

# Chapter 0, Structuring Web Pages with HTML5

John M. Morrison

August 22, 2023

## Contents

<b>0</b>	<b>Introduction</b>	<b>2</b>
<b>1</b>	<b>The User Experience, Explained</b>	<b>3</b>
1.1	What's Behind the User Experience? . . . . .	3
<b>2</b>	<b>A Look Behind the Scenes: The Internet is Mike TV!</b>	<b>4</b>
2.1	IP and MAC Numbers . . . . .	5
2.2	URLs and IP Numbers . . . . .	5
<b>3</b>	<b>Introducing HTML</b>	<b>6</b>
3.1	The Parts of Speech in HTML . . . . .	7
3.2	Some Examples of Tags . . . . .	9
<b>4</b>	<b>The Document Tree</b>	<b>10</b>
<b>5</b>	<b>Proper Nesting</b>	<b>13</b>
<b>6</b>	<b>Block Level and Inline Elements</b>	<b>14</b>
6.1	Validating Pages . . . . .	14
<b>7</b>	<b>Expanding Your HTML Vocabulary</b>	<b>15</b>
<b>8</b>	<b>HTML Etudes</b>	<b>18</b>
<b>9</b>	<b>Presenting Data and Making Tables</b>	<b>22</b>

<b>10 Sniffing Around for More, and How to Avoid the Bad Kids on the Block</b>	<b>26</b>
10.1 I'm Innocent, Don't Frame Me! . . . . .	26
10.2 Son, you have some undesirable attributes . . . . .	26
10.3 Exploring HTML files . . . . .	27
<b>11 Terminology Roundup</b>	<b>30</b>

## 0 Introduction

When you view web pages on the Internet, you use a piece of software called a *browser*. Your browser is actually a computer in software, i. e., a *virtual computer*, which understands three languages. These are as follows.

- **HTML** The acronym stands for *Hypertext Markup Language*. This gives web pages their *structure*. It describes such things as document sections, paragraphs, lists, tables, media files, and hypertext links. All of these exists inside portions of a document called *elements*, which make up rectangular subregions of the browser window. HTML is a markup language with a fairly simple grammar that understood by the browser. The browser uses HTML to format and display a document. We will be using HTML5, which is the current standard.
- **CSS** This acronym stands for *Cascading Style Sheets*. Style sheets determine the *appearance* of web pages. They control such things as page layout, colors, and fonts. CSS works with HTML and controls the style of various elements of your document, and how they are displayed on your page. One style sheet can control the appearance of a whole group of documents, giving them a consistent appearance.
- **JavaScript** This is a full-featured computer language, unlike its friends HTML and CSS. It gives web pages their *behavior*. JavaScript can reach down into the lower two layers, and thereby cause web pages to change their appearance or structure in response to user interaction.

Throughout we will use the Firefox or Chrome browsers; both have very nice arrays of web development tools. It is a good idea to have both installed and to experiment with both, since they are both very widely used.

- Download Firefox here: <http://www.mozilla.com>
- You can download Chrome here:  
<http://www.google.com/intl/en-US/chrome/browser/>

# 1 The User Experience, Explained

To view sites on the Internet, you must have some kind of Internet connection. At your home or school, you will need to use an ISP (internet service provider) to gain access to the Internet. Your browser works via this connection.

The Internet is organized in a manner entirely analogous to that of a computer's file system. It is a file system that is spread across millions of computers that are connected to each other worldwide. In a computer file system, you specify the location of a file using a path. The internet works similarly; its paths are called URLs, or *uniform resource locators*. All resources on the Internet (web pages, media, etc) have a unique URL that gives their location. Computers on the internet that offer items for you to browse are called *servers*. Your computer, when you are viewing websites, is called a *client* on the internet.

Just as with file systems, URLs can be absolute or relative. Absolute URLs begin with a prefix of the form `foo://`. The most common prefixes are `http://`, `https://` and `ftp://`. Relative URLs often occur on web pages; these are relative to the page you are viewing. They are used to refer to other pages on the same site. They will have no prefix.

