Day 4

This is CS50 for MBAs. Harvard Busines School. Spring 2015.

Cheng Gong

Table of Contents

Questions from Yesterday

- Suppose that a company does not want its employees visiting Facebook while at work, and so it blocks outbound Internet access to the site. Technically speaking, how might the company be implementing that restriction? How might a ~~clever~~ troublesome employee potentially circumvent the block?

  # The company might block any DNS requests for `www.facebook.com`, but an employee can just visit the IP address of Facebook directly if they know it, or change their DNS server in Network Preferences to something custom, like Google's easy-to-remember `8.8.8.8`. But now Google knows every site you visit, since you're asking it for DNS information …

  # The default ones that were there already are configured by DHCP, when you first connect to a network.

  # The company also just block packets to any IP address that belonged to Facebook, or even look inside a packet, since HTTP requests include a `Host:` field, and block packets they don't like.

  # A proxy is just some server that we might send all our traffic to, and then it forwards it along for us. Routers are proxies too. Another type of proxy is a VPN (virtual private network), which we can use to go around website blocking, since it has a completely encrypted connection. So our data, including all the headers and packets, are secure, up until it reaches our VPN, after which it is decrypted and

forwarded along to its actual destination. This keeps anything in between us and our VPN from reading our traffic.

    - We can use this to avoid people in public locations from seeing our data on a public network, or firewalls in other countries that might block certain websites.

  - The cost of this, though, is latency, or time between requests and responses, since we need to send our traffic to another server first.
  - Tor is a similar software where all the computers running it form a network, and routes data randomly among them, so it would be extremely difficult to trace the origin, or even destination, of data on the network.
    - Not too long ago there was a bomb threat made on campus by email, but the student, though he was using Tor, was among only a few people on campus connected to the network, which made him easier to find.

- When should GET be used in an HTTP request?
  - Sometimes, when we submit a query to a webpage, we do want the input to be in the URL and saved, like for images of cats, so we can bookmark it or link to it more easily, instead of having to type it in every time.
- When should POST be used in an HTTP request?
  - POST is for data to remain private, like passwords or credit card numbers, or also when sending large files that don't fit in the URL of a GET request.
- What's the difference between a char and a short (both have 16 bits) and similarly between int/float and long/double?
  - A char has no negative sign, and is meant to store characters, while a short has a sign bit in its leftmost bit, and used to store integers that can fit in the 15 bits left.
  - An int is an integer, with no decimal place, while a float is a real number which does.
  - A long is an integer with more (twice as many, to be precise) bits, and a double is a float that also has twice as many bits.
  - We have these data types because we want the flexibility of being able to choose between saving space and not having as many empty bits, or having larger numbers or more precision. With thousands or millions of numbers, that difference is much greater.
- Is the 2011 Citibank hack related to the dangers of using GET requests when they should be structured as POST?

  - David had trouble finding sources that detailed the hack, but the gist is as follows:
  - The bank's website, at the time, used to take you to a URL that looked something like this, after you logged in:

    ```
    http://www.citibank.com/balance?account=123
    ```

  - But now all we would need to do is to change the URL to any account number we want, since the website didn't check whether you were actually logged into that account, as it should have. Sometimes really simple programming errors can lead to really large security holes.

- Why do companies have multiple IP addresses? For example, why did Google show ten or so IP addresses opposed to just one?
  - Google has multiple servers, for redundancy and latency, and having all those IP addresses just allows for any of them to be contacted. Facebook only has one IP address, but all of their traffic might just be automatically balanced by that one server to others.
- Can we showcase some of the Scratch projects? I'm curious what my classmates did! Thanks!
  - We have a gallery[1] here, if you're interested!
- I understand why the Patriot's failure happened. However, how did they fix it? Did they change the time unit from 1/10 of second to something with a terminary binary expansion, e.g. 1/4 of second?
  - David looked around and there's not much information, but it seems that though there was a function that convert clock time to floats relatively precisely, it was only being used some of the time, so operations on imprecise and precise values created more errors than there otherwise would be.
- IP addresses only correspond to domain names, or something like "www.citibank.com", or "cs50.harvard.edu".

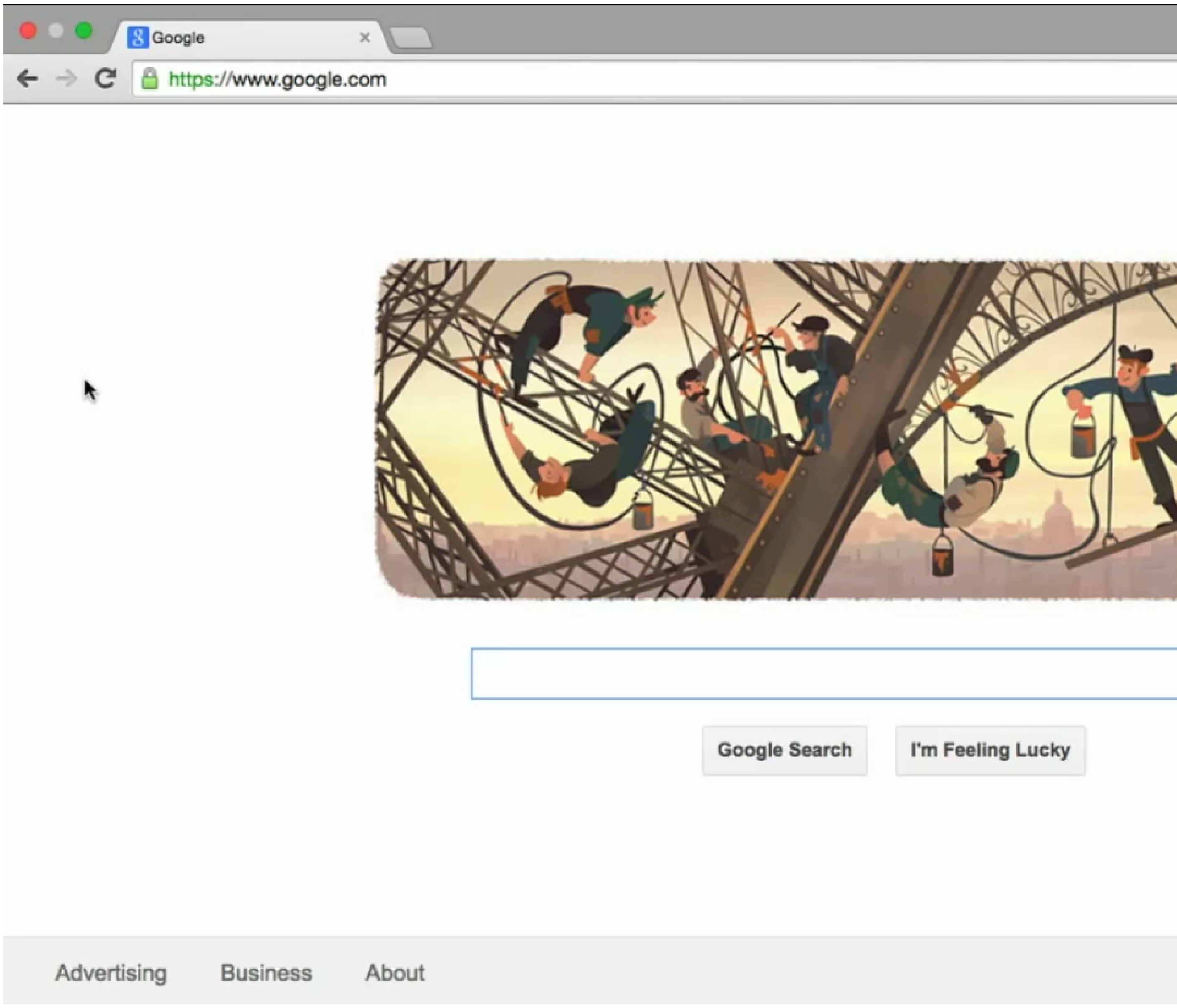
[1] https://scratch.mit.edu/studios/1070152/

HTML5

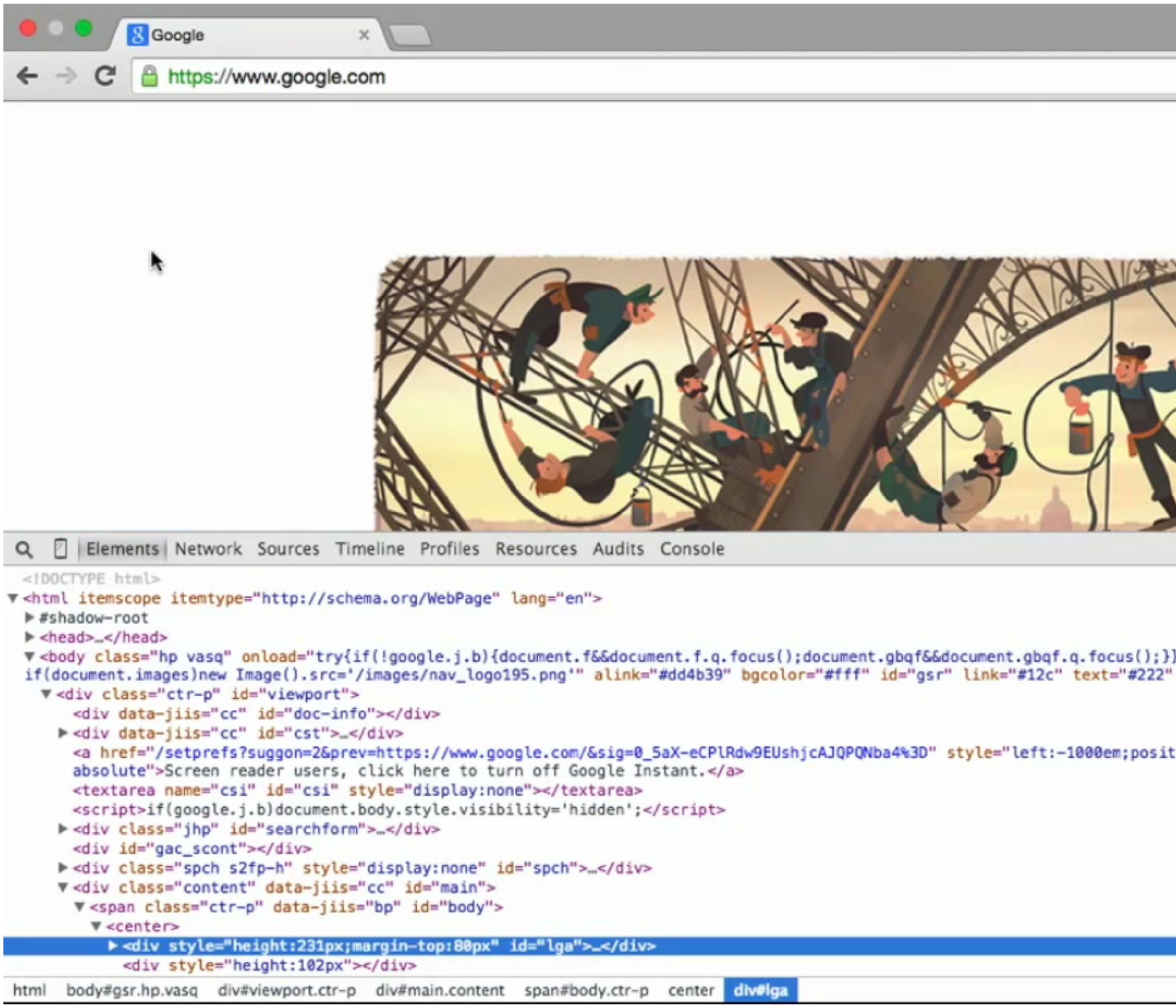
- HTML5 is the fifth version of a language used to markup webpages, and we'll manually connect to Google from the command line like last time:

```
% telnet www.google.com 80
Trying 173.194.219.99...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com
```

- We get a lot of code as text, but if we visit the page in the browser, we'll see the usual sparse text, buttons, input boxes, images, and links:

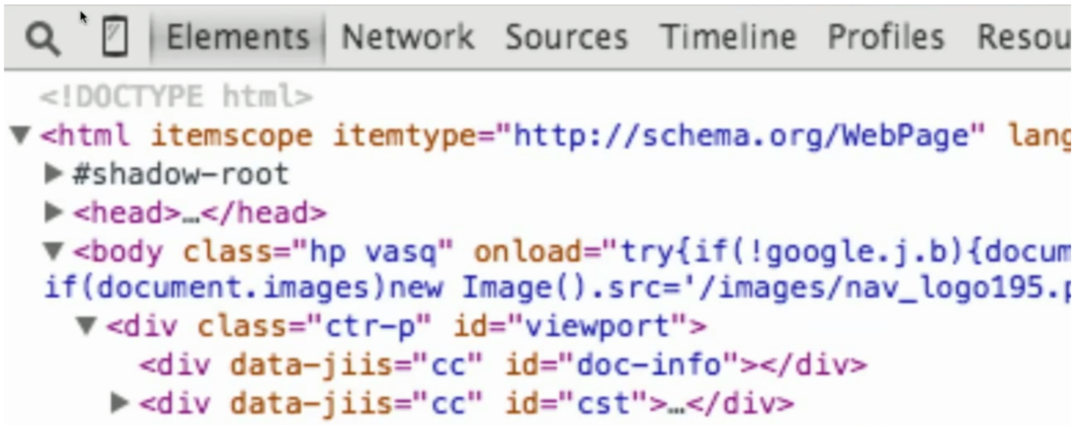


- We can right-click a blank space, and click Inspect Element, to get this feature:



```
<!DOCTYPE html>
▼<html itemscope itemtype="http://schema.org/WebPage" lang="en">
  ▶#shadow-root
  ▶<head>…</head>
  ▼<body class="hp vasq" onload="try{if(!google.j.b){document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}}
  if(document.images)new Image().src='/images/nav_logo195.png'" alink="#dd4b39" bgcolor="#fff" id="gsr" link="#12c" text="#222"
    ▼<div class="ctr-p" id="viewport">
       <div data-jiis="cc" id="doc-info"></div>
      ▶<div data-jiis="cc" id="cst">…</div>
       <a href="/setprefs?suggon=2&prev=https://www.google.com/&sig=0_5aX-eCPlRdw9EUshjcAJQPQNba4%3D" style="left:-1000em;posit
       absolute">Screen reader users, click here to turn off Google Instant.</a>
       <textarea name="csi" id="csi" style="display:none"></textarea>
       <script>if(google.j.b)document.body.style.visibility='hidden';</script>
      ▶<div class="jhp" id="searchform">…</div>
       <div id="gac_scont"></div>
      ▶<div class="spch s2fp-h" style="display:none" id="spch">…</div>
      ▼<div class="content" data-jiis="cc" id="main">
        ▼<span class="ctr-p" data-jiis="bp" id="body">
          ▼<center>
            ▶<div style="height:231px;margin-top:80px" id="lga">…</div>
             <div style="height:102px"></div>
```

- Today we'll look at the Elements tab, and all the code on the left is just a pretty-printed version of the code that we saw when we ran our `telnet` program.
- We notice that the code is now colored, by some pattern, and not just black and white and indented nicely with triangles for blocks we can expand and collapse.
- The version we got earlier was all black and white with no spacing, to save on bandwidth (and therefore costs) and time, but the computer can read it just the same. The colors and indentation are just for us human.
- If we start looking at the actual code, now, we see that the first word in color is something familiar, `<html`:



```
<!DOCTYPE html>
▼<html itemscope itemtype="http://schema.org/WebPage" lang
  ▶#shadow-root
  ▶<head>…</head>
  ▼<body class="hp vasq" onload="try{if(!google.j.b){docum
  if(document.images)new Image().src='/images/nav_logo195.
    ▼<div class="ctr-p" id="viewport">
        <div data-jiis="cc" id="doc-info"></div>
      ▶<div data-jiis="cc" id="cst">…</div>
```

  - We'll ignore that first gray line for now, and look at the second line. The `<` means that we're starting a new **tag**, which is just something that tells the browser that there's something special ahead. In this case, `<html>` (the `>` is further to the right, and we can ignore everything else within for now) tells the computer that we're starting a new HTML document.

  - The next purple tags we see are `<head>` and `</head>`, which is telling the browser that whatever is in between them should go in the `head` of the webpage. You can think of the `head` as the stuff at the top, like the title, that blue "g" icon, and other stuff that's not displayed. The next chunk, `<body>`, is everything that you see in the actual page.

- So HTML isn't really a programming language in that it has anything fancy like variables or loops, but rather a markup language that describes pieces of a webpage.

- Let's look at a really simple webpage:

```
<!DOCTYPE html>

<html>
    <head>
        <title>hello, world</title>
    </head>
    <body>
        hello, world
    </body>
</html>
```

  - All we needed to write this was Notepad or TextEdit.

  # The first line is required by the HTML standard, and just says the document type is HTML.

  # After that, we notice the `<html>` as usual, but now at the bottom we see `</html>`, which is telling the browser that the page has ended.

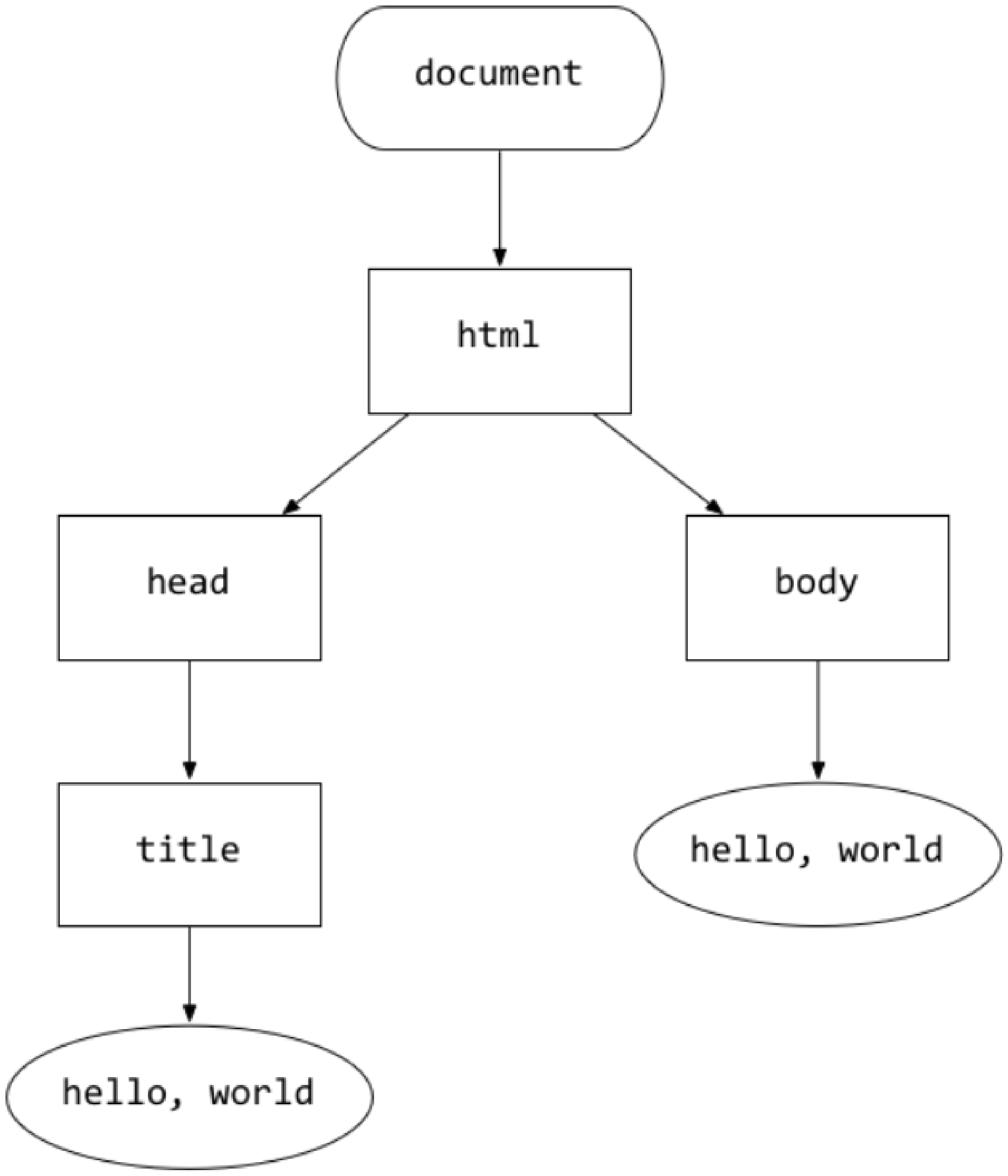
  # We'll start calling everything within an open (start) and close (end) tag an element.

  # Notice how everything inside an element is indented, and that every tag that's opened is closed.

  # `<title>` and `</title>` are on the same line because `hello, world` was just one line, but it makes no difference to the computer.

  # Also notice how tags are closed symmetrically, as in `<title>` is closed before `<head>`, since it was opened inside `<head>`.

- We can draw this page as a tree:

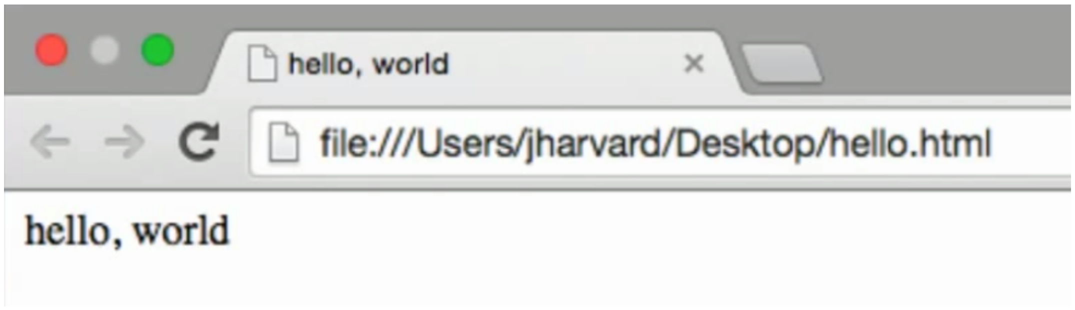


# The shapes don't really matter, and the root element is called the `document`, which we can think of as everything in the file.

# Then within the `document`, we have the `html` element, and now we see how the children of that are labeled `head` and `body`, matching what's in the source code.

# The `head` element has the `title` element, with the text `hello, world` inside, and since that's actual text, we've made it an oval rather than a rectangle.

