Playing with typography

If you don't know I have a ton of different type and layout experiments in their own website. I'll start sharing some of the demos I've been working on via Twitter and explain the code for some here.

The full demo is available in Codepen.

The HTML

The HTML is as simple as it comes. It's an h1 heading element inside a `container` div. We will use the container to place the title and the `myTitle` heading as the target for lettering.js

```
<div class='container'>
  <h1 class='myTitle'>Nightfall</h1>
</div>
```

This example uses only the heading. We could add more text and assume that this is the title for a document.

Javascript

Unlike most of my projects, Lettering.js is a jQuery plugin. While I don't normally use or recommend jQuery for production (it's not a value judgement on jQuery, it's just an additional dependency that is usually not needed) I'll make an exception for this demo but will illustrate an alternative without jQuery and some of the problems I encountered when using it.

The first part of this section is to add jQuery. To do so I use a technique I learned from the HTML5 Boilerplate that works when jQuery is not present for whatever reason.

```
<script src="http://code.jquery.com/jquery-1.12.4.min.js"
  integrity="sha256-ZosEbRLbNQzLpnKIkEdrPv7lOy9C27hHQ+Xp8a4MxAQ="
  crossorigin="anonymous"></script>
```

```
<script>window.jQuery ||
  document.write('<script src="/js/jquery-1.12.4.min.js"><\/script>'
</script>
```

We first load jQuery from a CDN as normal. In this case I've chosen jQuery's own CDN.

As soon as we load it we check for the global `window.jQuery` object. If it exists we use it, otherwise we use `document.write` to dynamically create a link to a local version of the script matching the version we get from CDN.

Since jQuery is still popular we will seldom encounter this issue in existing projects but brand new projects, particularly when starting in your workstation.

Next, we load Lettering.js and initialize it.

```
<script src='js/jquery.lettering.js'></script>
<script>
  $(document).ready(function() {
    $('.myTitle').lettering();
  });
</script>
```

The rest of the work is done in CSS.

We first import that Typekit project that we want to use. Typekit recommends using the `link` element to load the stylesheet but I want to make sure that the font is available before we do all the manipulation.

When defining the body element, I set the overall background color and the default font for the document, which is not the font we'll be using for the heading; this is on purpose.

```
@import url("https://use.typekit.net/aet8yjj.css");

body {
  background-color: #fbfbf6;
```

```
  font-family: Raleway, sans-serif;
}
```

The container element is where the magic starts. We set up a linear gradient for the background, the height and width for the element, the font size and the breaking behavior.

Because we will treat each letter as its own container we want to break whenever we need to.

One last item regarding the container. I've omitted the vendor prefixed syntax. Depending on what browsers you must support I recommend testing this to make sure that they support the syntax you provide for the gradient.

```
.container {
  margin-top: -1.25em;
  background-color: rgb(33, 35, 66);
  background: linear-gradient(to bottom, #212342 0%, #fff 100%);
  color: rgb(255, 255, 255);
  height: 100%;
  width: 65%;
  font-size:18em;
  word-break: break-all;
  overflow-wrap: break-word;
}
```

For the h1 ekenebt we do a few things: We setup the font we want to use, we make it all uppercase, we setup the line height to be closer than normal and finish by adding padding to the element so it won't be flush against the margins and lose some of the text shadow effect.

All spans elements that Lettering.js generates will get display: relative so we can play with moving them around.

```
h1 {
  font-family: 'b'bebas-neue'sans-serif;
  text-transform: uppercase;
```

```
  line-height: .65em;
    padding: .05em;
}

span {
  position: relative;
}
```

Lettering will dynamically inject a `span` element with a class equal to `char` plus a number indicating the location of the letter in the word we initialized.

> They all have three attributes in common:

- `z-index` to indicate the stacking order among the letters; larger positive numbers indicate a higher position in the stack, closer to the viewer and negative numbers indicate lower positions in the stack, away from the viewer
- `text-shadow` produces a shadow from the source element. Parameters are: offset-x (x-axis blur distance from text), offset-y (y-axis blur distance from text), blur-radius (the bigger the blur the wider and lighter it becomes) and color (the color of the shadow)
- `margin-left` to indicate how close letters are to each other

We can add other elements to individual characters as needed to get the effect that we wanted. One idea I've been playing with is to use SASS to generate random colors for each letter.

```
.char1 {
  z-index: 4;
  text-shadow: -0.02em 0.02em 0.2em rgba(10, 10, 10, .8);
  margin-left: -0.05em;
}

.char2 {
  z-index: 3;
  text-shadow: -0.02em 0.02em 0.2em rgba(10, 10, 10, .8);
  margin-left: -0.025em;
  top: 0.05em;
```

```
}

.char3 {
  z-index: 9;
  text-shadow: -0.02em 0.02em 0.05em rgba(10, 10, 10, .8);
  margin-left: -0.05em;
}

.char4 {
  z-index: 5;
  text-shadow: 0.01em -0.01em 0.14em rgba(10, 10, 10, .8);
  margin-left: -0.05em;
  top: -0.01em;
}

.char5 {
  z-index: 2;
  text-shadow: -0.02em -0.02em 0.14em rgba(10, 10, 10, .8);
  margin-left: -0.06em;
  top: 0.02em;
}

.char6 {
  z-index: 10;
  text-shadow: -0.02em -0.02em 0.14em rgba(10, 10, 10, .8);
  margin-left: -0.06em;
  top: -0.02em;
}

.char7 {
  z-index: 8;
  text-shadow: -0.02em -0.02em 0.14em rgba(10, 10, 10, .8);
  margin-left: -0.05em;
}

.char8 {
  z-index: 6;
  text-shadow: -0.02em -0.02em 0.14em rgba(10, 10, 10, .8);
```

```
  margin-left: -0.08em;
  top: -0.02em;
}

.char9 {
  z-index: 7;
  text-shadow: -0.02em -0.02em 0.14em rgba(10, 10, 10, .8);
  margin-left: -0.08em;
}
```

One last aspect is to make sure that it looks decent in our target devices and browsers. I have to look at it in an iPad and iPhone to make sure.

Non jQuery Alternative

Based on Jeremy Keith's gist this is a quick way to do some of the slicing and span/ class addition without having to use jQuery.

The HTML and CSS remain the same, although we may have to tweak the CSS to make it look identical. The Javascript changes to the code show below:

```
 function sliceString(selector) {
    if (!document.querySelector) return;
    var string = document.querySelector(selector).innerText,
        total = string.length,
        html = '';
    for (var i=0; i<total; i++) {
        html+= `<span class="char${i+1}">${string.charAt(i)} </span>`;
    }
    document.querySelector(selector).innerHTML = html;
}
sliceString('.myTitle');
```

This needs further testing, particularly in Firefox where some users of Jeremy's code reported problems