Web Development

Marvin'x notes

date: 2012-07-06 21:26:01

name: cs253/Unit 1

CS253 - Lesson 1: The Basics

[TOC]

Course Outline

In this class you will learn how to build web applications! Throughout this course you will be building a blog that will have user registration and allow you to submit user entries. This blog will be online for anyone to see and play with!

In this lesson you are going to cover the following basics of the web:

- **the web**- what it is, what it looks like and how the pieces fit together,
- **HTML**- the main document type of the web,
- **URLs**-how you refer to documents on the web,
- **HTTP**- the protocol that the web runs on,
- **Web Applications**- what they are and how they fit into the big picture,

World Wide Web

The **world wide web** is a collection of HTML documents. **HTML** stands for HyperText Markup Language and it is the basis for almost every webpage - it is what glues everything together.

The links between pages are called **hyper links**, or just links, and give the internet its web-like characteristic.

The web was invented in the early 1990's and has somewhere around 30 billion pages!

Quiz: Course Outline

What is the main type of document on the web?

- a. HTML
- b. Microsoft Word
- c. PDF
- d. Plain Text

Quiz: File Types

What type of files can be found on the web?

- a. Plain text
- b. HTML
- c. Images
- d. Videos
- e. Music

Quiz: Components of the Web

2012-04-05-Image1.jpg

Use the letters that are associated with each piece of technology and match them up with their name and definition.

1. **HTTP** - the main protocol of the web
2. **Servers** - computers that host the files that makes up the web
3. **Internet** - the world's largest computer network
4. **Browser** - a program that runs on your computer to display files found on the web

Quiz: Best Browser

Which of these browsers is the best?

- a. Internet Explorer
- b. Firefox
- c. Chrome
- d. Safari
- e. Opera

Intro to HTML

HTML, or HyperText Markup Language is the heart of the web. It is made up of:

1. text content, which is what you see
2. markup, which is what the content looks like
3. references to other documents, for example, images and videos
4. links to other pages

Explore HTML here, www.udacity.com/html_playground/, and play around in this fake little test browser.

A few things to note:

Plain text is plain text in HTML, for the most part. 2012-04-05-Image2.jpg

If you want to make the text look different you have to use markup.

HTML markup is made up of things called "tags". Tags look something like this:

- `<NAME> Contents \</NAME>`

The contents can include other tags, or it may just be text. The `<NAME>` tag is called the "**opening tag**" and the `</NAME>` tag is called the "**closing tag**". Notice that the only difference between the opening and closing tags is that the closing tag has a slash, **/**, in front of the tag name.

The whole construct: `<NAME> Contents </NAME>` can also be referred to as "**an element**".

Let's learn about our first tag.

The Bold Tag

The first tag we are going to learn about is the bold tag. It looks like this:

- `<b>Contents</b>`

It has an opening `<b>` tag and a closing `</b>` tag with the slash, and anything that appears between the opening and closing tags appears bold.

b for bold.

OK, quiz time:

Quiz: Bold Tag

In the box below, make the words reasonably straightforward bold:

More Tags

Now we want to teach you another tag. This is the tag `<em>em</em>`, which stands for **emphasis**, and it makes things italic. The complete element will look like this:

- `<em>contents</em>`

There is the opening `<em>` tag and the closing tag, the contents that we want to appear italic, and then the closing `</em>` tag with the slash. The structure is just like the bold tag we saw earlier.

Quiz: Italics

What we want you to do is to make the phrase 'HTML is reasonably straightforward' italic.

Quiz: Missing End Tag

What would happen if we were to forget to close the `<em>` tag in our example?

- Nothing is italicized
- The browser crashes
- Everything after `<em>` is italicized
- Your guess is as good as mine

Making Links

Now we will teach you all a new concept. This is **HTML attributes.** Attributes appear something like this:

- `<TAG ATTR="value">Contents</TAG>`

We still have our opening tag name and the closing tag as before, but now we have this new thing called an attribute.

Attributes have a name (in the example above it is simply called **ATTR**) and a value (inside the quotes). Tags may have multiple attributes.

An example of a tag that uses attributes is the **anchor tag** `<a>`, and a full example would look something like this:

- `<a href="http://www.reddit.com">derp</a>`

Here, within the opening tag we have the opening a, an attribute called **href**, the value of the attribute which is a URL in this case. If this were rendered in a browser we would just see the word "derp", but it would be a link to reddit.com.

Quiz: Making Links

Make the words **my favorite** in the phrase below a link to udacity.com.