- We'll save this file as `hello.html` to the Desktop, and open it:

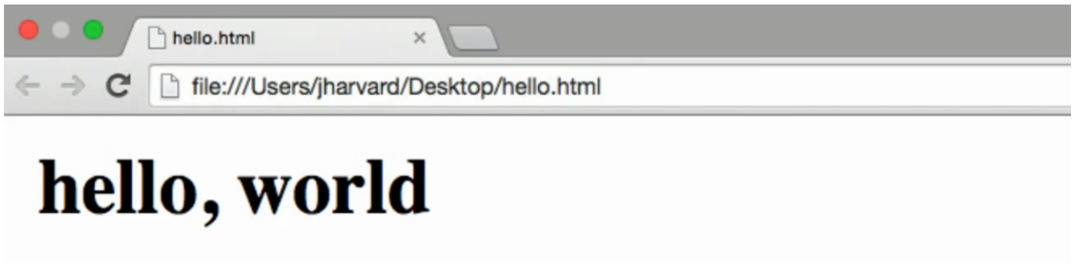


hello, world

# Notice the title is in the top of the tab, and our text is in the body.

- We can add tabs to make our text bold:

```
<!DOCTYPE html>

<html>
    <head>
        <title>hello, world</title>
    </head>
    <body>
        <b>hello, world</b>
    </body>
</html>
```

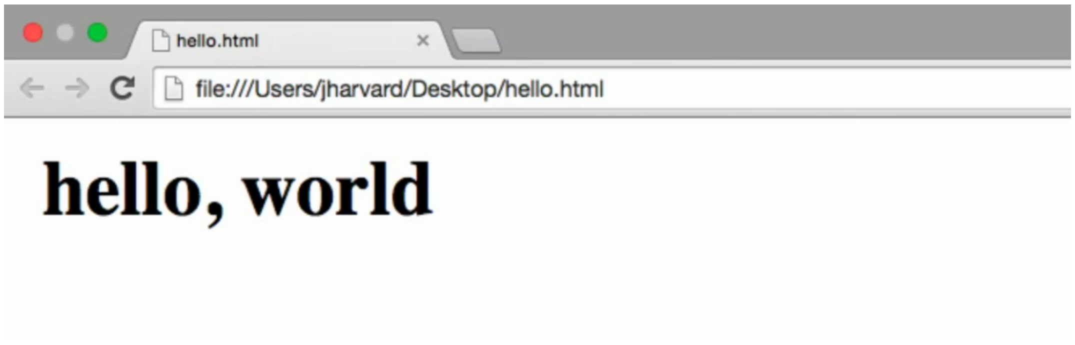


hello, world

# (The text is bigger because David zoomed in more in the browser, but it's still the same actual size.)

- If we remove the title, we see this:

```
<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        <b>hello, world</b>
    </body>
</html>
```

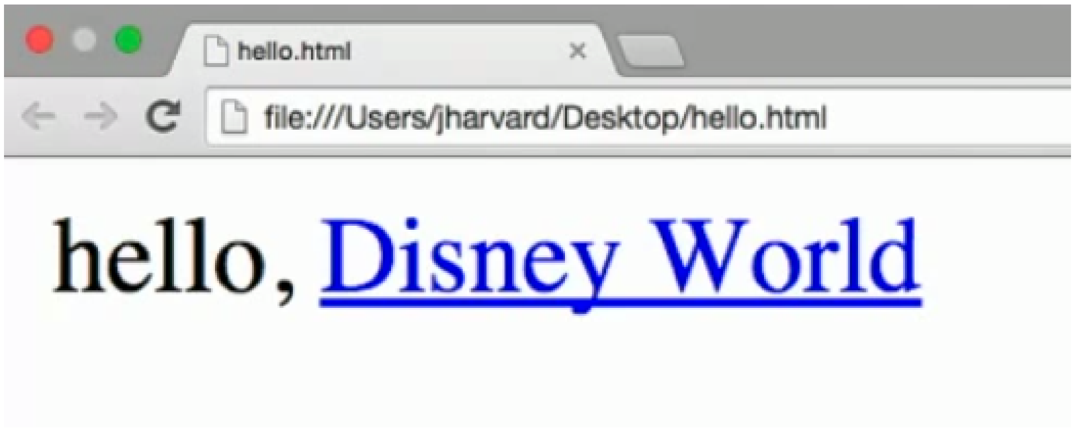


hello, world

# So Chrome chooses to display the name of the file, instead of nothing, but that was a decision made by the browser and not us, since we didn't specify anything.

- We can make a link, too:

```
<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        hello, <a href="http://www.disney.com/">Disney World</a>
    </body>
</html>
```



# Every time we've made a change, we had to save the file from TextEdit, and refresh in Chrome.

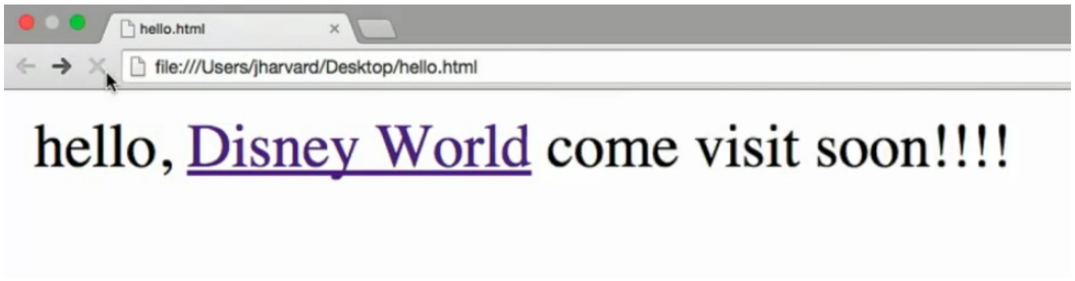
# A link has historically been called an anchor, hence the `<a>` tag, but we also pass in, as an attribute, the `href`, or hyper-reference (also an ancient word for the actual link) to the place we want the link to actually go.

# Notice the fundamental syntax of the `href` portion, where the attribute name is to the left of an equals sign, with the value to the right.

# If we left out the `http://` part of the link, or even the ending tag of `</a>`, it would still work, but only because browsers automatically guess what it thinks is missing. Only if we use complete syntax can we be sure that the page will be displayed as we intended. (Well, browsers actually all display things slightly differently, and making web pages that look the same on all browsers (cross-browser compatibility) has actually been a pain for the longest time.)

- What if we wanted to say something else a few lines down? Let's try this:

```
<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        hello, <a href="http://www.disney.com/">Disney World</a>

        come visit soon!!!!
    </body>
</html>
```

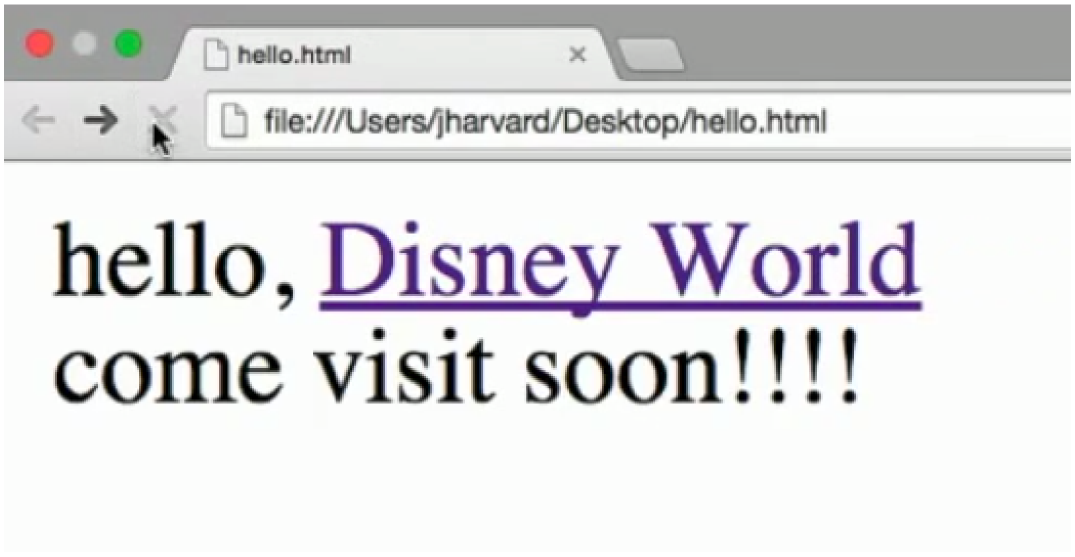


hello, Disney World come visit soon!!!!

# Hmm, not what we intended, but this makes sense because we have all those spaces and lines, and browsers will only place a maximum of one space between text.

- Thankfully, we can manually add a line break:

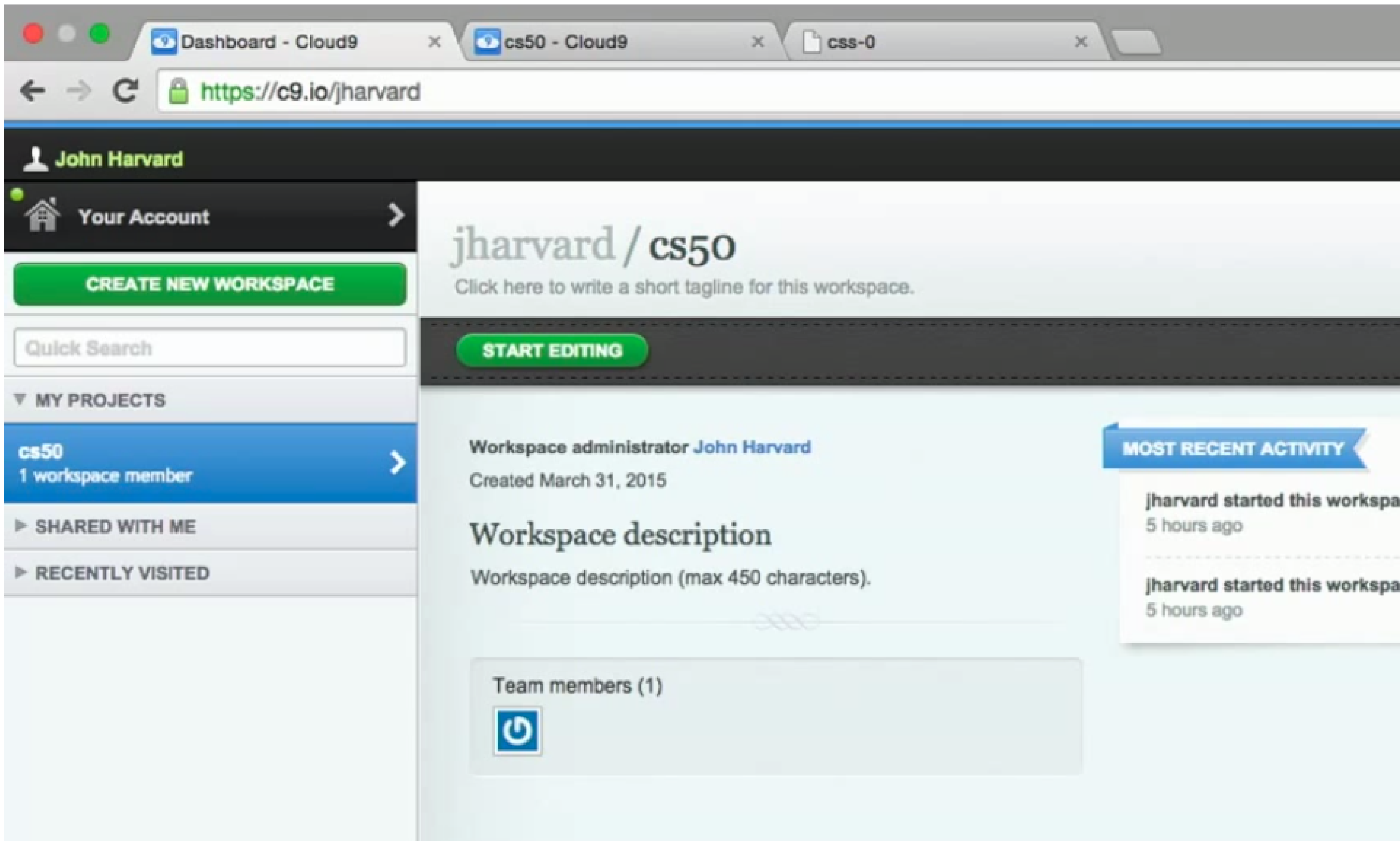
```
<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        hello, <a href="http://www.disney.com/">Disney World</a>
        <br/>
        come visit soon!!!!
    </body>
</html>
```



hello, Disney World
come visit soon!!!!

  # And notice how we closed our `<br>`, line break, tag. This is because a line break is an element that can't have anything inside, and is either there or not, so we use a slightly different syntax where the open and close tags are combined as you see there.

