Week 6 – Google Fonts & scrollTop

Introduction

This week we're going to learn two things: First, how to integrate Google fonts into Web sites. Second, how to use a jQuery function called `.scrollTop()` to smoothly scroll the page to a particular location.

Google Fonts

For the most of the history of Web design, only a few 'standard' fonts have been available. A 'standard' font is essentially a font that designers could rely on being installed on the user's computer. If you used a font that wasn't commonly found on most computers, then the browser would display text using its default font, which was usually Times.

Developments in CSS and font technology, however, have changed this. Web fonts can be found in a variety of places but a useful central repository is the Google Fonts project - http://www.google.com/fonts/

Google Fonts allows you to select from a wide range of typefaces and it gives you the code to make them work in your Web site. There are a few different ways of using Google fonts, but the method we're going to advocate is the @import method which allows you to call the Google fonts into your stylesheet and then you can use them with the `font-family` property for any selector.

@import

The @import feature of CSS allows you to import styles from other stylesheets. In the context of Google fonts, it basically means that you have to do very little because Google has done all the hard work for you.

The only caveat of which you need to be aware, is that @import rules must go at the top of your stylesheet above all other rules. Other than that, it is stupidly easy to use Google fonts.

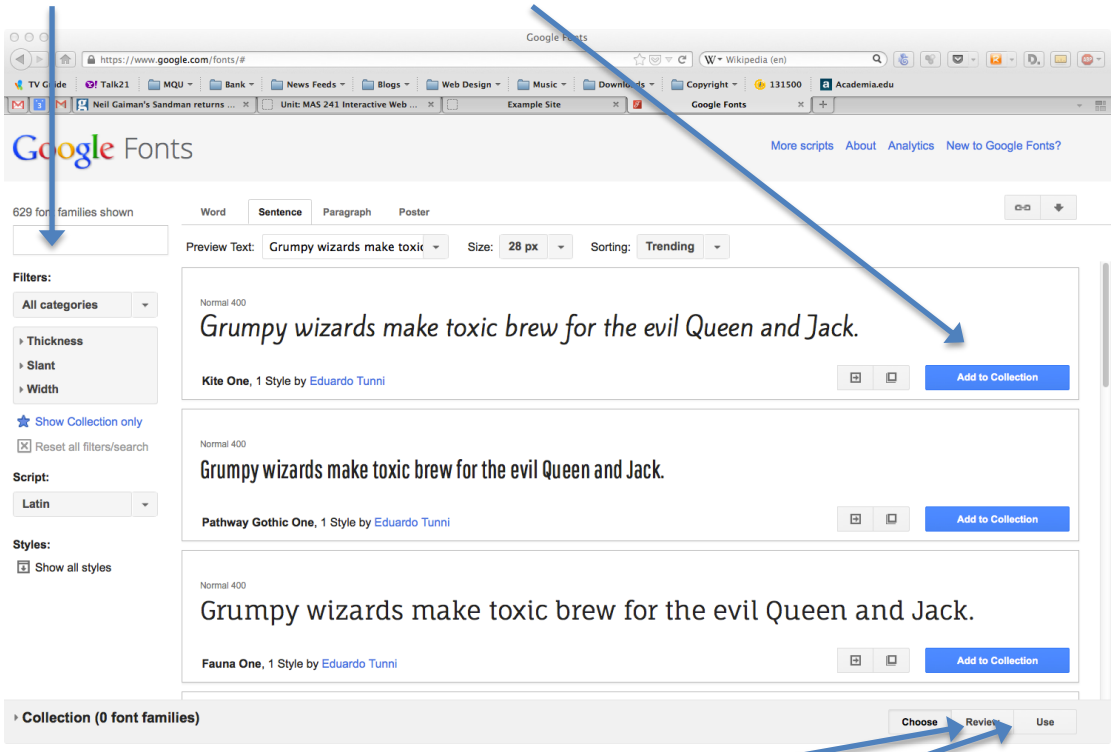
Using Google Fonts

First of all, go to http://www.google.com/fonts/

You'll be presented with a list of fonts and examples of what they look like. On the left-hand side of the page you can use the menu to filter for different types of font such as serif, sans serif, display and handwriting.

