINT

Quotes Rotator

Previous Blueprint back to the Codrops article

General Frossard, undefeated, thought he had been defeated, and so he was. General von Zastrow was half-defeated, but refused to be and so was not. This was the secret of the Prussian victory.

Henri Bonnal

One day I would like to greet the Germans here; not even a field mouse would come out alive.

Marshal Patrice MacMahon

Fleschuez, one of those diehard officers, was shot in the chest as he urgently conferred with a colleague. He fell down, gasping for air but otherwise unharmed; he had been saved by his wallet, which he had tucked into his breast pocket before the battle.

Geoffery Wawro

He was living day to day, confining himself to the routine details of administration which he understood, and trusting to his good luck to pull him through.

Now let's begin looking at the provided CSS files starting with default.css.

Why would we have more than one CSS file? This could be done to break up a long piece of CSS code or, as in this case, you may have separate CSS files that deal with different components of your page. In our case (or codrops' case) the default.css file will deal with styles that are consistent on all codrops web pages (usually in the top header of each page) while component.css will deal with the specific content of our page (which will likely differ from the content styles of other codrops pages…).

Let's start with the provided code in default.css:

```
@import url(http://fonts.googleapis.com/css?family=Lato:300,400,700);

@font-face {
	font-family: 'fontawesome';
	src:url('../fonts/fontawesome.eot');
	src:url('../fonts/fontawesome.eot?#iefix') format('embedded-opentype'),
		url('../fonts/fontawesome.svg#fontawesome') format('svg'),
		url('../fonts/fontawesome.woff') format('woff'),
		url('../fonts/fontawesome.ttf') format('truetype');
	font-weight: normal;
	font-style: normal;
}
```

First we are importing a Google font (Lato) so a user's machine doesn't need to have this font locally.

Then, we use the @font-face at-rule to define and use an extended font set. In this case we will be using a subset of the fontawesome font set (see fontawesome.io) that was provided in our project's start files under the fonts folder. The fontawesome font set consists of **icon fonts *which then can be treated like text*** in our HTML and CSS. This means we don't need to use images for these symbols which improves our page's performance. Yay!

You can go to fontawesome.io, generate a subset of fonts that you want to use in your app, and download them for use. We'll see later that we can access the symbol we wish from the font set using a Unicode character.

```
body, html {
	font-size: 100%;
	padding: 0;
	margin: 0;
}

/* Reset for all tags including any first or last children added via CSS. */
*,
*:after,
*:before {	/* Note use of browser prefixes here */
	-webkit-box-sizing: border-box;
	-moz-box-sizing: border-box;
	box-sizing: border-box;
}

/* Clearfix hack by Nicolas Gallagher: http://nicolasgallagher.com/micro-clearfix-hack   - Read more about it… */
.clearfix:before,
.clearfix:after {
	content: " ";
	display: table;
}

.clearfix:after {
	clear: both;
}

body {
   font-family: 'Lato', Calibri, Arial, sans-serif;
   color: #47a3da;   /* color we'll be using throughout as our theme color */
}

a {
	color: #aaa;
	text-decoration: none;
}
```

Most of the CSS code above should be self explanatory. If you have any questions about it, try looking up the attributes and what they are used for and then let me know if you have further questions.

The box-sizing CSS property is used to alter the default CSS box model used to calculate widths and heights of elements. Using the **border-box** value means the width and height properties include the padding and border, but not the margin. This is the box model used by Internet Explorer when the document is in Quirks mode.

```
a:hover {
        color: #000;
}

.main,
.container > header {
        width: 90%;
        max-width: 69em;    /* cannot exceed width of 69em */
        margin: 0 auto;        /* center within its container */
        padding: 0 1.875em 3.125em 1.875em;     /* order is TRBL – top, right, bottom, left */
}

.container > header {
        padding: 2.875em 1.875em 1.875em;      /* 3 values = top, left and right, bottom */
}
```

Apply some styles to *both* (via the comma in our selector) our **div.main** and **header** that is a direct child of our div.container. Then alter the padding for just the **header**.