- But this web page only lives on our laptop, and not online. Today, there are web-based web development environments, and that's just a fancy way of saying there are web apps that you can code in, with all the setup and configuration done for you.
- We'll use one such service, Cloud9, where we've already signed up and created a workspace (the instructions for you for which are in Project 1):



Workspace administrator John Harvard
Created March 31, 2015

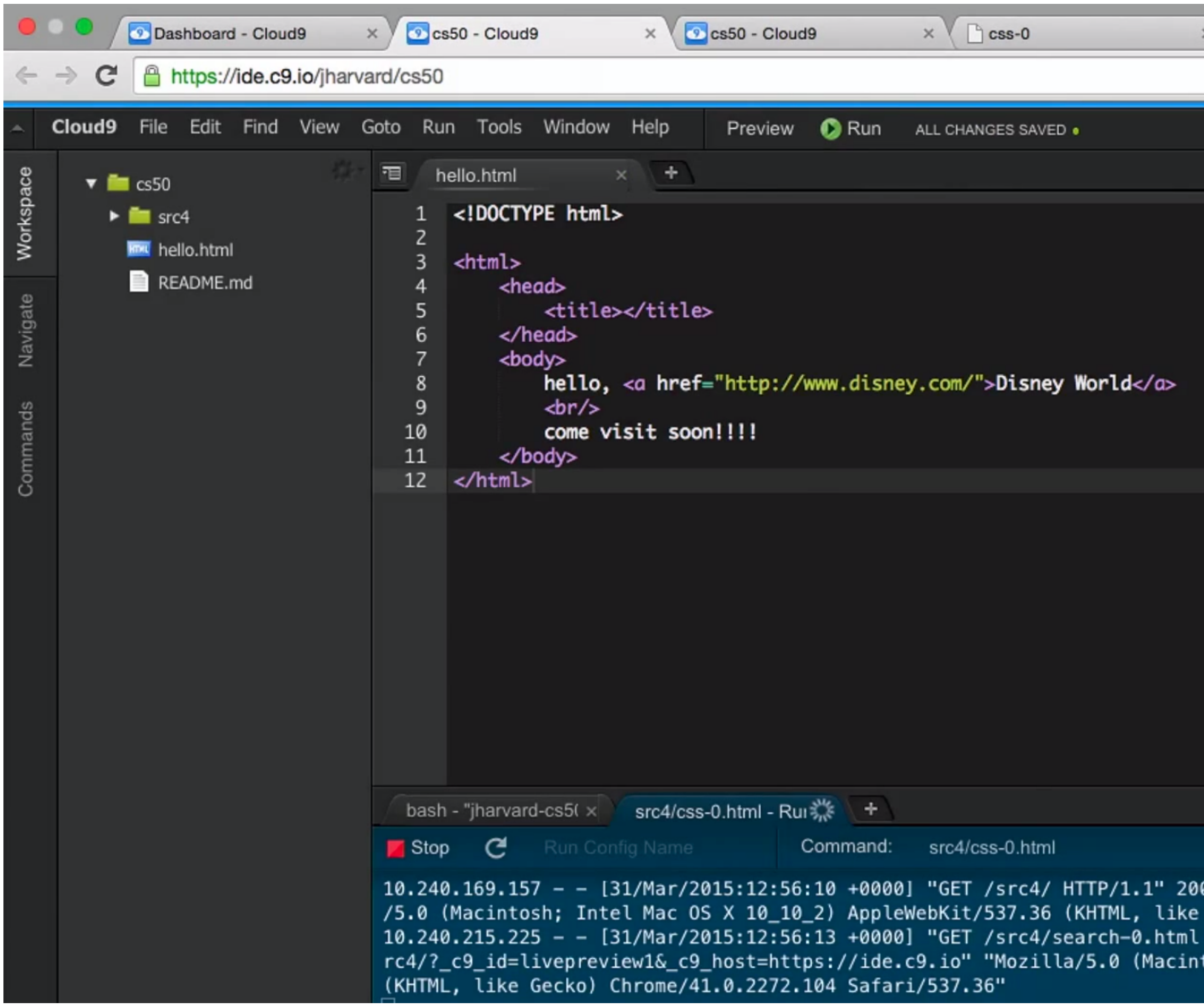
Workspace description

Workspace description (max 450 characters).

jharvard started this workspa
5 hours ago

jharvard started this workspa
5 hours ago

- We'll press the `Start editing` button, and paste our code in:

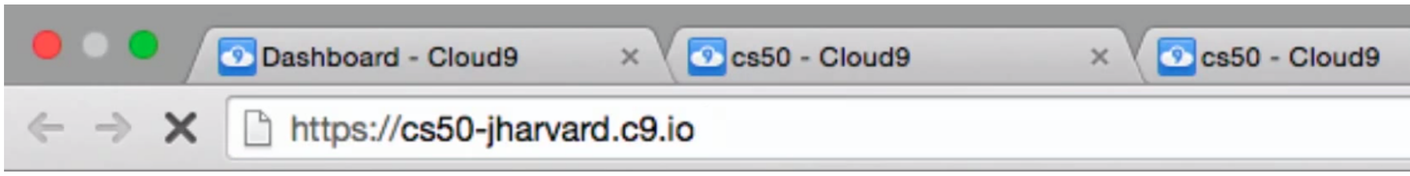


```
<!DOCTYPE html>

<html>
    <head>
        <title></title>
    </head>
    <body>
        hello, <a href="http://www.disney.com/">Disney World</a>
        <br/>
        come visit soon!!!!
    </body>
</html>
```

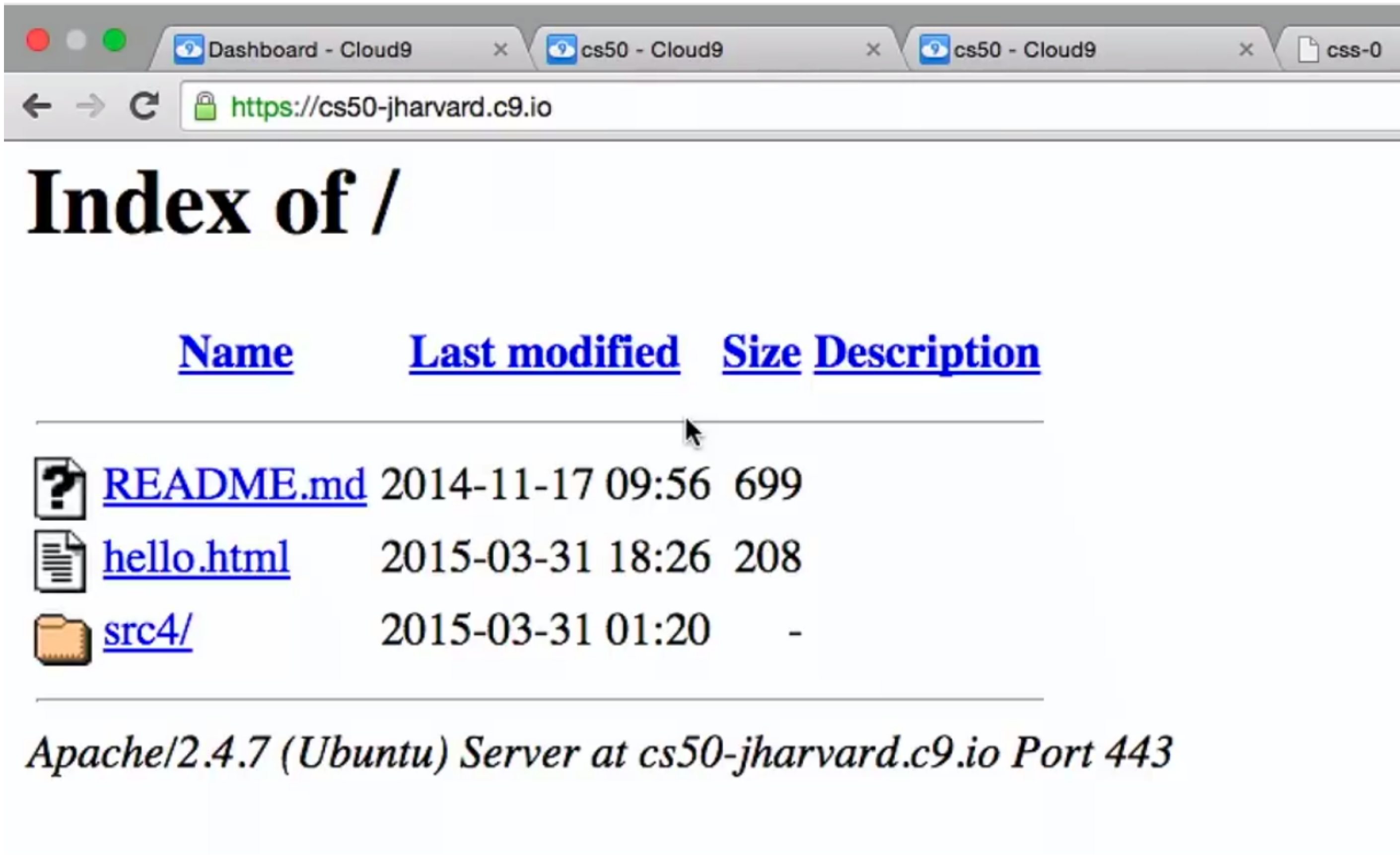
```
10.240.169.157 - - [31/Mar/2015:12:56:10 +0000] "GET /src4/ HTTP/1.1" 20
/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like
10.240.215.225 - - [31/Mar/2015:12:56:13 +0000] "GET /src4/search-0.html
rc4/?_c9_id=livepreview1&_c9_host=https://ide.c9.io" "Mozilla/5.0 (Macin
(KHTML, like Gecko) Chrome/41.0.2272.104 Safari/537.36"
```

- What we've done here is connect to one of Cloud9's servers, somewhere in the cloud, and save our `hello.html` there, with the built-in text editor that we can access in our browser.

- When we sign up for an account, we also get assigned a nice URL for our server, that looks something like this:



- And if we go to it, we get a page that's just a listing of our files on our server:

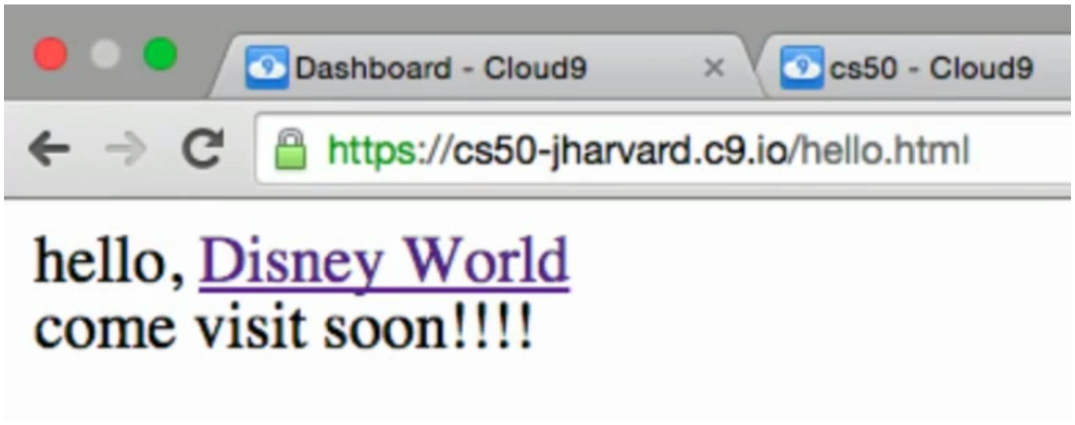


Index of /

Name	Last modified	Size	Description
README.md	2014-11-17 09:56	699	
hello.html	2015-03-31 18:26	208	
src4/	2015-03-31 01:20	-	

Apache/2.4.7 (Ubuntu) Server at cs50-jharvard.c9.io Port 443

- If we clicked `hello.html`, we see what we expected, and now it's actually live on the internet:



hello, Disney World
come visit soon!!!!