The prefix `http://` means *Hypertext Transfer Protocol*. The text in an HTML document is called hypertext; this name owes to the presence of links to other documents in HTML; the additional `s` in `https://` means “secure;” you will often see this on sites that require a password and login. The prefix `ftp://` denotes *file transfer protocol*; this is used download files from servers in any format.

## 1.1 What's Behind the User Experience?

Here is a rough description of what happens when you go to a website. Suppose you start Firefox and type the URL

```
https://johnmmorrison.com
```

into the URL window at the top of your browser. You then hit the ENTER key to get things started and....

1. When you surf the web, your computer, the client, is the seeker of data on the web. Via your Internet connection, your computer contacts the *host* or *server* `johnmmorrison.com` and makes an *HTTP request* for the page there.
2. Residing on the host is a program called a *web server*; it acts as an intermediary between the file system and the web. It receives the HTTP request and sends back the contents of the *index page* for the site. The

Apache Server is the most common web server used today; information about it can be found at [2]. The most common name for an index page is `index.html` or `index.php`. The browser will also download any CSS or JavaScript present on or linked to the page. If media are to be displayed, these are downloaded, too.

3. Your browser interprets the HTML and CSS and formats the page. Any JavaScript present is loaded into the browser process's memory. The page is displayed in the content pane of your browser.
4. If you click on interactive elements on the page, the JavaScript on them runs in your browser on your machine. If a modification occurs on the page you are viewing, the page residing on the server is not changed, rather the change occurs on the copy of the page your browser has downloaded; the server's page is not changed by JavaScript. You will often, for that reason, hear JavaScript described as a *client-side language*. If you click on a link to another page, the target page loads and the process begins anew.

## 2 A Look Behind the Scenes: The Internet is Mike TV!

To more fully understand how a page gets from server to your client, we first refer you to a famous film, *Willy Wonka & the Chocolate Factory*, which was based on the novel [1] by the puckish Roald Dahl. This pleasingly dark story relates the tale of the fates of several unwholesome brats in the quest for a great prize (no spoilers here).

One of these fine little terrors is named Mike TV. His entire weltanschauung is derived from television. How sad. It's time to crank up you lappy and watch this charming little vignette:

<https://www.youtube.com/watch?v=007uWNS5zQM>

A masterful performance is made by the late, great Gene Wilder here as Mike is shuffled off to an ignominious, and entirely deserved, fate. Lest you think this metaphor is fanciful, it's time to view a more serious video.

[https://www.youtube.com/watch?v=Cq\\_g5u-sDqU](https://www.youtube.com/watch?v=Cq_g5u-sDqU)

Yes, the page and its contents are broken up by the web server into little bits called *packets*, each of which knows where it came from, where it's going, and how to assemble itself, along with the others, once they all arrive at the destination. In short, *the Internet is Mike TV!*

## 2.1 IP and MAC Numbers

All devices connected to the web have two identifiers. A MAC number is unique fixed address given to hardware devices. Every computer or internet device has its own MAC number. When you connect to the Internet, your machine is issued an IP number. If you use the Internet via work or school, your connection may, in part, be authenticated by your device's MAC number, as well as your entry of a username and password to use the network.

Your IP number might be static (you have the same IP number all of the time) or dynamic (your IP is allocated from a pool of IP numbers owned by your Internet service provider (ISP)). The IP number provides the address to which packets arrive when you download a web page and its ancillary files. If two devices have the same IP number, mysterious and awful stuff happens.

### Googling Exercises

1. Determine your computer's MAC number.
2. What IP number are you using in your current session? How do you find out?

## 2.2 URLs and IP Numbers

Every client or server must have an IP number to use the internet. You would properly ask, "How, given a server's URL, do you find its IP number?" This is handled by something called DNS (domain name servers). Here is how it all works.

When you visit a site, you download its HTML, CSS, JavaScript and media files. Your browser will cache [store] these so if you return to the site, the cached version will be shown. This saves time if you are switching back and forth between a few sites. If you reload the site, the cache for it will be overwritten with the new version. As well, the browser also saves the URL for the site and its IP number in its *DNS cache*. This leaves the question: how did it learn the IP? Think Cat-in-the-Hat.