Adding Images

The next tag we are going to introduce is the image tag. The image tag looks like this, `<img>`, and you won't be surprised to learn it is for including images. The image tag has the following structure:

- `<img src="URL" alt="text">`

The image tag has an attribute named **src** (for "source") which equals the URL which is the location of the image file to be downloaded. This is followed by a second attribute called **alt**. This stands for "alternate" and is the text that is to be displayed when the image doesn't load.

The alt attribute is "required" in the sense that html parsers will complain at you if it is not there. Nothing will actually break if it is missing but it is really good practice to include it. If the image is moved or the requested URL is missing for some reason, this is the text that is displayed.

The text associated with the alt attribute is also the text picked up by the reading software used by blind people to access the web. It doesn't take much effort to add an alt attribute to your images, but it can make somebody's day just that little bit easier...

There is one more thing to be aware of about the image tag. Every tag we have looked at so far had a closing tag. Image tags don't. There is not contents

to an image tag, and so the closing tag is not required.

Tags, like `<img>`, that don't require a closing tag are called **void tags**.

Images will appear inline with text.

Whitespace

Let's talk about whitespace for a moment.

You may have noticed that if you entered text into the editor on multiple lines, the browser window rendered it onto a single line. This is because in HTML, all whitespace, new lines, tabs, spaces are all converted into a single space.

To force the browser to display text on multiple lines we can use another tag called the break tag, and it looks like this:

- `<br>`

`<br>` is also a void tag.

The effect of `<br>` is to cause the browser to move to a new line before displaying the next piece of content. Multiple `<br>` tags will cause the browser to move down multiple lines before it displays further content.

Another way of creating line-breaks is to use the paragraph tag `<p>`. The paragraph tag is **NOT** a void tag. Paragraph tags have content and appear like this:

- `<p>Content</p>`

The content between the opening and closing tags will be rendered by the browser as a single paragraph.

Quiz: Whitespace

Enter the HTML to draw the following text:

- Hello everyone!
- We're using two lines now!

on two lines, using the `<br>` tag.

Quiz: Paragraph Tag

OK, now let's do the same thing, only this time using the paragraph tag. Enter the HTML to draw the following text:

- Hello everyone!
- We're using two lines now!

on two lines, using the `<p>` tag.

Inline vs. Block

One last thing I'd like to talk about regarding `<br>` and `<p>`, is the question of why we have two different ways of creating line-breaks? Why do we have the `<br>` tag and also the `<p>` tag?

The answer is that the `<br>` tag is what we call an **INLINE** tag and the `<p>` tag is what we call a **BLOCK** tag.

What the `<br>` tag was actually doing is telling the browser to end the line.

The `<p>` tag acts differently. What the `<p>` tag does is to create an invisible "box". So the HTML code:

- `<p>text1</p>text2`

Creates an invisible "box" around text1. This box can have height and width. Text2 will be outside this "box".

The differences between inline and block tags will come up a fair amount in this course, and it is important to know that there is a distinction between them and they have different behaviours. So far, all of the elements that we have learned (other than `<p>`) are inline elements.

Span and Div

Two more elements that I'd like to teach you are called **Span** and **Div**. Both span and div are normal elements, that is they can both have content:

- `<span>Content</span>`
- `<div>Content</div>`

The only difference between these elements is that span is an inline element whereas div is a block element. The only function of these elements is to contain their content and there is a way of attaching styles to them to change the way in which their contents display. This is done by attaching attributes to the tags so they look something like this:

- `<span class="foo">Content</span>`
- `<div class="bar">Content</div>`

The class attribute refers to a **CSS class**. CSS classes are not something that we are going to spend a lot of time on in this course, as we will provide the CSS where needed. CSS is a separate language for adding styles to your documents.

We will be using `<span>` and `<div>` elements a lot to control the layout of text, and the important thing to remember is that `<span>` is an inline tag, and `<div>` is a block tag.

Quiz: Span and Div

Select all of the elements that are inline. Some of these are elements that you haven't seen before, so I would like you to trty them out in your browser and see if you can figure it out for yourself:

a p

div img

span strong

br form

Document Structure

That is enough HTML elements for now. I want to quickly talk about the structure of an actual HTML document before we move on.

What we have seen up until now has just been some very simple markup. We've been seeing this like this:

- `<b>Content</b>`

with just a little text and a few simple tags, but an actual HTML document has quite a bit more to it. Here we see a whole lot more HTML:

```
<!DOCTYPE HTML>
<html>
<head>
- <title>Title!</title>
</head>
<body>
- <b>Content</b>
</body>
</html>
```

This is what a complete HTML document looks like. Let's look at this piece by piece. The first line:

```
<!DOCTYPE HTML>
```

specifies the doctype. i.e. what kind of HTML this is. The "HTML" string within this tag used to be a *LOT* more complicated, but now that we are using HTML5 we have a nice clean, simple doctype.