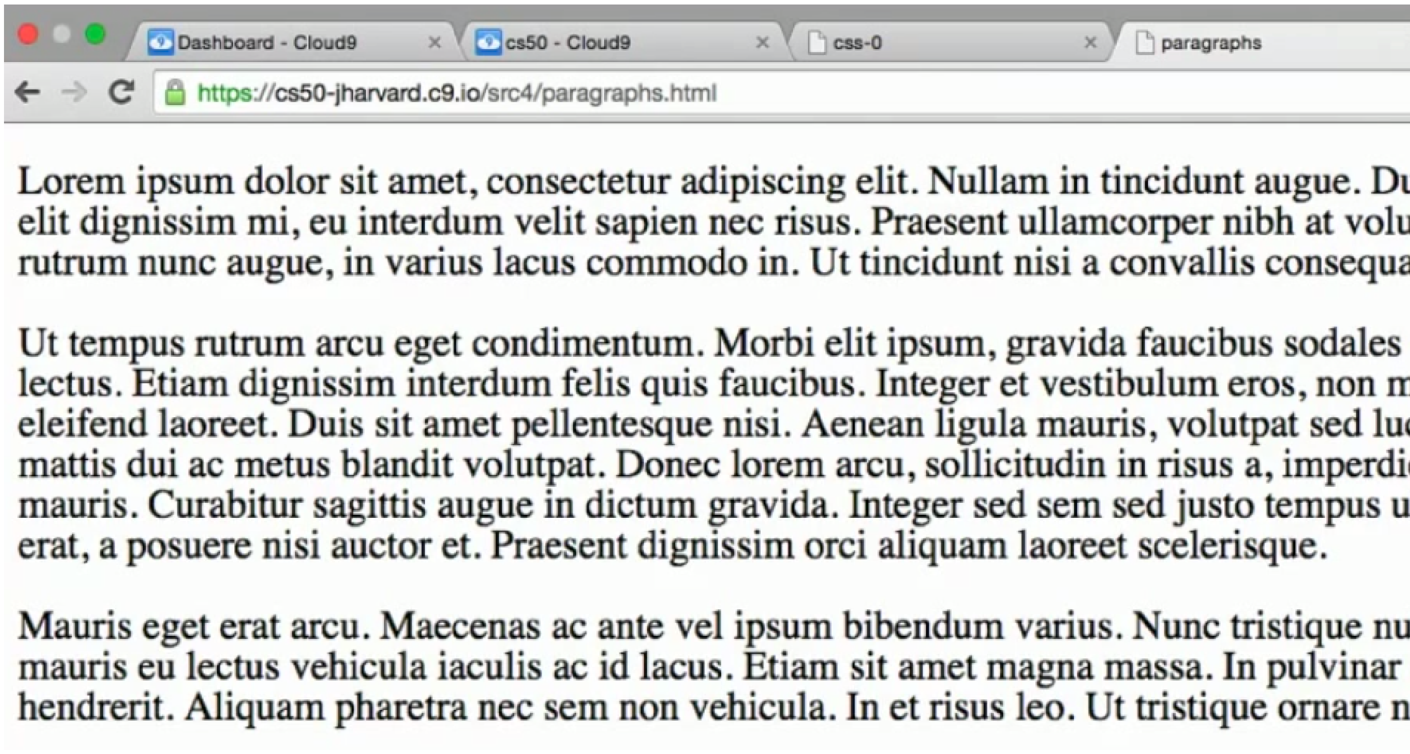
- We'll go through a few of the examples that we've put up here[2].

[2] http://cdn.cs50.net/2015/mba/classes/4/src4/

- First is `paragraphs.html`[3]:

[3] http://cdn.cs50.net/2015/mba/classes/4/src4/paragraphs.html

```
<!DOCTYPE html>

<!--

paragraphs.html

David J. Malan
malan@harvard.edu

Displays some text.

Demonstrates paragraphs.

-->

<html>
    <head>
        <title>paragraphs</title>
    </head>
    <body>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus,
 erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent
 ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum
 nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis
 consequat. Fusce sed pulvinar nulla.
        </p>
        <p>
            Ut tempus rutrum arcu eget condimentum. Morbi elit ipsum,
 gravida faucibus sodales quis, varius at mi. Suspendisse id viverra
 lectus. Etiam dignissim interdum felis quis faucibus. Integer et
 vestibulum eros, non malesuada felis. Pellentesque porttitor eleifend
 laoreet. Duis sit amet pellentesque nisi. Aenean ligula mauris, volutpat
 sed luctus in, consectetur id turpis. Phasellus mattis dui ac metus
 blandit volutpat. Donec lorem arcu, sollicitudin in risus a, imperdiet
 condimentum augue. Ut at facilisis mauris. Curabitur sagittis augue in
 dictum gravida. Integer sed sem sed justo tempus ultrices eu non magna.
 Phasellus semper eros erat, a posuere nisi auctor et. Praesent dignissim
 orci aliquam laoreet scelerisque.
        </p>
        <p>
            Mauris eget erat arcu. Maecenas ac ante vel ipsum bibendum
 varius. Nunc tristique nulla eget tincidunt molestie. Morbi sed mauris
 eu lectus vehicula iaculis ac id lacus. Etiam sit amet magna massa. In
 pulvinar sapien ac mi ultrices, quis consequat nisl hendrerit. Aliquam
 pharetra nec sem non vehicula. In et risus leo. Ut tristique ornare nisl
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam in tincidunt augue. Du
elit dignissim mi, eu interdum velit sapien nec risus. Praesent ullamcorper nibh at volu
rutrum nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis consequa

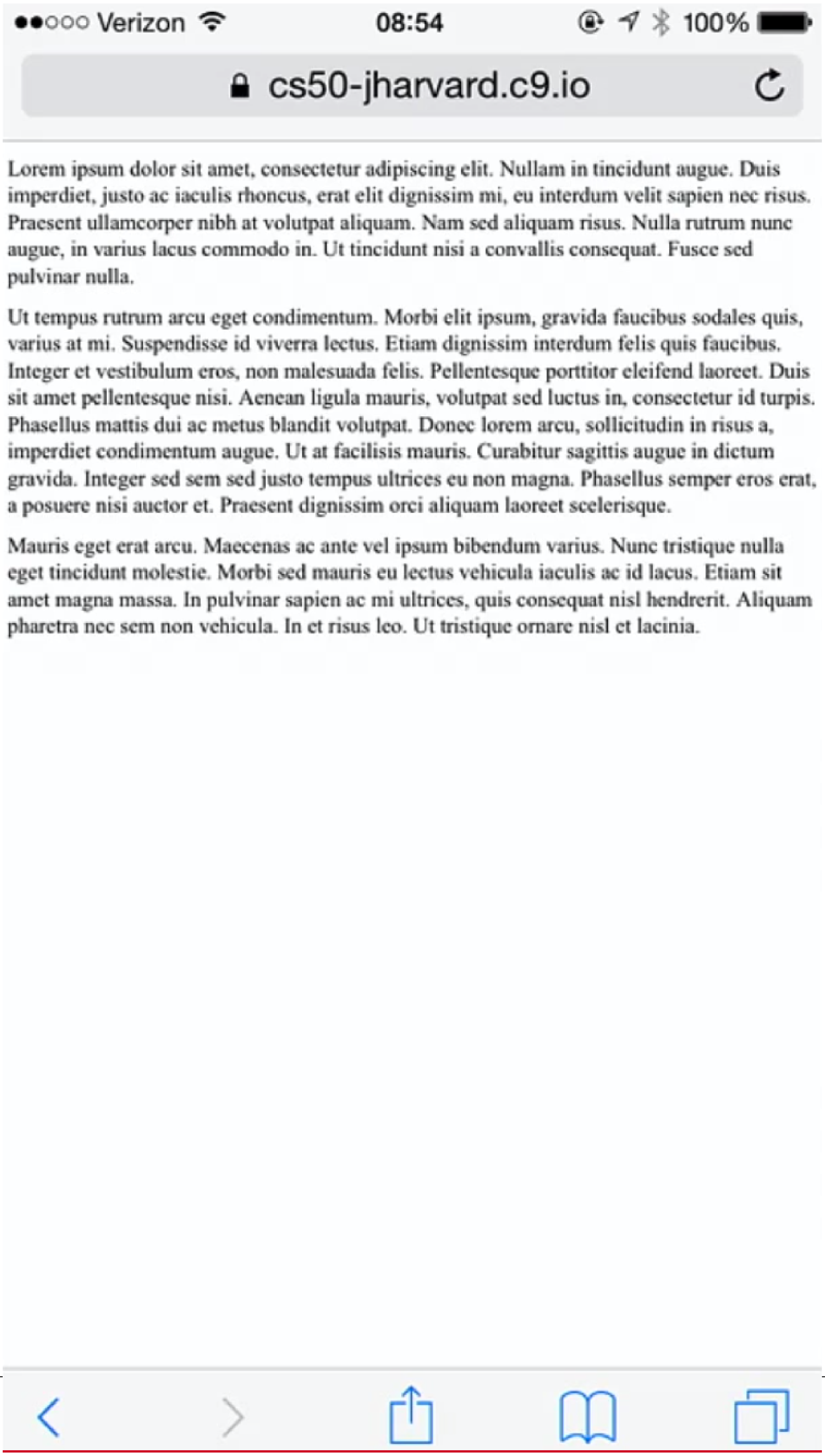
Ut tempus rutrum arcu eget condimentum. Morbi elit ipsum, gravida faucibus sodales
lectus. Etiam dignissim interdum felis quis faucibus. Integer et vestibulum eros, non m
eleifend laoreet. Duis sit amet pellentesque nisi. Aenean ligula mauris, volutpat sed luc
mattis dui ac metus blandit volutpat. Donec lorem arcu, sollicitudin in risus a, imperdi
mauris. Curabitur sagittis augue in dictum gravida. Integer sed sem sed justo tempus u
erat, a posuere nisi auctor et. Praesent dignissim orci aliquam laoreet scelerisque.

Mauris eget erat arcu. Maecenas ac ante vel ipsum bibendum varius. Nunc tristique nu
mauris eu lectus vehicula iaculis ac id lacus. Etiam sit amet magna massa. In pulvinar
hendrerit. Aliquam pharetra nec sem non vehicula. In et risus leo. Ut tristique ornare n

  # Turns out, on line 3, the `<!--` marks a comment, which is something about the source code that isn't displayed.

  # The new tag around all that text, `<p>`, marks paragraphs.

- Turns out, to make a website mobile-friendly, we only need another line of code.
- The original `paragraphs.html` looked like this:



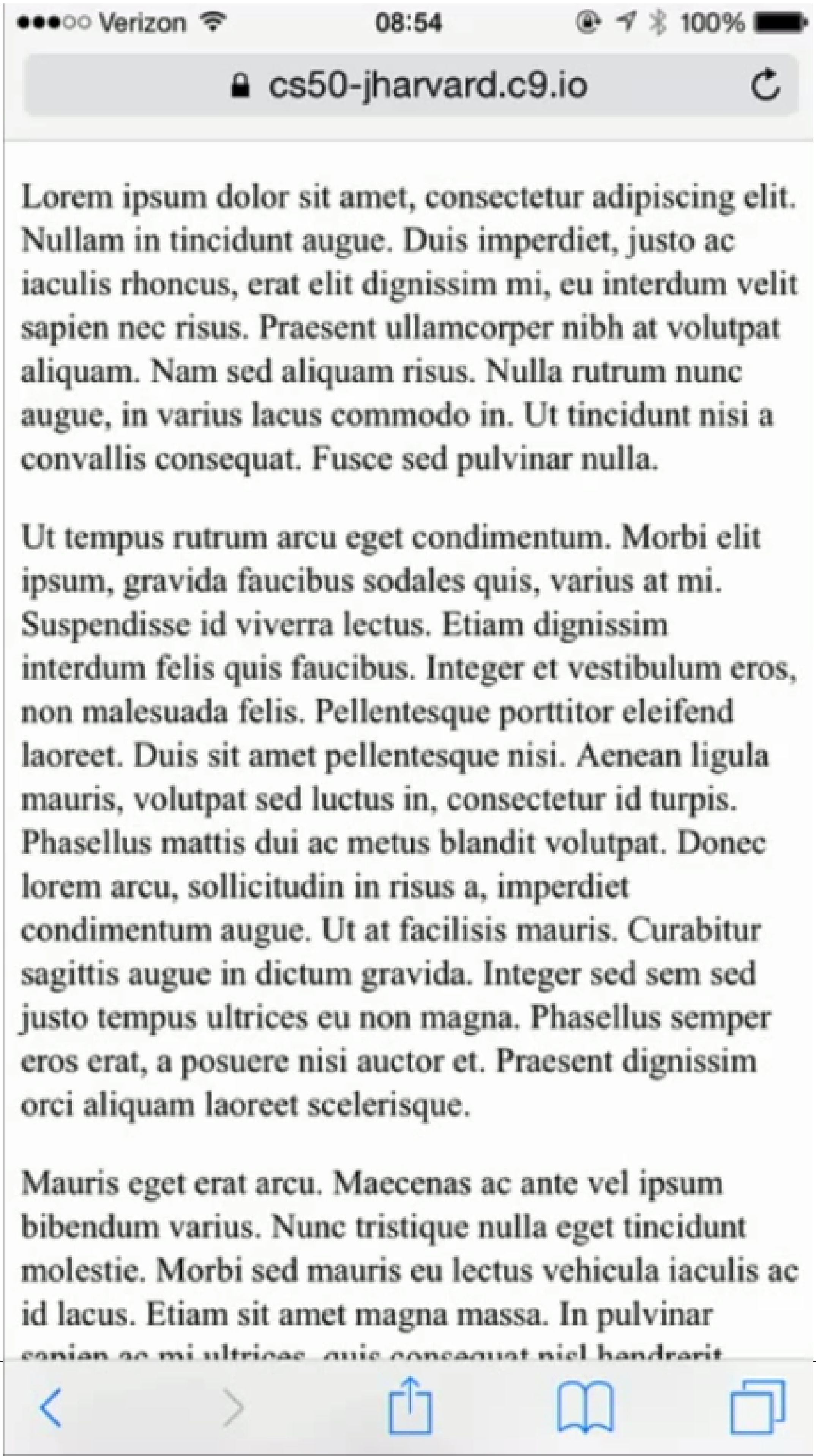
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus, erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis consequat. Fusce sed pulvinar nulla.

Ut tempus rutrum arcu eget condimentum. Morbi elit ipsum, gravida faucibus sodales quis, varius at mi. Suspendisse id viverra lectus. Etiam dignissim interdum felis quis faucibus. Integer et vestibulum eros, non malesuada felis. Pellentesque porttitor eleifend laoreet. Duis sit amet pellentesque nisi. Aenean ligula mauris, volutpat sed luctus in, consectetur id turpis. Phasellus mattis dui ac metus blandit volutpat. Donec lorem arcu, sollicitudin in risus a, imperdiet condimentum augue. Ut at facilisis mauris. Curabitur sagittis augue in dictum gravida. Integer sed sem sed justo tempus ultrices eu non magna. Phasellus semper eros erat, a posuere nisi auctor et. Praesent dignissim orci aliquam laoreet scelerisque.

Mauris eget erat arcu. Maecenas ac ante vel ipsum bibendum varius. Nunc tristique nulla eget tincidunt molestie. Morbi sed mauris eu lectus vehicula iaculis ac id lacus. Etiam sit amet magna massa. In pulvinar sapien ac mi ultrices, quis consequat nisl hendrerit. Aliquam pharetra nec sem non vehicula. In et risus leo. Ut tristique ornare nisl et lacinia.

- The text is a bit small, so let’s open `responsive.html`[4]:

[4] http://cdn.cs50.net/2015/mba/classes/4/src4/responsive.html



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus, erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis consequat. Fusce sed pulvinar nulla.

Ut tempus rutrum arcu eget condimentum. Morbi elit ipsum, gravida faucibus sodales quis, varius at mi. Suspendisse id viverra lectus. Etiam dignissim interdum felis quis faucibus. Integer et vestibulum eros, non malesuada felis. Pellentesque porttitor eleifend laoreet. Duis sit amet pellentesque nisi. Aenean ligula mauris, volutpat sed luctus in, consectetur id turpis. Phasellus mattis dui ac metus blandit volutpat. Donec lorem arcu, sollicitudin in risus a, imperdiet condimentum augue. Ut at facilisis mauris. Curabitur sagittis augue in dictum gravida. Integer sed sem sed justo tempus ultrices eu non magna. Phasellus semper eros erat, a posuere nisi auctor et. Praesent dignissim orci aliquam laoreet scelerisque.

Mauris eget erat arcu. Maecenas ac ante vel ipsum bibendum varius. Nunc tristique nulla eget tincidunt molestie. Morbi sed mauris eu lectus vehicula iaculis ac id lacus. Etiam sit amet magna massa. In pulvinar sapien ac mi ultrices, quis consequat nisl hendrerit

- "Responsive" is just a buzzword for websites that "respond" to different device sizes, and if we looked at the source code:

```
<!DOCTYPE html>

<!--

responsive.html

David J. Malan
malan@harvard.edu

Displays some text.

Demonstrates responsive layout.

-->

<html>
    <head>
        <meta name="viewport" content="width=device-width, initial-
scale=1.0"/>
        <title>responsive</title>
    </head>
    <body>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 Nullam in tincidunt augue. Duis imperdiet, justo ac iaculis rhoncus,
 erat elit dignissim mi, eu interdum velit sapien nec risus. Praesent
 ullamcorper nibh at volutpat aliquam. Nam sed aliquam risus. Nulla rutrum
 nunc augue, in varius lacus commodo in. Ut tincidunt nisi a convallis
 consequat. Fusce sed pulvinar nulla.
        </p>
...
```

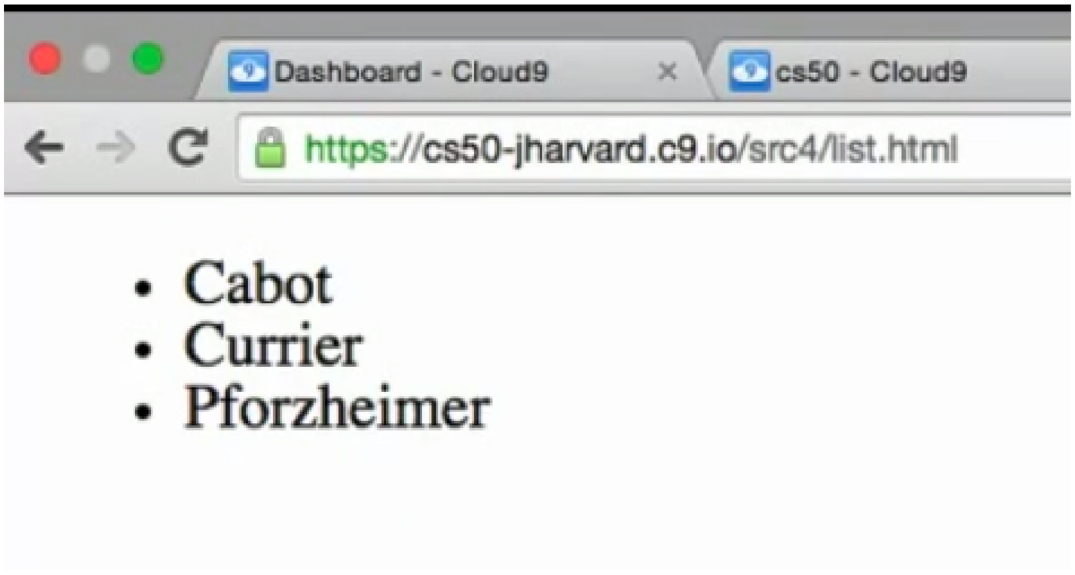
# The new tag on line 18, `<meta>`, has an attribute called `name` which is `viewport`, kind of cryptic, but the next attribute, `content`, has something inside called `width=device-width, initial-scale=1.0`, which is just scaling the page to 100%, of the width of the device.

  # And all of these standard HTML values and descriptors can be found by looking in a book or on Google for what you want to do. There's no way for anyone to know all of these tags, and most everyone looks up what they need as they need it. The important part is the syntax, with the tag name, an attribute, an `=`, and the values in the correct place and order.

- Let's look at open `list.html`[5]:

```
<!DOCTYPE html>

<!--

list.html

David J. Malan
malan@harvard.edu

Displays houses in the Quad.

Demonstrates (unordered) lists.