If you go to a new site, the browser will check its DNS cache and notice that the site is not in its cache. It then checks your ISP's DNS cache. This contains the URLs and IPs for sites recently visited by all of the users in your ISP. If the browser finds it there, it uses the IP to download the site and puts the site in its cache and adds the IP and URL to its DNS cache. What happens if your browser does not find the IP matching the URL you entered in the ISP DNS cache? Time to punt.

Your browser then connects to a nexus of sites on the web that hold a giant lookup table of server URLs and IPs. If it does not find it there the server you

are searching for probably does not exist. You get an error message. Otherwise, the IP is found and the site is downloaded. Its URL and IP go into your ISP's DNS cache and your browser's DNS cache.

Periodically, DNS caches and your browser's cache gets purged of items not visited for a relatively long time.

### 3 Introducing HTML

To put all of this in context, let us begin to introduce HTML. HTML is a formal language for the structuring of web pages. Because it is a language, it has parts of speech, a vocabulary, and grammatical rules you must follow for a page to be well-formed. The browser will take your document and *parse* it; this is the process of extracting meaning from the document so the browser can do its job.

The first page you make will be the *index page*. The name of this depends on how Apache is configured, but it is most commonly called `index.html`. If you are just working on your local machine, it is a good idea to give this principal page the name `index.html`. Place the text shown below in the file; it is a minimal HTML5 document. Also, make a copy of this file and call it `shell.html`. You can copy this shell as you make other pages. Doing so will save you some a lot of typing.

For now, you can create this page on your local machine with your text editor. If you have access to a server, you will learn a little later in this chapter how to set up your site and get the file where it is needed. If you know how to use the UNIX text editor `vim`, you can create the file directly on the server.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>
My first Page
</title>
<meta charset="utf-8">
</head>
<body>
<p>Hello</p>
</body>
</html>
```

**How do I see it in the browser?** Pull down the File menu in your browser and select the Open... menu item. This will bring up a file chooser dialog. Use it to open your file. You will see a page with the word "Hello" on it and your title will be visible in the browser's title tab. The title gets obscured if you have

too many tabs open in your browser window. If you pull the tab out of the browser window, the title is easily visible.

Let us begin by looking at the contents of the index file and learning about what they do. The header

```
<!DOCTYPE html>
```

specifies the document's type. This document is an HTML5 document. It will enable the W3C validator, which you shall meet soon, to check your HTML for correctness. This header is an example of a token called a *tag*, which is a part of speech in the HTML language. All tags are enclosed in angle brackets like this `< ... >`. The line

```
<html lang="en">
```

begins your document and tells the browser you are using the American dialect of the English language. Now let's look at the rest of it.

```
<head>           This begins the document's head.
<title>         This begins the document's title.
My first Page   The title is "My first page".
</title>       This ends the title.
<meta charset="utf-8"> This says we are using the utf-8 character set
</head>       This ends the head.
<body>         This begins the body.
<p>Hello</p>   Here is all the document text in a
                paragraph.
</body>       This ends the body.
</html>       This ends the document.
```

Our next job is to understand how HTML works. It is a language with its own grammar and vocabulary. We will first focus on grammar, then we will show you how to expand your vocabulary so you can create all of the familiar features you see on web pages.

### 3.1 The Parts of Speech in HTML

Now we will explain what we are seeing. HTML is, is what is called a *markup language*; it specifies the structure of a document. We can use it to “talk” to the browser and get it to display text, images, and other items in its content pane.

We shall now look at the parts of speech in HTML and how they relate to each other. The most basic part of speech in HTML is the tag.

Text forms the nouns of HTML. It is flowed onto a web page. All text in HTML is bounded by tags.

Tags are tokens that have the form `<foo>`, where `foo` is a group of alphanumeric characters. Tags come in three types: opening tags, closing tags and self-closing tags. Here is a simple field guide. In each case here `tagname` is called the *type* of the tag.

1. Opening tags look like this: `<tagname>`. You can see that `<head>` is an opening tag.
2. Closing tags look like this: `</tagname>`. The string inside of the closing tag always begins with a slash. You can see that `</html>` is a closing tag.
3. Self-closing tags often look like regular opening tags : `<tagname>`, an ending `/` is no longer required. You may see `<tagname/>` in older code.