The `<html>` and `</html>` tags surround the rest of the document providing a kind of "container" for the document.

The `<head>` and `</head>` tags enclose the document header. This part of the document contains metadata about the document and other document information. JavaScript and CSS would also be included in the document header.

The `<title>` element contains the title of the page. This is what will appear at the top of your browser, and in the browser tab, when you open the document.

Next, we have the `<body>` and `</body>` tags that enclose the actual contents of the document. So far, we have been looking at elements that would be included in the body of the document. In fact, most of this course will be concerned with generating the content that fits between the body tags. The rest of the HTML that makes up the document is important, and you will see it, and we'll be sending it over the wire, but it doesn't change very often and it is pretty simple. All of the interesting stuff happens between the body tags.

Introducing URLs

Let's talk about URLs.

URL stands for ***Uniform Resource Locator***. An example of a URL would look like this:

http://www.udacity.com//

This has three parts:

1. The protocol: http
2. The host: www.udacity.com
3. The path /

The protocol can be a number of things. For our purposes this will be http almost all of the time (it can also be https, ftp and some other protocols). It is separated from the host by a colon and two slashes - **://**

The host is the host name or domain name of the server that has the document that we want to access.

The path is the document we want to access. The shortest possible path is just the single slash.

Quiz: Correct URL

Given the protocol **ftp**, the host **www.udacity.com**, and the path **/about**, what is the URL?

Query Parameters

OK, let's add something new to our URLs, called **Query Parameters** (also known as **GET Parameters**). Here is an example of a query parameter:

- http://www.example.com/foo?p=1

This has a simple URL of the type we have already seen, with the path **/foo**, to which we have added **?p=1**. This adds an extra parameter with the name p and the value 1. URLs can have multiple query parameters:

- **http://www.example.com/foo?p=1&q=neat**

The URL now has two query parameters. The fist query parameter is separated from the URL by a question mark, and the query parameters are separated from each other by ampersands.

So what are query parameters actually for? Query parameters pass information to the server when you request the particular path. This information can

be used for all sorts of things and we will discuss some of these later in the class.

Quiz: Query Parameters
In the following URL, what is the value of the **z** parameter?

http://example.com/foo?p=foo&z=p

Fragments
Let's add one more piece to our URLs. This is a **fragment**. A fragment is separated from the rest of the URL by a hash sign:

- http://www.example.com#fragment

A fragment is usually used to reference a part of the page that you are looking at, although there are some fancier uses if you are going to do some complex **JavaScript** things**.**

The fragment is not sent to the server when you make a request. The fragment just exists in the browser.

URLs can have both fragments and query parameters, in which case the fragment follows the query parameter(s):

- **http://www.example.com/foo?p=1#fragment**

Port
The last part of the URL that we are going to look at for now is the **port**:

- http://localhost:8000/

When you make a web-request to the server, in order to make an internet connection you need two things:

1. the address of the machine (localhost in the example above)
2. a port (8000 in the above example)

By default, the port equals **80**. If you want to use a port other than 80, you can include it in the URL between the host and the path, separated from the host by a colon.

The URL shown in the example above will become very familiar to you. This will be your local development URL for a lot of this course, and also for much of your career as a web developer. You will be constantly accessing your local machine, and you will probably be doing on something other than port 80.

There are even more parts to a URL, but these aren't relevant to us right now, so we will deal with them as we go.

Quiz: Port
Identify the different parts of this URL:

http://example.com:80/toys?p=foo#blah

host

protocol

fragment

query

port

HTTP
HTTP is the main protocol of the web. It is what your browser uses to communicate with web servers. HTTP stands for HyperText Transfer Protocol.

A request from your browser for the URL:

http://www.example.com/foo

begins with a request line that looks something like this:

- GET /foo HTTP/1.1

HTTP is a very simple text protocol, so this text is sent over the Internet exactly as shown. The request line has three main parts:

1. Method - the type of request we are making to the server (GET)
2. Path - from the URL (/foo)
3. Version - HTTP/version (HTTP/1.1)

The two main methods we will be considering are GET and POST.

The host name doesn't appear in the request line. That is because we are already connected to the host. The host is used to make the connection, the path is used to make the request.

Quiz: GET
Given the following URL:

http://example.com/foo/logo.png?p=1#tricky

what is the request line to GET this URL using HTTP/1.1?