Again, this is CSS you should be familiar with, but look it over carefully checking out the comments I've added. Now, let's finish off the provided CSS code…

```
.container > header h1 {
        font-size: 2.125em;
        line-height: 1.3;
        margin: 0;
        float: left;
        font-weight: 400;
}

.container > header span {
        display: block;      /* span tags are normally have display set to inline */
        font-weight: 700;
        text-transform: uppercase;
        letter-spacing: 0.5em;
        padding: 0 0 0.6em 0.1em;
}

.container > header nav {
        float: right;    /* float nav containing our two <a> tags to right margin */
}
```

Notice the float: left; above in the h1 descendant of our header tag. This causes the h1 to go as far to the left as is allowed on the line of content it is on taking into account padding, margin and any other content already to its left.

Also, note that float'ing content to the **left** also *causes following element(s) to wrap up to the **right*** of the h1 if there is enough space for it. Floating a tag to the right would likewise cause following element(s) to wrap up to its left as long as there is space.

Ok wonderful, but now its finally time for us to begin entering our own CSS code to bring this design to life!

Finishing off default.css by completing styling of the header's nav <a> tags:

Now as we type the following code I'll give it you in chunks to type in, but you can (and should) test as you go. In other words, don't always type in the entire shown block of code before trying it out. Type in the selector with the curly braces and then test (by looking at the browser) after typing each line of code (or every few lines) to see what that style accomplishes on your page.

I'll "verbalize" what the selector is supposed to be selecting before showing the code:

> Select all (both) of the <a> tags that are descendants of any <nav> tags that are descendants of any header tags that are direct children of any element with a class of "container". Got that! ☺
>
> Now type in the following rule to style these <a> tags and remember to test frequently to see the changes that occur.

```
.container > header nav a {
   display: block;
   float: left;
   position: relative;
   width: 2.5em;
   height: 2.5em;
   background: #fff;
   color: transparent;
   margin: 0 0.1em;
   border: 4px solid #47a3da;
   border-radius: 50%;
   text-indent: -500em;    /* -8000px */
}
```

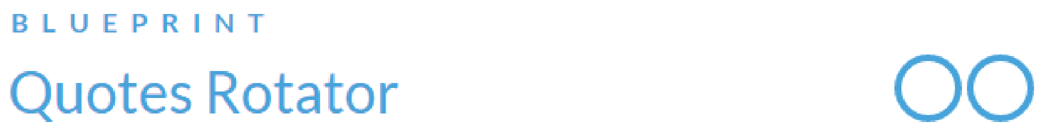
Things to note in the above code:

- Since <a> tags are inline elements, set them to display: block so they are not treated as inline.
- Remember they are children of the <nav> that was previously floated to the right.
- Float them to the left within the <nav> (which is a box). This will have the effect of putting the two <a> tags side-by-side. When you float multiple elements to left (or

right), you create side-by-side columns because following tags wrap up provided there is enough space for them to do so.

- Use position: relative to make each of the <a> tags "positioning parent" elements. This will be used shortly to absolutely position their children elements (we will be creating shortly via ::after) relative to the edges of the <a> tag positioning parents.
- The transparent color keyword represents a fully transparent color, i.e. the color seen will be the background color which in our case is white (#fff).
- Set them to same width and height and make them circular via border-radius. Give a slight margin value to separate them a bit, set a 4px wide border so you see the circular shape.
- Finally, move the text content of the <a> tags far out of sight to the left because we are going to replace the text with icon fonts shortly. We still want the text content of the links to exist for accessibility software.

You should now see this up in the header.



Create a **::after** child on each of the <a> tags. **::after** is a CSS pseudo-element that allows you to add a **last child** to an element directly via CSS bypassing HTML code. You must supply the style with a content attribute at a minimum even if the content value is empty ("").