```
<meta charset="utf-8">
```

The type of this tag is `meta`; the `charset="utf-8"` is called an *attribute*; in this case, the attribute is saying, “Use the extended English character set.” The item `charset` is called a *property* and `"utf-8"` is the property’s *value*. Notice that, in an attribute, we do not put spaces around the `=` sign. This is a nearly universally-observed style convention. Many tags will require you to specify one or more attributes for them to do their jobs.

Grammatically, tags are verbs in imperative form. Open tags say “Begin ...!” Closing tags say, “Stop ...!”, and self-closing tags say “Do ... right now!”

An opening tag and a closing tag *match* if they have the same type. Material between matching tags is called the *element* bounded by the tags. Self-closing tags bound *empty elements*. The purpose of tags is to *delimit* elements; i.e., they tell where elements begin and end.

Text by itself is an element. In the example we saw the line

```
<p>Hello</p>
```

The text `Hello` is an element all by itself. It is an odd exception because there is no tag specifying the end of the text element. Text elements can be thought of as self-beginning and self-ending. When the text runs out, that is it. The beginning or end of text is always attended by some other tag. In this example a paragraph tag does that job. Text elements must always occur inside of some other element. They should never be placed directly into the body of a document.

Every HTML tag has a set of default properties, which are specified by your browser’s *user agent*. For instance, the `body` tag by default makes all text black and left-justified, and the background white. The user agent also specified things like default margins for the document and the default appearance of elements



that we will see later such as lists. You can use attributes to modify the behavior of elements bounded by tags. Attributes attached to a tag are in force inside of that tag's element. You can have several attributes as shown here. Note that the value attached to each attribute is always inside of quote marks.

```
<tagname property1="blah" property2="ugh"...>stuff</tag>
```

Attributes behave like adverbs that modify the action indicated by the imperative conveyed by the tag. Notice that the value attached to each property must be enclosed in quotes. You may use single or double quotes, but you must use the same type of quote on both ends of the attribute value. Closing tags must not contain any attributes; only opening or self-closing tags may have attributes.

## 3.2 Some Examples of Tags

Now let us look at our very simple HTML document. Line numbers have been added here for convenience.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <title>
5 My first Page
6 </title>
7 <meta charset="utf-8">
8 </head>
9 <body>
10 <p>Hello</p>
11 </body>
12 </html>
```

Line 2 contains the tag `<html lang="en">` which demarcates the beginning of the document. Its matching tag, `</html>` occurs on line 12. The `html` element contains the entire document, save for the `!DOCTYPE` declaration. Anything other than the `!DOCTYPE` tag outside of the `html` element will not be seen by the browser.

Line 3 begins the head of the document and line 8 ends it. This document contains one self-closing tag, the `meta` tag.

Try inserting this after line 10 and before lines 11 and 12.

```
<p></p>
```

What we have here is image tag inside of a paragraph tag; the `img` tag places an image on the page. If the image is not available the `alt` text is displayed instead. The `alt` text is also used by screen readers to describe images to blind computer users. Using this attribute makes your page accessible, and it is a standard on the Web. It also avoids the ugly little icon with the red X in it you often see on the web. Images are like text in that they should not be naked inside of the body; you should place them inside another element, such as a paragraph element. You should think of an image as an “overgrown character.”

If you wish, you can navigate to the page with the rhino image, download it, save it as `rhino.gif`, and place it in the same directory as your HTML file. Display it using the following code.

```
<p></p>
```

### Programming Exercises

1. Place an image tag inside of this paragraph element.

```
<p style="text-align:center"></p>
```

and see it get centered on your page. How do you think you can right-justify the image?

2. Add this attribute to the image tag, `style="width:50%"`. What happens? Change the percentage and observe the effect.
3. Now replace the width's value with `200px` then `400px`. What happens?
4. Go out on the Internet and download an image (right click and use Save Image As). Put it on your index page.