Quiz: Most Common Method
Which method is most often used for requesting a document from the server?

Making Requests
We had our request line that looked something like this

GET /foo HTTP/1.1

This is followed by a number of headers. Headers have this format:

Name: value

When you make a request, the request line and all of the associated headers are sent at once. Some of the more popular headers are:

Host: www.example.com

User-Agent: chrome

The Host header is required in HTTP/1.1 but isn't strictly required in HTTP/1.0. We have already used the host from the URL to connect to the server, so why would we need to include it in a header? Well, web-servers may have multiple names, and one machine may host many individual websites.

The User-Agent header describes who is making the request. It will generally be your web-browser, which helps the server to know what type of machine is making the request.

User Agent Header

Steve talks about his experience when getting Reddit up-and-running. User-Agents are one of the most important headers in an HTTP request.

User-Agents were really important to Reddit. Reddit was a site that was online and really popular. Users would often write scripts and so on to pull content down from Reddit, and mostly they were doing good things. Sometimes, however users would do bad things, for example spammers looking for weaknesses or trying to gain access to the system.

User-Agents were important. Sometimes users would hit the site a little too hard, or too fast, hurting the website for real users. If they had a legitimate User-Agent, the team at Reddit could look at them and take steps to moderate the behaviour . The Google-bot (Google's web crawler) was a really good example of this. However, when people turned up with fake user agents pretending to be a browser and hurting the site, then the team at Reddit would simply block them.

Using good User-Agents when you write software that interacts with other people's websites it a really nice, courteous thing to do, and is one of the things that makes the web work well for everybody. So it is important to have a nice, accurate User-Agent and to be honest whenever you can.

Quiz: Valid Headers

Which of these are valid headers?

- User Agent: chrome
- Host: www.hipmunk.com
- host www.example.com
- User-Agent: ignore me I'm a spammer
- i-made-this-up: whatever

http Response

OK, now that we have seen the HTTP requests, let's take a look at HTTP responses.

The basic HTTP response looks similar to the HTTP request. For example, if we send the request:

GET /foo HTTP/1.1

The response may look something like:

HTTP/1.1 200 OK

This is called the status line and is analogous to the request line we saw earlier. The version in the status line should match the version in the request (HTTP/1.1). The version is followed by two additional pieces of information:

The status code

The reason phrase (an English-language description of the status code)

There are some really common status codes:

200 OK

302 Found

404 Not found

500 Server error

If the code starts with a 2 (e.g. 200), this means everything worked alright. If it starts with 3, it means there's a bit more work to be done. If it starts with 4 (e.g. 404), it indicates that there was an error on the browser side, and if it starts with 5, it means that there was a error on the server side.

Response Headers

Just like the request line we saw earlier, the status line is followed by a number of headers. Here are a few examples that are commonly included with HTTP responses:

- HTTP/1.1 200 OK
- **Date: Tue Mar 2012 04:33:33 GMT**
- **Sever: Apache / 2.2.3**
- **Content-Type: text/html**
- **Content-Length: 1539**

The Date header should be there every time. It is the date that the request was made.

The Server header is similar to the User-Agent header in request line. Typically it contains the name and version number of the server. However, this could be useful information to would-be hackers, so it may be missing or contain made-up information.

Content-Type is simply the type of document that is being returned.

Content-Length is the length of the document. It is often included, but isn't strictly required since the browser will know when the data is complete as the connection will close.

There follows a demonstration of using Telnet to send a request line and headers, and view the response.

Note for Windows OS: In some versions, like Windows Seven, Telnet comes deactivated by default. To use it first head towards the control panel, programs and features, turn windows features on or off, and look for the Telnet client, select it and you're ready.

Quiz: Telnet Request

Use telnet to make a request to:

- http://www.example.com

The request is to be a GET request for the path /

What is the status code?

What is the value of the location header?

Web Applications

Most of this course is going to focus on how to run programs on servers. The purpose of a web-server is to respond to HTTP requests. There are two main classifications for server responses:

1. Static
2. Dynamic

Content is considered static if it is a pre-written file that the server simply returns. Examples of static content include image files.

More interesting are dynamic requests, which are requests where the responses are built on-the-fly by a program running on the server. Just about all the content online these days is dynamic. The programs that build these dynamic responses are called web applications.

A web application is just a program that generates content. It is run from the server, speaks http, and generates dynamic content requested by the browser and is what we are going to learn to build on this course.

Quiz: Dynamic Content

Which of these requests is for dynamic content?

- Your Facebook page
- The Udacity.com logo
- A blog's front page
- Google search results

This page was last edited on 2013/03/25 20:29:09.