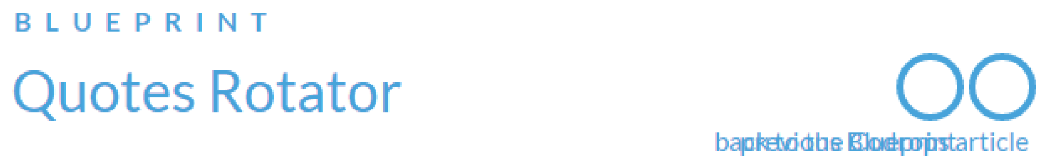
```
.container > header nav a::after {
   content: attr(data-info);
   color: #47a3da;
   position: absolute;
   top: 120%;
   right: 0;
   width: 600%;
   text-align: right;
   pointer-events: none;
   opacity: 0;
}
```

Things to note in the above code:

- **::after** creates a new, last-child element inside each of the selected <a> tags. You can use either a single colon or double colon with these. Either will work, but :: is the newer way of doing it.
- The required content attribute is being set to the value of the selected <a> tag's data-info attribute that was set in the HTML using the CSS attr() function. For the first <a> tag this will the string value "previous Blueprint" and for the second <a> tag it will be "back to the Codrops article".
- These new after children elements will be absolutely positioned *relative to the closest position parent element* (in our case, the <a> tag parent because we set position: relative on them).
- The top edge of each after child will be positioned 120% down from the top edge of its <a> tag position parent and the right edge will be lined up with the right edge of its <a> tag position parent.

  Note that in this case, the positioning parent element (<a>) is also the parent element. That is **not** always the case.
- Make the width of these after children 600% of their parent tag's (<a>) width so we can see the text.
- Set pointer-events: none so the after children cannot be selected.
- Make them transparent via opacity: 0

Our header's nav links should now look like this with opacity set to 1 for testing (don't forget to set it back to 0 though…):



Yikes! We see the link text, but it is overlapping and that just doesn't look good. Don't worry. Turn the opacity back to 0 and we'll fix that up in a moment.

Put a hover pseudoclass on the ::after child. Note that the :hover must be placed on the <a> tag for this to work.

```
.container > header nav a:hover::after {
   opacity: 1;
}

.container > header nav a:hover {
   background: #47a3da;
}
```

Things to note in the above code:

- When the ::after child is hovered over, change its opacity back to 1 so we can see just the hovered over one's text content (solving our earlier problem of them being overlapped when both were visible).

- Also, add a hover effect to the <a> tag itself which changes its background color to our nice blue color. You'll see why we do this shortly.

Test out your hover effects. You should see the text of the hovered link's after child appear when the link is hovered over and the circle's background color should turn blue.

Next, create **::before** pseudoelements on any element with either the class .icon-drop or .icon-arrow-left. Notice these are the nav's <a> tags again. The **::before** pseudoelement creates a before child (always inserted as the *first child*) inside the selected element(s).

Also, note that these <a> tags already have **::after** (last) child elements created in them (we just did this). So, now they have a new first child (**::before**) and new last child (**::after**).

You may want to do the style rules that have the content attribute first that set the icon content of each <a> tag's first child so you can see them and how they change as the styles are applied to both of them.

```
/* Icons */
.icon-drop::before,
.icon-arrow-left::before {
   font-family: 'fontawesome';
   position: absolute;
   top: 0;
   width: 100%;
   height: 100%;
   font-style: normal;
   font-weight: normal;
   line-height: 2;
   text-align: center;
   color: #47a3da;
   text-indent: 8000px;
   padding-left: .5em;
}

.icon-drop::before {
   content: "\e000";
}

.icon-arrow-left::before {
   content: "\f060";
}

.container > header nav a:hover::before {
   color: #fff;
}
```

Things to note in the above code:

Apply the following styles to the ::before children of both of the elements

- Set font on these new children to 'fontawesom' so we have access to our icon fonts and reset font-style and font-weight to normal.

- These new before children elements will be absolutely positioned *relative to the closest position parent element* (in our case, the <a> tag parent because we set position: relative on them).

- The top edge of each before child will lined up with the top edge of its <a> tag position parent and will have the same width and height as the <a> parent tag.

- Set a double line height on them to increase vertical spacing, center them (remember they are text), and set color to our theme color.