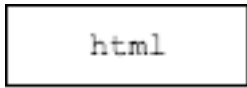
## 4 The Document Tree

Key to understanding how the style sheets of the next chapter work is an understanding of the document tree. Well-formed HTML documents have a tree structure that the browser uses to format them. This is a rooted tree, and it has an appearance entirely similar to that of your file system. When your browser parses the HTML in your document, it creates this tree in memory.

Let us begin with a super simple HTML file.

```
<html lang="en">  
</html>
```

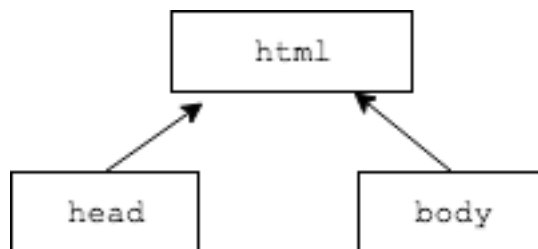
The tree just has a lonely root.



Now let us install a head and a body.

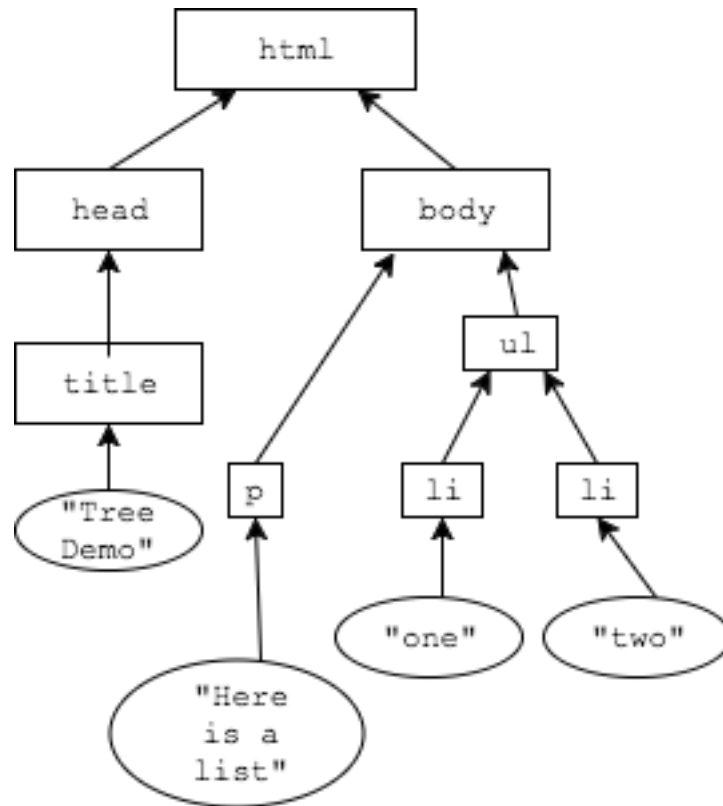
```
<html lang="en">
<head>
</head>
<body>
</body>
</html>
```

Here is what happens to the tree; the root node now has two (sibling) children.



Now we will put in a little content.

```
<html lang="en">
<head>
<title>Tree Demo</title>
</head>
<body>
<p>Here is a list</p>
<ul>
  <li>one</li>
  <li>two</li>
</ul>
</body>
</html>
```



Here is what we have. The `html` element is the root containing the entire document. Its children are the `head` and `body` elements. Inside of the `title` element is a *text element*, shown in an oval with its text content. You can see a similar thing in the `body` element.

Now you ask, “Why is this tree structure important?” One reason is that it completely specifies the structure of a document. It shows which elements are inside of other elements. All of the tag nodes (insides of rectangles) actually store any attributes for that element.

When we develop CSS and JavaScript, we will locate elements in a document using this tree and we will change their appearance using CSS or change the tree itself using JavaScript. It is absolutely critical to understand how the document tree works, for all that follows depends mightily upon it.

### Exercise

1. Create an HTML file and draw its document tree. Look ahead in the next section, and then open your document in the browser and open the

Elements tab in the developer tools. You have a solution manual to this exercise!