-->

<html>
    <head>
        <title>list</title>
    </head>
    <body>
        <ul>
            <li>Cabot</li>
            <li>Currier</li>
            <li>Pforzheimer</li>
        </ul>
    </body>
</html>
```

[5] http://cdn.cs50.net/2015/mba/classes/4/src4/list.html

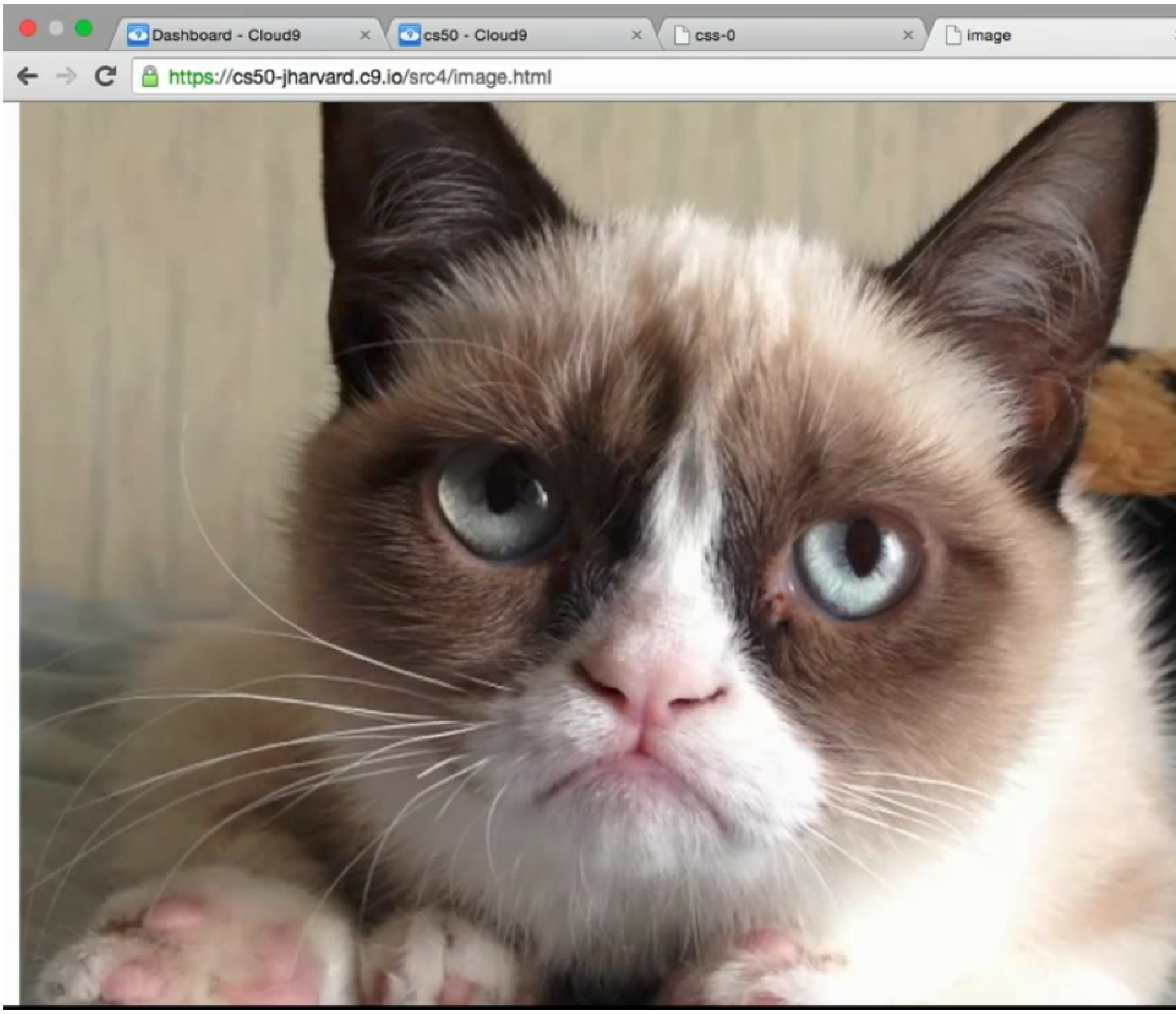


- Cabot
- Currier
- Pforzheimer

  # `<ul>` stands for unordered list, which means there will be bullet points for each `<li>`, or list item, and if we wanted a numbered list, we could use an `<ol>`, an ordered list.

- Let's look at `image.html` [6]:

[6] http://cdn.cs50.net/2015/mba/classes/4/src4/image.html

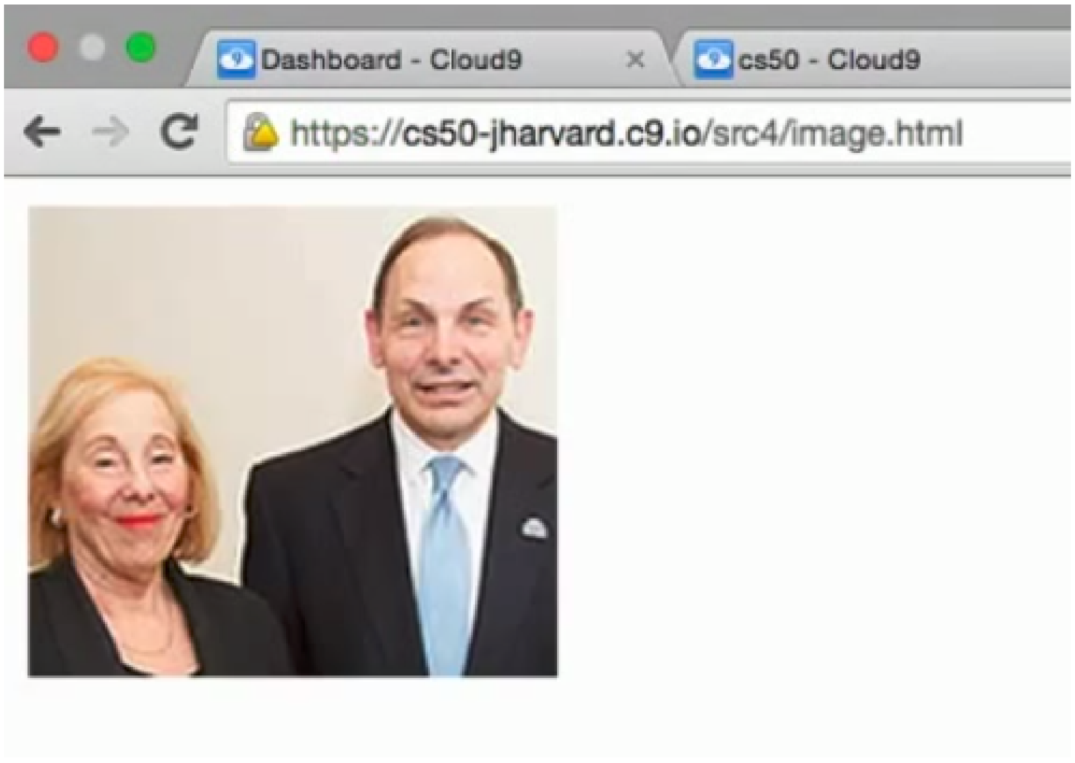


- All we need to do is to upload or download a picture to our server, and refer to it in HTML like this:

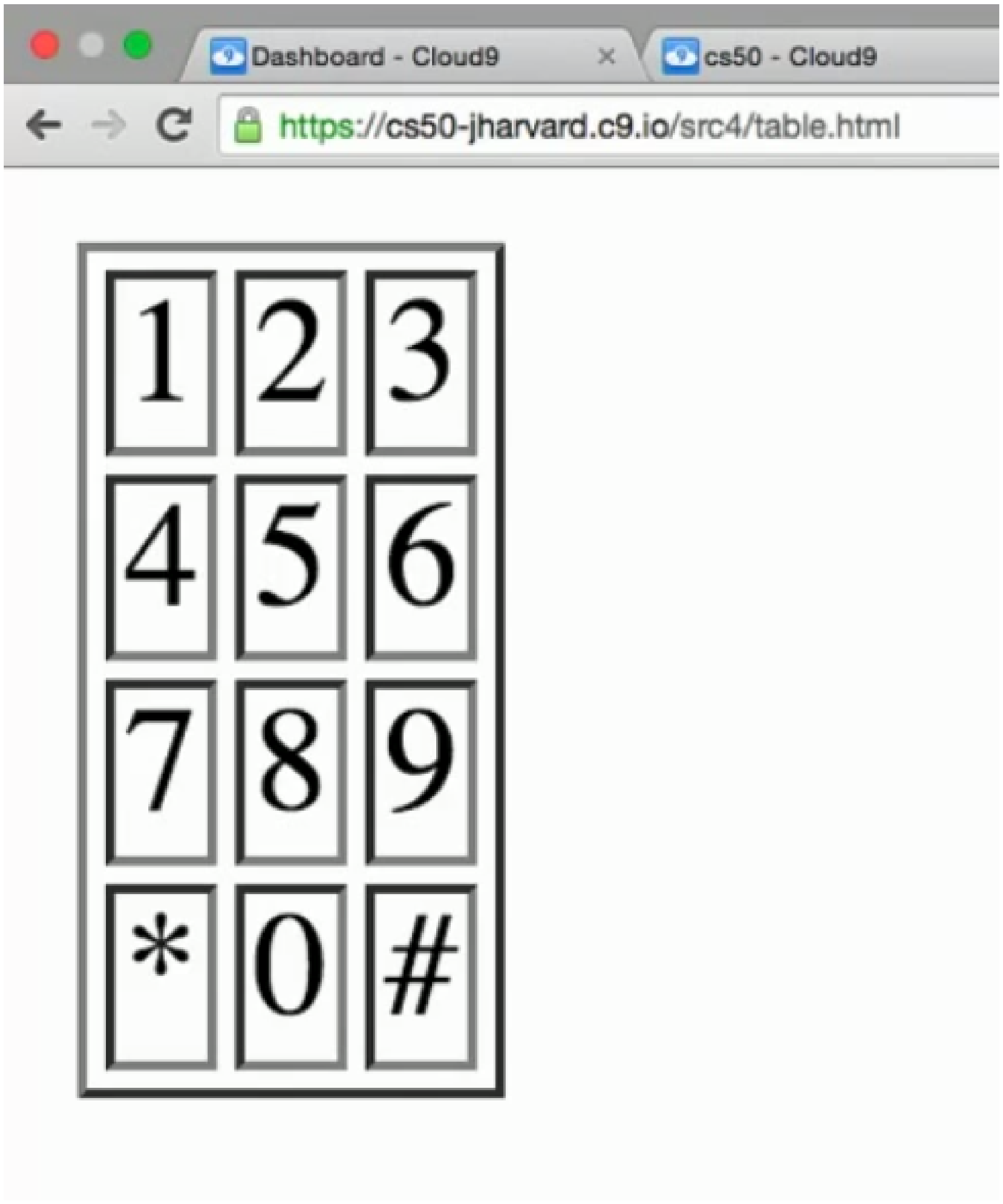
```
<!DOCTYPE html>

<!--

image.html

David J. Malan
malan@harvard.edu

This is Grumpy Cat.

Demonstrates images.

-->

<html>
    <head>
        <title>image</title>
    </head>
    <body>
        <!-- http://knowyourmeme.com/memes/grumpy-cat -->
        <img alt="Grumpy Cat" src="cat.jpg"/>
    </body>
</html>
```

  # Notice line 21 is a comment that reminds us where the image is from, and line 22 is a new tag, `<img>`, that has alternative text that reads `Grumpy Cat` in case the image doesn't load, and refers to a `src`, source, file of `cat.jpg`.

- We can actually replace that with any image URL on the internet, and borrow one from HBS's home page:



- The `cat.jpg` image was in the same directory where we had `image.html`, `src4`.
- We can also make tables in HTML, that we can later style to make prettier:

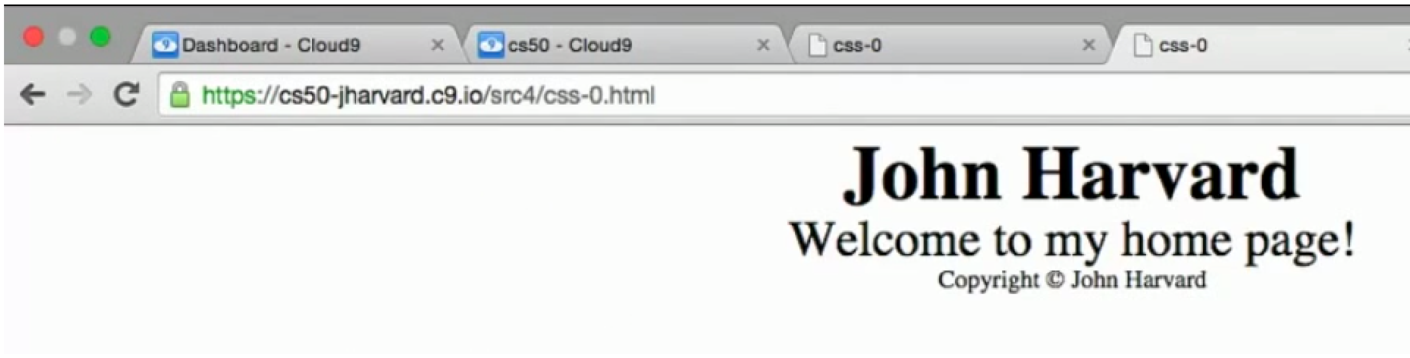


1	2	3
4	5	6
7	8	9
*	0	#

CSS3

- So let's look at `css-0.html` [7]:

[7] http://cdn.cs50.net/2015/mba/classes/4/src4/css-0.html



John Harvard

Welcome to my home page!

Copyright © John Harvard

- CSS stands for Cascading Style Sheets, which is used to style webpages. It’s another language that you can use inside HTML, or in a separate file that your HTML page links to.
- Let’s look at the source code:

```
<!DOCTYPE html>

<!--

css-0.html

David J. Malan
malan@harvard.edu

Implements a (simple) home page for John Harvard.

Demonstrates inline CSS.

-->

<html>
    <head>
        <title>css-0</title>
    </head>
    <body>
        <div style="text-align: center;">
            <div style="font-size: 36px; font-weight: bold;">
                John Harvard
            </div>
            <div style="font-size: 24px;">
                Welcome to my home page!
            </div>
            <div style="font-size: 12px;">
                Copyright &#169; John Harvard
            </div>
        </div>
    </body>
</html>
```

# There are lots of `div`s here, like invisible rectangles on the screen that divides a page, and each of them have a `style` attribute.

# The values of each of the `style` attributes seem to have their own structure too. It’s not just a single word, but key-value pairs, where "key" is some property, and "value" is just the value of that property.

  # For example, on line 21, the property called `text-align` is being set to a value of `center`. So it seems like the text inside this particular `div` is going to be centered. And we only know the particular names of these properties and values by looking it up on online references.

  # And inside that division, we have another one, where we've set the font size to 36 pixels and the font weight. The `<b>` tag in HTML is a little outdated, and it's better to factor out design into CSS anyways.

  # On line 29, we have something else that's new, `©`, which is an HTML entity, or just a tiny bit of code that represents the copyright symbol.

- In `css-1.html`[8], we've separated the CSS from the HTML somewhat:

[8] http://cdn.cs50.net/2015/mba/classes/4/src4/css-1.html

```
<!DOCTYPE html>

<!--

css-1.html

David J. Malan
malan@harvard.edu

Implements a (simple) home page for John Harvard.

Demonstrates CSS selectors.