- Very important to text-indent them back in by the same amount we moved the <a> tag content offscreen earlier so we can see them.

- Finally, add just a bit of padding on their left to truly center them within the circle of the <a> parent tag.

- ***Note we did NOT set a content attribute here because we are styling two before children here. We want each of them to have separate content.

Apply the following styles to the ::before children of each element individually

- Ok, now we can set the content on each of the before children of our nav's <a> tags separately using the fontawesome Unicode character code for them. You can see the left arrow's page, with its Unicode value on fontawesome's website here.

Finally, place a hover effect for the ::before children of nav's <a> tags

- Here is where we flip the colors on hover. Make the text color of the icon fonts change to white on hover. Remember, the background of the link already changes to our blue on hover from earlier…

After applying and testing all of the above styles you should now see this:

BLUEPRINT
Quotes Rotator

Nice! We have cool, efficient text icons (via fontawesome's icon fonts) that show a text hint when the link is hovered over. Note the drop icon is codrops' branding icon.

Next let's work on the styling of the rest of our page's content in component.css.

For component.css I'll be typing in comments about what we want to do next and then showing the code to accomplish that. Follow along, look things up when necessary and feel free to add in your own notes to better establish your learning. I'll show you screenshots every so often, but you should test more often than that by looking at your browser results. Let's get started!

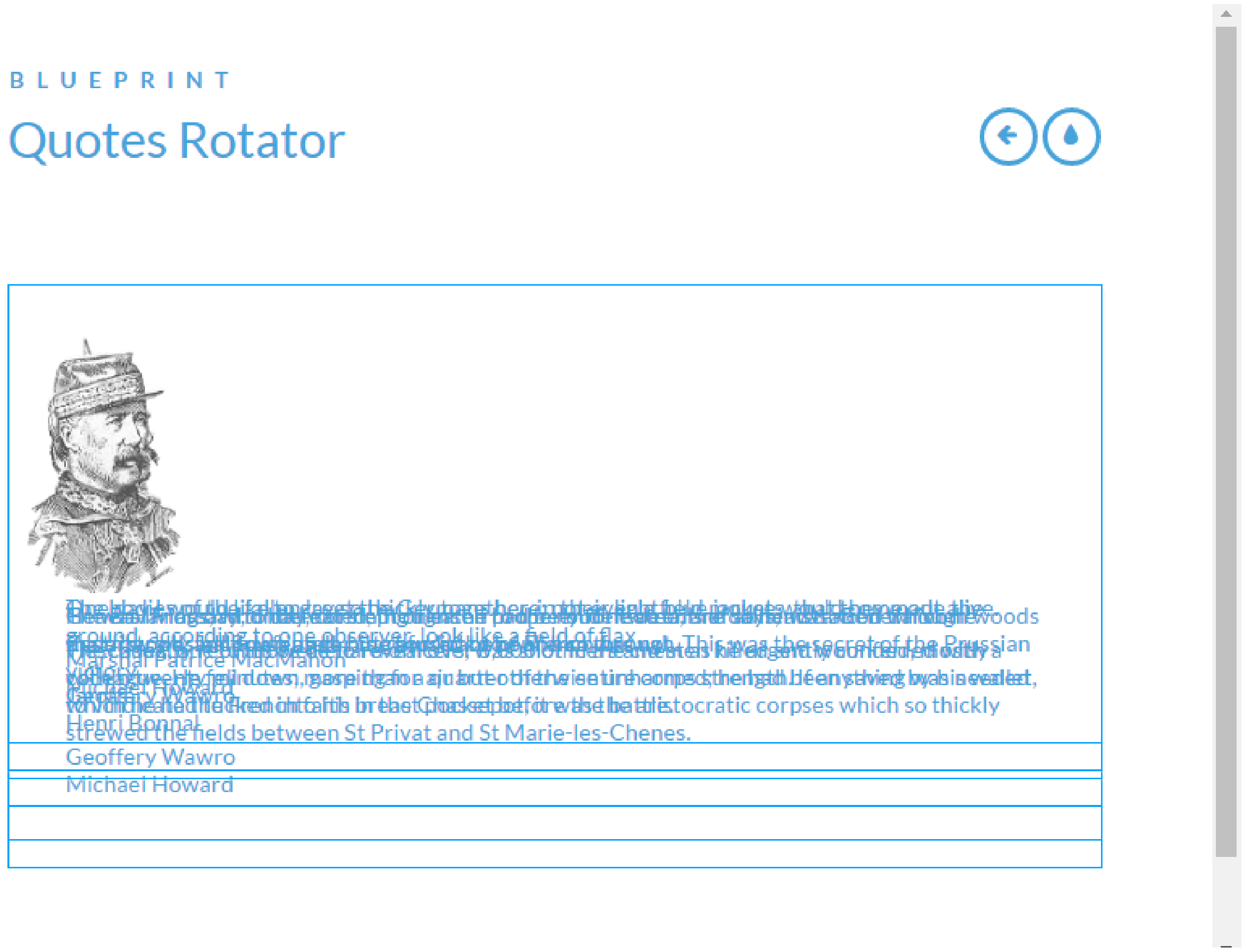
Select the div whose class is "quoteRotator", make it a **positioning parent** so we can absolutely position any of its descendant elements *relative to its edges*. Also, add some margin space above and below it and center it within its parent container (which is div.main).

```
.quoteRotator {
   position: relative;    /* makes this element a positioning parent */
   margin: 3em auto 5em;     /* 3em for top, auto for left and right, 5em for bottom */
   max-width: 50em;      /* 800px */
   width: 100%;
   min-height: 25em;      /* 400px */
}
```