2. Draw a document tree and create the HTML file from it. In the diagram we just studied, here is what to do. Start in the `html` block. This is the `html`. Find the left child (`head`) and go there. This is `head`. Now enter the `title` element. This is `<title>` Inside of the `title` element is a text element. Put that text in `title` and return immediately to `title`; note that text elements are self-terminating. Now exit the `title` element; this yields `</title>`. By now you should have the idea. Use the tree to generate the entire document. Make your own file and do this yourself.

## 5 Proper Nesting

Observing these rules will make our pages render correctly. They determine the hierarchy among your page's elements.

1. Every opening tag must be matched with a closing tag of the same type. This way, the opening and closing tags bound the element corresponding to that tag.
2. Tags must close in the reverse order that they open.

If the first rule is violated, Otherwise, the element “leaks” out of the tag and the browser has to figure out what you are doing. This, at the least, interferes with the browser's efficiency, and at worst, produces unanticipated errors in the rendering of your document.

The second rule ensures the tree structure of the document: elements can only overlap if one of the elements is entirely inside of the other.

These rules constitute the *proper nesting rule*. A document meeting this specification is said to be *well-formed*. Keeping your documents well-formed helps the user's browser to format your page efficiently to the screen, and will render as you expect them to.

Here is a simple test for well-formedness of a document. Start with an empty stack of slips of paper. Now scan through the document in order. Imagine that every time you see an opening tag, you write its type on a slip of paper and put in on top of the stack. Whenever you see a closing tag, you first check the top of the stack. If the type of the closing tag does not match the type of the tag on the top of the stack, you know your document is not well-formed. If it is, remove the top tag from the stack. Do this until you have traversed the entire document. When you are done, the stack should be empty. If not, the slips left tell you the types of unclosed tags you have in your document.

**Note** The browser is happy to make choices for you. Sadly, these will often be crappy choices. Using proper HTML tells the browser *exactly* what to do. Do not allow the browser to make choices, or you will only live a short time to regret it.

Self-closing tags close themselves, so you do not have to worry about matching for them. Since they bound empty elements, you do not have to worry about bounding the element inside of a self-closing tag.

Tags types should consist solely of lower-case letters or numbers. You will see upper-case tags on some older pages. Do not do this. Following these simple rules will make your page load faster and work better.

The validator, which you will soon meet, will help you to keep your documents well-formed.

## 6 Block Level and Inline Elements

HTML has three types of elements, *block-level*, *inline*, and *inline-block* elements. Block-level elements may go directly into the body of a document. Block-level elements bound an element that is a vertical section of a page. The paragraph element bounded by the `<p>` tag, is a block-level element.

Inline elements must occur inside of block-level elements. An example of this is the `<img>` tag. Text elements are inline elements. Inline elements describe textographical portions of your documents, i.e., they “flow like text.”

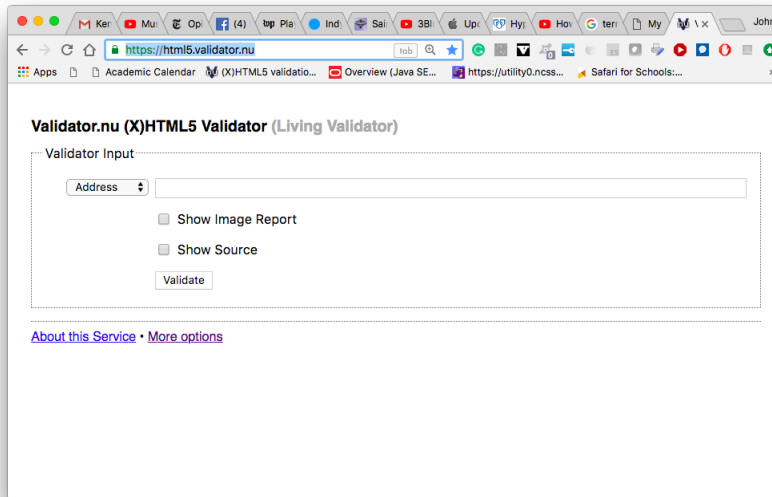
Inline-block elements are empty elements that place things such as images and text boxes on the page. We will discuss these in more detail later. For now, treat them as if they are inline elements.