When you've found fonts you like, just click the 'Add to Collection' button next to each font.

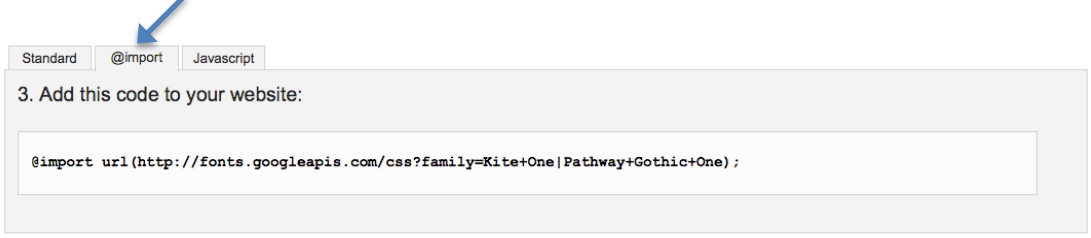
FILTER FONTS **ADD TO COLLECTION**



REVIEW **USE**

Once you've added all the fonts you want to use to a collection, you can review them (and delete from the collection if necessary) and once you're happy, you can head to the final stage to use them.

Once you're in the use stage, head straight down to step 3. Make sure that you select the @import tab because the system defaults to standard.



Do what it says on the tin and copy the code and paste it into your stylesheet. Remember, @import rules must go at the top before any of your own styles.

The final step is to integrate the fonts into your CSS. Google gives you the code to declare the font family. For example, I've added two fonts to my collection – Kite One and Pathway Gothic One. I'm going to set one for the body and one for the header:

```
body {
   background-color: #222222;
   color: #ffffff;
   font-family: 'Kite One', sans-serif;
}
header {
   background-color: #ffffff;
   color: #222222;
   font-family: 'Pathway Gothic One', sans-serif;
   font-size: 36px;
}
```

Too easy.

You'll find that Google fonts can really improve the presentation of your Web pages, but as I've said in a lecture, don't go crazy and clutter your pages with too many fonts. It is inadvisable to use more than three different fonts on any single page because you need consistency and don't want to create visual noise.

Smooth Scrolling

Look at the example page for this week. You'll see I've created a single HTML document with navigation links on the left. If you scroll down the page I want you to notice two things:

First, that the navigation box stays put. This is because I've positioned it using `position: fixed` and used the `top` and `left` properties to give it a bit of space away from the edge of the window.

Second, there are four `<h1>` tags used. Each one has an ID that corresponds with the `href` attribute of the links in the navigation box.

Navigation	Headings
`<a href="#home">home</a>`	`<h1 id="home">Home</h1>`
`<a href="#audio">audio</a>`	`<h1 id="audio">Audio</h1>`
`<a href="#video">video</a>`	`<h1 id="video">Video</h1>`
`<a href="#contact">contact</a`	`<h1 id="contact">Contact</h1>`

When an `href` value is something like #home or #video it refers to a location on the *same page* rather than the filename of another page. Clicking a link with such an `href` jumps to the location of the element with an ID that matches the `href`.

This is really handy for setting up links that are local to an individual page. The instantaneous jump however, is a bit visually harsh. Using a small amount of jQuery coding we can measure how far a particular element is from the top of the page and then scroll the page to that location.

This smooth scrolling requires us to do a few things:

1. Retrieve the `href` from the link that is clicked.
2. Find out how far down the page the element with the matching ID is.
3. Scroll the page to that point.