These styles should cause the main content (the quotes) to drop down a bit due to the top margin value being applied (3em). No other changes will be too noticeable at this point.

Select the div whose class is “quoteContent” each of which represent one of our **quotes**. Remember, they each contain a pattern of an <img> and <blockquote> where the blockquote tag contains <p> and <footer> tags. There are several of these div’s.

```
.quoteContent {

        Let’s absolutely position all of these quote div’s. By setting top to 0, we will be
stacking all of them on top of each other with their top-left corners at the top-left
corner of their positioning parent (which is div.quoteRotator from earlier). Note that
top: 0 is top’s default value, but we’ll set it here just in case it was set anywhere
previously.
   position: absolute;
   top: 0;
   min-height: 12.5em;      /* 200px – must be at least this tall */

        Set a very thin border on the top and bottom of these div’s and put 2em’s of
padding between the div content and the borders.
   border-top: 1px solid #f4f4f4;
   border-bottom: 1px solid #f4f4f4;
   padding: 2em 0;

        z-index is for stacking order (the higher the value, the closer the item is to the
forefront in the z-axis, i.e., the closer it is to us) and only has an effect when the
position attribute is used.

        Set width of these div’s to the same width as their physical parent tag’s width –
        in this case, still div.quoteRotator.

        Lastly set opacity, but keep it at 1 until you test and see your page looking like
        the following screenshot.  ***Then come back and set opacity back to 0 as we
        don’t want any of the quotes initially showing.

   z-index: 0;
   width: 100%;
   opacity: 1;   /* set this back to 0 after you confirm the other styles are working */
}
```



BLUEPRINT

Quotes Rotator

[illegible] woods ground, according to one observer, look like a field of flax. [illegible] This was the secret of the Prussian [illegible] tocratic corpses which so thickly strewed the fields between St Privat and St Marie-les-Chenes.

Marshal Patrice MacMahon

Henri Bonnal

Geoffery Wawro

Michael Howard

Did you remember to set the **opacity** back to **0** for your div.quoteContent's?

Select any div whose class is "quoteContent" which is a ***descendant*** of our div.quoteRotator element, but only if div.quoteRotator is a descendant of an element that has a class of "no-js". Remember that our <html> element started with a class of "**no-js**", but when our modernizr.custom.js file executed in the head section, it changed this class to "**js**" if the user's browser has JavaScript enabled (which all browsers do by default). You can verify this by right-clicking your browser, choosing **Inspect**, and looking at the live HTML's <html> element to verify that its class has been changed to "**js**". Notice that our Modernizr code also added the "**csstransitions**" class there to check if the browser supports them.