Tags that appear in the `<head>` element do not admit to these classifications; they are metadata (data about) the document and they specify document properties, including such things as the document’s title, any style sheets or JavaScript linked to the document, and character set being used.

As we introduce new tags, we will specify thier type: block, inline, or inline-block.

### 6.1 Validating Pages

You will also want to use the validator to check the grammar on your pages. To do this, go to the site <https://html5.validator.nu/>. Here is what the page looks like.



The **Address** button is actually a menu. If your server is visible to the world, you can choose the **Address** and enter your URL. You can change to **File Upload** to upload a file. You can also right-click on your page, select View Source, select all and paste the text into the Text Field. Try all three ways and see what you prefer.

Go to the first error message. It will indicate that the fault lies on a particular line. Use your text editor to go to that line in your HTML file.

You should look on and around that line. Fix the first error message's problem. If the next couple of errors are easily fixable, fix them too. If not, it's time to revalidate. Repeat the process until the document validates.

You should take a document you know validates, place errors in it and look at the error messages. This way, you will become familiar with how the messaging works and you will more easily debug your document. This ability to read and debug with error messages is a very valuable one to a programmer; it pays to gain a lot of skill at it.

## 7 Expanding Your HTML Vocabulary

Now put this code into the body of your document.

```
<h1>Hello</h1>
<h2 style="text-align:right">Hello</h2>
<h3 style="text-align:center">Hello</h3>
```

```
<h4 style="text-align:left">Hello</h4>
<h5>Hello</h5>
<h6>Hello</h6>
```

The tags `h1`–`h6` produce *headline text*; by default, the text contained in them is bold and left-justified. The larger the number, the smaller the text. The headline text elements are all block-level. The text `Hello` appear in each headline element above is an inline element.

The `style` attribute indicates that you are using a *local style sheet*; this affects the display of the text. Also notice that when a tag closes, the attributes given it are forgotten. This ensures that attributes of a tag are only in force inside of that tag's element.

If you omit a closing tag, you will get a “leak:” the effect of the tag will leak beyond the point where you intended it to stop. Remember the *Robert Fulghum Rule*: if you open a tag make sure you close it when you are done! Remember, self-closing tags close themselves so you don't have to worry about closing them.

You can validate your page to ensure it stays well-formed as you create it. This makes it easier to extirpate errors and keep your page valid. The validator will tell you where to look if your HTML is not valid.

Let's add a paragraph of text and make some of the text bold. Bold text is produced by the `<strong>` tag in its default guise. We will also add italics with the `<em>` tag. Observe how the closing tag tells the browser where to start and stop bold-facing and italicizing text. Append this code to the headline text you placed in the body of your page.

```
<p>Here is some text that says something <strong>very important
</strong> <em>Never</em> overdo changes in font.</p>
```

```
<p>Notice how paragraphs are begun and ended by using open and
close paragraph tags. <strong>Always</strong> close your
paragraphs. This way, your intent is clear and your paragraphs
will render cleanly. Failing to close paragraphs may result
in nastygrams from the validator.</p>
```

Observe that the `<strong>` and `<em>` tags shown here are inline elements.

Now let's make our first link to another page. To make a link, you use the inline tag `<a>`; the `a` stands for “anchor.” The anchor tag has an property called `href`, which stands for “hypertext reference.” To make a link to Google, enter this text.

```
<p>Here is a link to <a href="http://www.google.com">Google</a>.</p>
```

The contents of the `<a>` element form the visible link text. We put this example in a paragraph by itself. You can put links anywhere in a text element. They can



reside alone in a paragraph or they can be embedded in text. Since anchor tags are inline elements, they should not appear directly in the body of a document.

By default the browser displays the link text underlined in the familiar blue. If you fail to close the anchor tag, the link text will leak onto the rest of your document; your text editor's syntax coloring will make leaks clear to you. To see this add a couple of sentences to the text and remove the closing tag.

The quote-enclosed value assigned to `href` is a URL. This URL can also be a file name for a file located in the same directory as your index file.