This is the code that makes it happen. We'll break it down and explain it in a moment:

```
$('nav ul li a').click(function(e) {
  e.preventDefault();
  target = $( $(this).attr('href') ).position().top;
  $('html,body').stop().animate({ scrollTop: target }, { duration:
  500});
});
```

As you can see, there's not much code involved, but it's more complex than what we've done before, so let's break it down.

```
$('nav ul li a').click(function(e) {
  e.preventDefault();
  target = $( $(this).attr('href') ).position().top;
  $('html,body').stop().animate({ scrollTop: target }, { duration:
  500});
});
```

The selector refers to the `<a>` tags that are nested inside of `<li>` tags inside of a `<ul>` tag that is inside of the `<nav>`. It simply reflects the HTML structure.

We're binding a click event to this selector, so that when one of these links is clicked, it will execute the function. Remember, a function is an action.

```
$('nav ul li a').click(function(e) {
  e.preventDefault();
  target = $( $(this).attr('href') ).position().top;
  $('html,body').stop().animate({ scrollTop: target }, { duration:
  500});
});
```

This line of code means prevent the default event from happening. In this instance, the default event of clicking a hyperlink is to move the page to another location, whether that's a location on the same page or another page entirely. This line stops the hyperlinks from behaving as they normally would.

```
$('nav ul li a').click(function(e) {
  e.preventDefault();
  target = $( $(this).attr('href') ).position().top;
  $('html,body').stop().animate({ scrollTop: target },{ duration:
  500});
});
```

Now this is a bit more complicated because there are several steps to this. Let's break it down:

```
$(this).attr('href')
```

The `.attr()` command allows jQuery to retrieve the value of an attribute (in this case the `href`) from a selector. Remember, the click event is bound to the `<a>` tags, each of which has an individual `href` value. Using `$(this)` jQuery is able to know the particular instance of `<a>` that has been clicked. Therefore, this line of code refers to the `href` of whichever `<a>` we click. Let's assume we've clicked the video link: the `href` value would therefore be `#video`.

```
$( $(this).attr('href') ).position().top;
```

Once we have the value of the relevant `href`, it forms our selector, for example:

```
$( '#video' ).position().top;
```

We can then use `.position().top` to tell us how far down the page the `<h1 id="video">` is. Now, we never know that number (measured in pixels), but our jQuery code does know it and that's what is important. If you really want to know, the `<h1 id="video">` is 1967.4332938964844 pixels from the top of the page.

```
target = $( $(this).attr('href') ).position().top;
```

`target` is what's called a variable. Variables are a staple part of programming. A variable is a method of storing information. In this case, I'm storing a number in a variable called `target`. That number is the distance of from `<h1 id="video">` the top of the page. We're not going to explore variables any further than this today. For now, just know that `target` becomes a way to refer to the position of `<h1 id="video">` on the page.

```
$('html,body').stop().animate({ scrollTop: target }, { duration: 500
});
```

The final section of the code is what makes the page scroll. We're using two selectors here – `html` and `body`. This is because some browsers refer to the page using `html` and others, `body`. If we want this to work across all browsers, then we need to include both.

Before we start animating the page, we use `.stop()` to kill any current page animation. This means that if the page is already scrolling to a location and we click another link, the page will stop where it is and start scrolling to the new location. If we don't use `.stop()` then the page will complete its current animation *before* starting the next one.

We looked at `.animate()` in week 4. You'll recall that it allows us to animate any CSS property that accepts a numerical value. In this instance we're animating the `scrollTop` property, which sets a distance from the top of the page to scroll. We're using our `target` variable as the value for this property.

If we put all this together and assume for a moment that we've clicked the contact link, this is what jQuery actually processes:

```
$('nav ul li a').click(function(e) {
  e.preventDefault();
  target = 2640.7833;
  $('html,body').stop().animate({ scrollTop: 2640.7833 }, {
  duration: 500});
});
```

Our variable `target` holds the numbers of pixels from the top that `<h1 id="contact">` is located, and then we animate `scrollTop` to that value – 2640.7833.

This is going to take time to sink in and for you to fully grasp. I strongly advise re-reading this document a few times and experimenting with the code yourselves. (You can copy the code from the example page.)