-->

<html>
    <head>
        <style>

            body
            {
                text-align: center;
            }

            #top
            {
                font-size: 36px;
                font-weight: bold;
            }

            #middle
            {
                font-size: 24px;
            }

            #bottom
            {
                font-size: 12px;
            }

        </style>
        <title>css-1</title>
    </head>
    <body>
        <div id="top">
            John Harvard
        </div>
```

    # We've replaced the `style` attribute with ones called `id`, and those IDs are referred to again in the `<head>` of the page.

    # This allows us to change the styling more easily, rather than going through and finding the correct spots within a webpage.

    # The `<style>` tag in the `<head>` is where we'll have all the styles now, and the curly braces and semicolons are again standard syntax we can look up in online references.

    # `#top`, for example, means that everything inside should be applied to elements with an `id` of `top`.

- But we can do even better, with `css-2.html`[9]:

[9] http://cdn.cs50.net/2015/mba/classes/4/src4/css-2.html

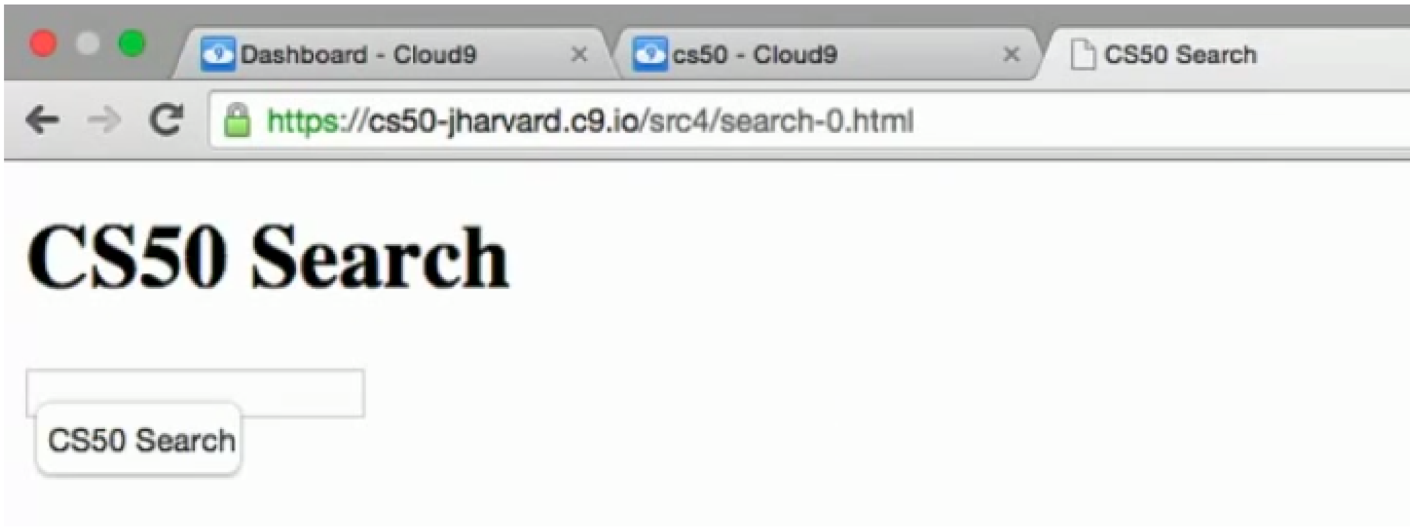
```
<!DOCTYPE html>

<!--

css-2.html

David J. Malan
malan@harvard.edu

Implements a (simple) home page for John Harvard.

Demonstrates external stylesheets.

-->

<html>
    <head>
        <link href="css-2.css" rel="stylesheet"/>
        <title>css-2</title>
    </head>
    <body>
        <div id="top">
            John Harvard
        </div>
        <div id="middle">
            Welcome to my home page!
        </div>
        <div id="bottom">
            Copyright &#169; John Harvard
        </div>
    </body>
</html>
```

# Now all the stuff that was once in the `<style>` tag has been moved to a file that looks like `css-2.css`. So multiple webpages on your website can share the same style just by linking to that file, without you having to copy all the styles manually.

- Finally, let's look at `search-0.html`[10]:

[10] http://cdn.cs50.net/2015/mba/classes/4/src4/search-0.html

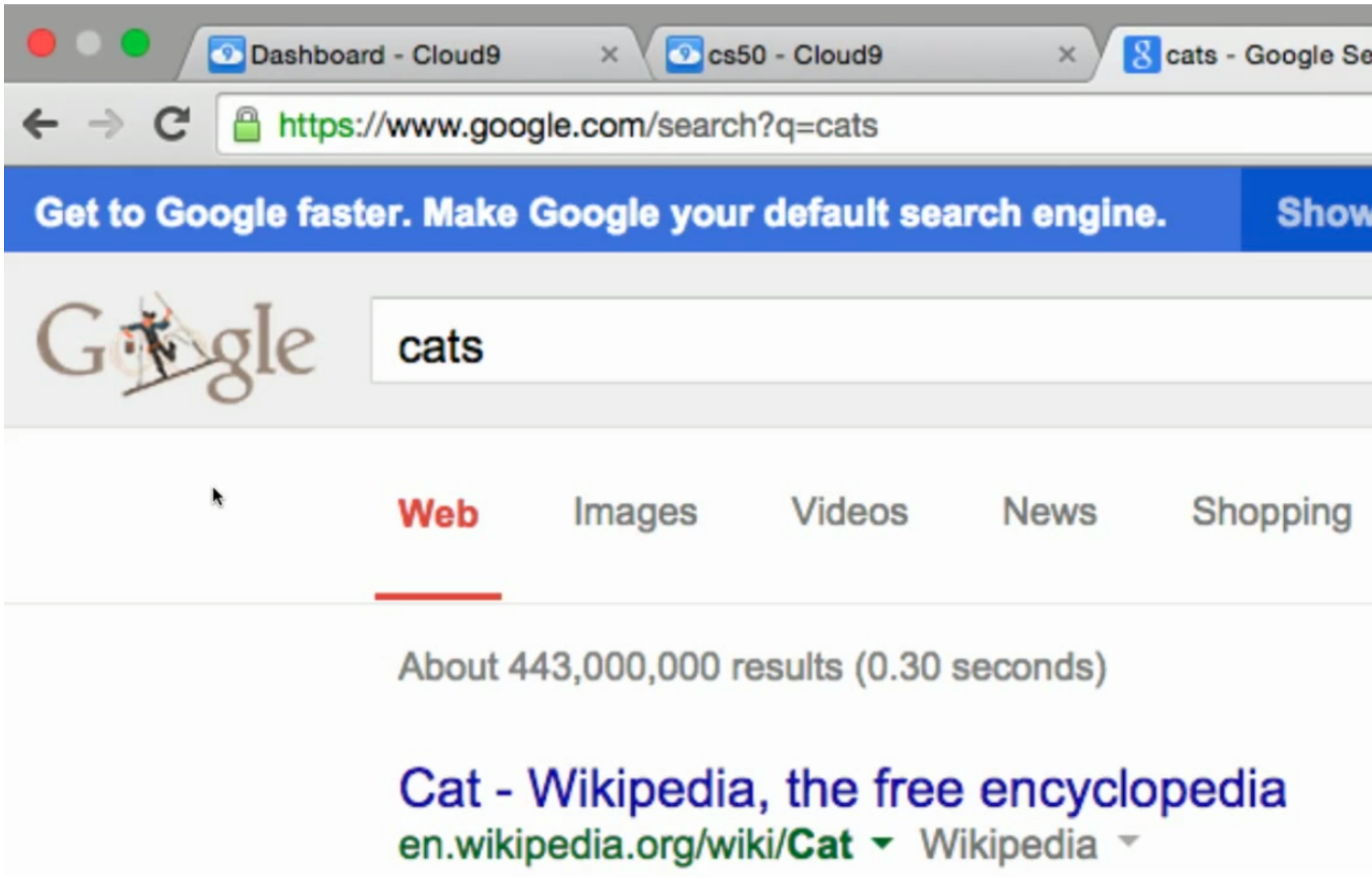
```html
<!DOCTYPE html>

<!--

search-0.html

David J. Malan
malan@harvard.edu

Demonstrates form submission.

-->

<html>
    <head>
        <title>CS50 Search</title>
    </head>
    <body>
        <h1>CS50 Search</h1>
        <form action="https://www.google.com/search" method="get">
            <input name="q" type="text"/>
            <br/>
            <input type="submit" value="CS50 Search"/>
        </form>
    </body>
</html>
```



# We can type something like `cats`, and then we'll get to the Google results page for cats. (This is sort of an example of front-end versus back-end, where we've implemented the form, the front-end, but all the processing is still done by Google, the back-end.)

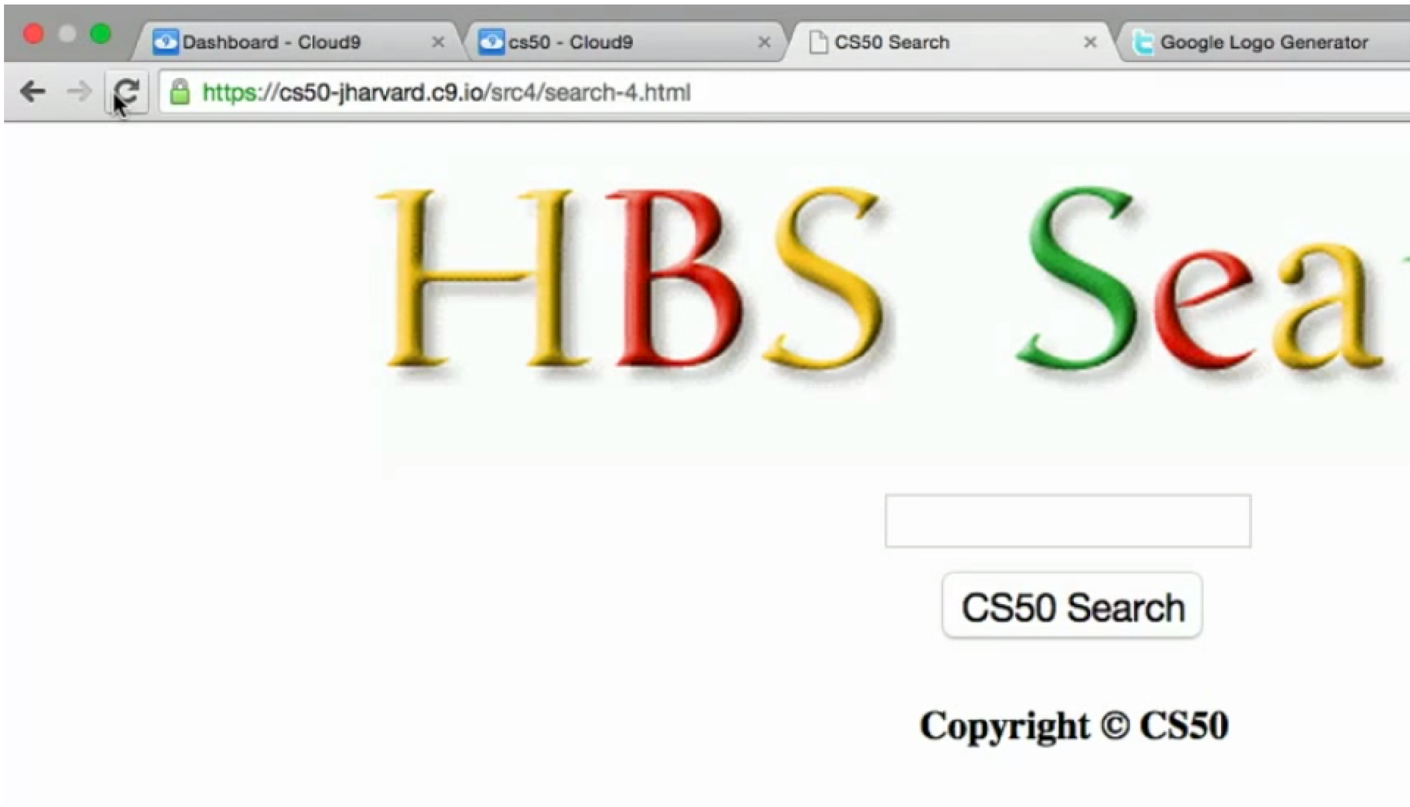
# So if we try to think about how this might be implemented, the page has an input box and a submit button, and when we click it, the browser is probably making a GET request.

- The URL we were taken to was this:


Dashboard - Cloud9
cs50 - Cloud9
cats - Google Se
https://www.google.com/search?q=cats
Get to Google faster. Make Google your default search engine.
Show
cats
Web
Images
Videos
News
Shopping
About 443,000,000 results (0.30 seconds)
Cat - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Cat
Wikipedia

- So it seem that we used HTML to build a link that looked like that, and if we look back at the source code, we first see an `<h1>` tag, or heading 1 tag, in the body, for the big bold title of **CS50 Search**. Then we have a `<form>` tag that has an `action` value of the first, hardcoded part of the URL, and then we have an input box for text with the name `q`. Then the `submit` button that we click will tell our browser to build the URL that we ended up at, based on what was inside the input box.

- We'll quickly upload a new file, `search.gif` that says "HBS Search", and get something like this with `search-4.html`[11]:



[11] http://cdn.cs50.net/2015/mba/classes/4/src4/search-4.html