In other words, our <html> element should only still have a class of "**no-js**" if the user's browser has JavaScript disabled. If that is the case, then we will remove any bottom border from our .quoteContent div's. You won't see this change unless you disable JavaScript in your browser.

```
.no-js .quoteRotator  .quoteContent {
   border-bottom: none;
}
```

Ok, let's keep going…

Select any element(s) whose has classes of "quoteContent" AND "quoteCurrent". Note that **none of our elements (tags) currently have a class of "quoteCurrent"**. That is ok as we will be adding this class to elements in our jQuery plugin's code.

Elements with both these classes (or in the case of JavaScript being disabled, our .quoteContent div's) will be setup as positioning parents (**position: relative**) with a high **z-index** value so they are on top of any overlapping elements. They will be clickable and fully opaque (visible).

```
.quoteContent.quoteCurrent,
.no-js .quoteRotator .quoteContent {
   position: relative;
   z-index: 100;
   pointer-events: auto;   /* make selected element(s) clickable */
   opacity: 1;
}
```

Note that the above styles really won't affect our page at this point so don't expect any changes to show up unless JavaScript is disabled. We'll see this effects applied when we add the "quoteCurrent" class to any of .quoteContent div's in our plugin.

Onward to the styling of our progress bar!

Ok, I know this may sound like a broken record at this point, but the styles we will set up for our page's progress bar will not show anything as far as changes on the page at this point. That is because we do not yet have any element setup as a progress bar in our HTML (with a class of "quoteProgress").

We will be dynamically adding this element via our JavaScript code when we get to coding our jQuery plugin. However, we want the styles that element will be using when it is created to be code now in our CSS. So here we go…

Select any element(s) with a class of "quoteProgress". There is no such element right now, but one will be created later in our JavaScript so let's set up its styles now.

```
/* Progress Bar */
.quoteProgress {
```

We will absolutely position this element at the top-left corner (top: 0) of its closest positioning parent element (we won't know what this is until we create this new element with a class of "quoteProgress" and add it somewhere in the DOM).

```
   position: absolute;
   top: 0;
   background: #47a3da;   /* our nice blue theme color again */
```

Start the progress bar, which is only going to be 1 pixel high, with no width so we can change its width dynamically over time in our JavaScript causing the bar to "grow" to represent a time interval.

```
   width: 0%;
   height: 1px;
   z-index: 1000;   /* make sure this bar is visible above any other elements */
}
```

Ok, now we will move on to styling our quote container and text along with the quote's author text. Again, this won't show up as we set the opacity to 0 on our .quoteContent div's that contain these elements. To see the changes simply go back and temporarily set the opacity on .quoteContent back to 1 (just don't forget to set it back to 0).

Select all <**blockquote**> elements that are descendants (inside of) our .quoteRotator div. Reset their **margin** and **padding** to **0** which will spread it out just a bit more horizontally.

```
.quoteRotator blockquote {
   margin: 0;
   padding: 0;
}
```

Select all of the paragraph elements inside our div.quoteRotator's blockquote elements. Increase their font size, set the text color to a medium gray, and set their font weight to just a bit less than normal (normal is 400). Also, adjust margin values a little to set up spacing between the paragraphs and their surrounding content.

```
.quoteRotator blockquote p {
   font-size: 2em;
   color: #888;
   font-weight: 300;
   margin: 0.4em 0 1em;
}
```

Ok, now let's select and style the footer (author) within those same blockquotes. Increase the font size a little.

```
.quoteRotator blockquote footer {
   font-size: 1.2em;
}
```

Select the same **footer** elements again and use the **::before** pseudoelement to create a new element through CSS. This new element will be inserted as the first child of each of our footer tags. We are using the content attribute to set the new child's content to the string "- " (a dash followed by a space). Just a little detail we are adding.

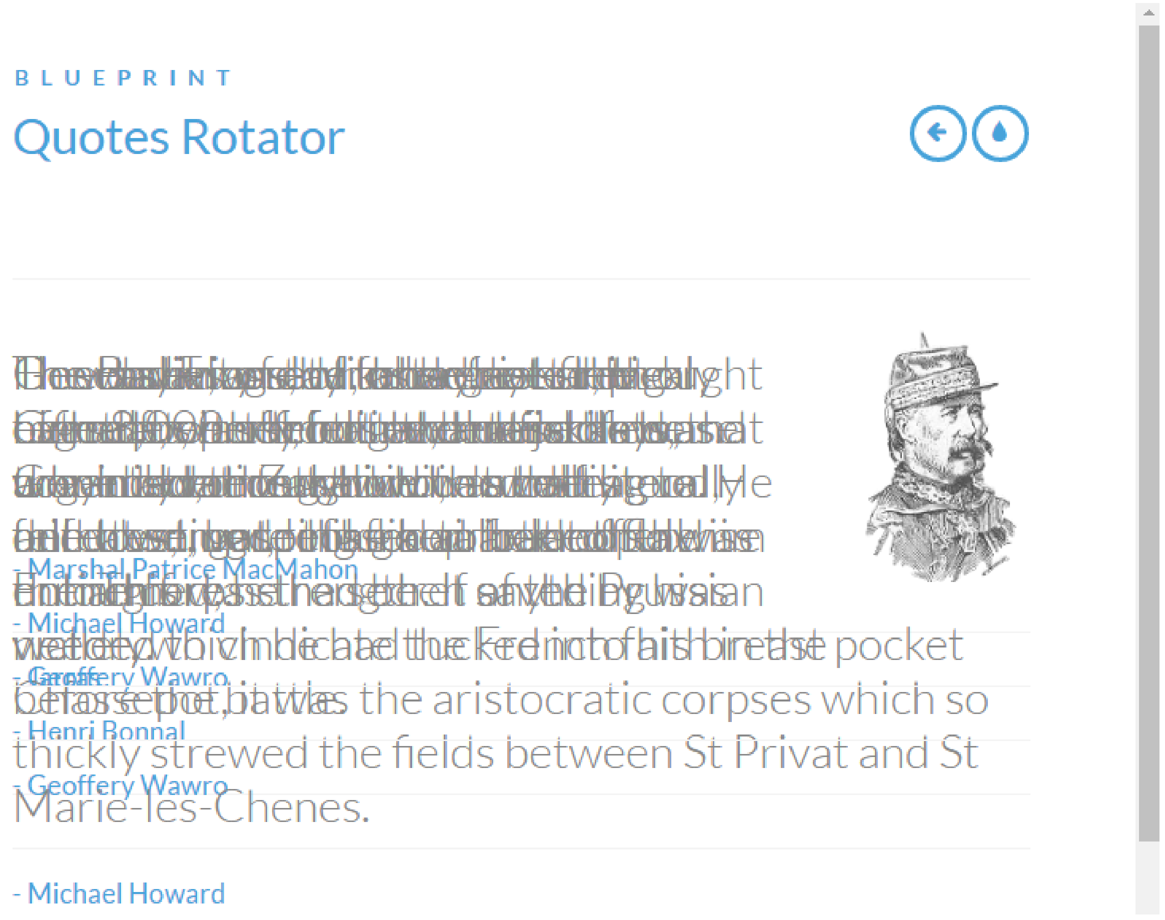
```
.quoteRotator blockquote footer::before {
   content: '- ';
}
```

Select all of the images inside of our .quoteContent div's and move them to the right margin. Remember that using a **float** will cause following tags to wrap their content up beside the floated element if there is space. Text content will automatically wrap up and flow around and below the floated element as you see our blockquote content doing as it wraps up to the left beside and continuing to flow below our floated image.

Place 3em's of margin space between them and any elements to their left.

```css
.quoteContent img {
   float: right;
   margin-left: 3em;
}
```

Here is what our page looks like now:



BLUEPRINT

Quotes Rotator

[illegible] the aristocratic corpses which so thickly strewed the fields between St Privat and St Marie-les-Chenes.

- Marshal Patrice MacMahon

- Michael Howard

- Geoffery Wawro

- Henri Bonnal

- Geoffery Wawro

- Michael Howard

Don't worry about the overlapping quote content. We'll be fixing all of that to work properly in our JavaScript code…

Once you have this working to your satisfaction, don't forget to go back and reset **opacity** to **0** on your **.quoteContent** selection.

One last thing we'll do with our CSS is to set up a quick media query to see one in action.

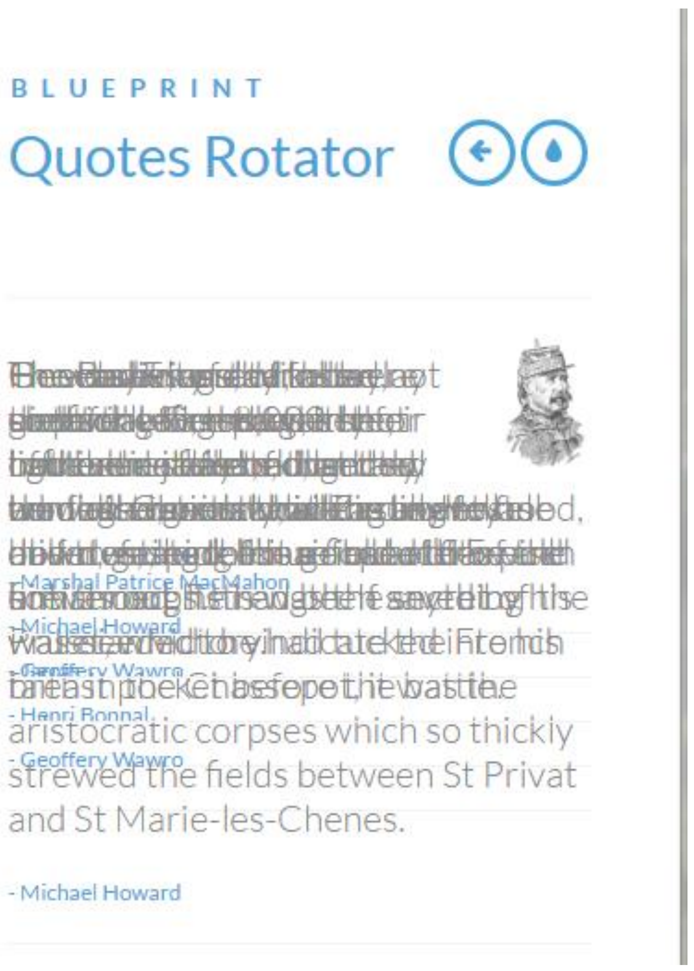
Setting a media query for a breakpoint of 28.75em's (460 pixels):

Apply the following styles if the browser is on a device screen and the device's screen has a width of 28.75em's or less (a standard mobile phone size).

```
/* Media Query */
@media screen and (max-width: 28.75em) {

   Set our div.quoteRotator's font to be 70% of what it was.

   .quoteRotator {
      font-size: 70%;
   }

   Decrease the size of our quote-related images

   .quoteContent img {
      width: 5em;
   }
}
```

You can manually resize your browser window's width and you should see these changes take effect when it reaches 28.75ems or less (460px). Note that you would need to set opacity back to 1 on your .quoteContent div's to see this.

Your page would look like this when you shrink it to this breakpoint:



[illegible]

- Marshal Patrice MacMahon

- Michael Howard

- Geoffery Wawro

- Henri Bonnal

- Geoffery Wawro

aristocratic corpses which so thickly strewed the fields between St Privat and St Marie-les-Chenes.

- Michael Howard

Ok, next we work on our JavaScript. We will be building a **jQuery plugin** that will bring our timed quote animations to life. We'll also add some JavaScript to the bottom of your .html file's body to call the new jQuery method our plugin creates and "plugs in" to jQuery.

This will be done via a video tutorial consisting of video lessons for creating our QuotesRotator jQuery plugin.