If you are using a server, you can link to any file the `public.html` subtree of your file system by using an absolute path or a path relative to the location of the page you are linking from. If your site gets fairly large, you should organize it into several directories, each of which contains further directories and other files. Each directory you create in your `public.html` subtree should have an `index.html` file.

You now have a small HTML vocabulary, which you should seek to expand. We will return to look at tables and lists, after we learn a little about style sheets.

1. Mozilla Tag Reference, [4] When using this list, avoid using non-standard, deprecated, and unimplemented tags. Click on the link for each tag and it will give you details on proper usage. Some attributes alter appearance; avoid these. Later we will learn how to use CSS to achieve any effects you want for formatting and color.
2. W3Schools, [6] has lots of useful tutorials. The Try It Editor provides a nice sandbox for experimentation.
3. Be a top dog with this site [3]. It seems to be an up-and-coming place for standards-compliant HTML. There is a very nice book that accompanies the site.

**Programming Exercises** For these exercises, you will need to go to the W3schools page. This is a quick guided tour to its very useful learning resources.

1. Go to the site `http://www.w3schools.com`. Click on the button that says Learn HTML. You will see an example entitled "Example" on the page. Click on the Try it Yourself button. Change the `<h1>` heading to `<h3>`. Hit the Run button.
2. Add another paragraph to the document and hit Run.
3. Place a link in the paragraph to Google. Hit Run. Note that the link does not work, but you can see it rendered with the correct appearance.
4. On the left side of the page, you will see links to tutorials on an assortment of topics. Click on the one that says HTML Lists.

5. Try the example; it introduces unordered lists. Then go to your index page and make an unordered list.
6. Scroll down to the Ordered List example and repeat the last exercise.
7. Go back to the original URL <http://www.w3schools.com>. Now select HTML REFERENCE. Here you will see a list of all HTML tags.
8. Scroll down to the `<ol>` tag. You will see complete information on that tag. Do not be concerned you don't understand it all. As we progress we will learn more. Experiment with the example there. You now know where to look to learn about new tags.
9. Try making a definition list using the `<dl>` tag. Use the example in the reference to figure out what is happening. You will see unfamiliar tags; look them up, too.

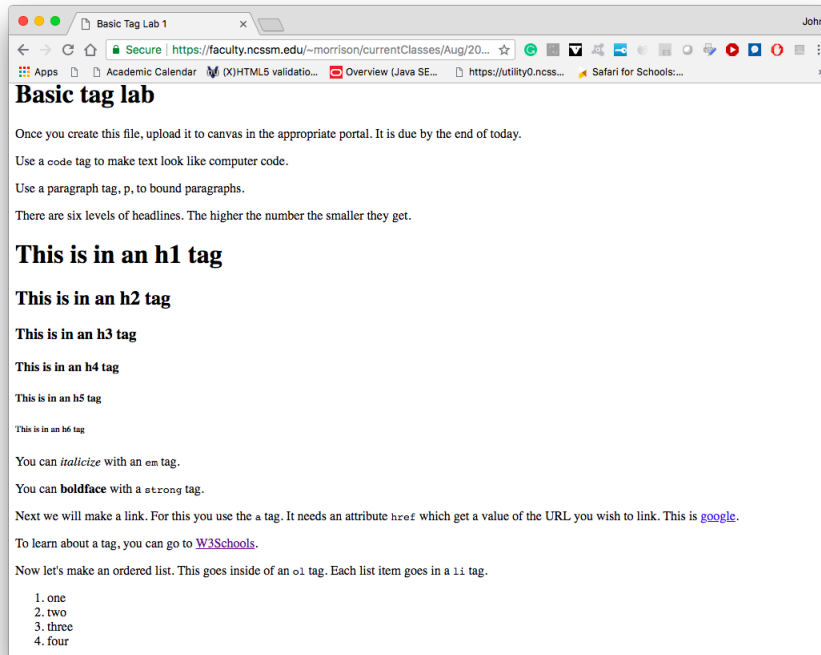
## 8 HTML Etudes

Here you will be shown images of small HTML pages. Your job is to create an HTML file that replicates them as best you can, using no styles. Your results should be able to secure the approbation of the HTML5 validator.

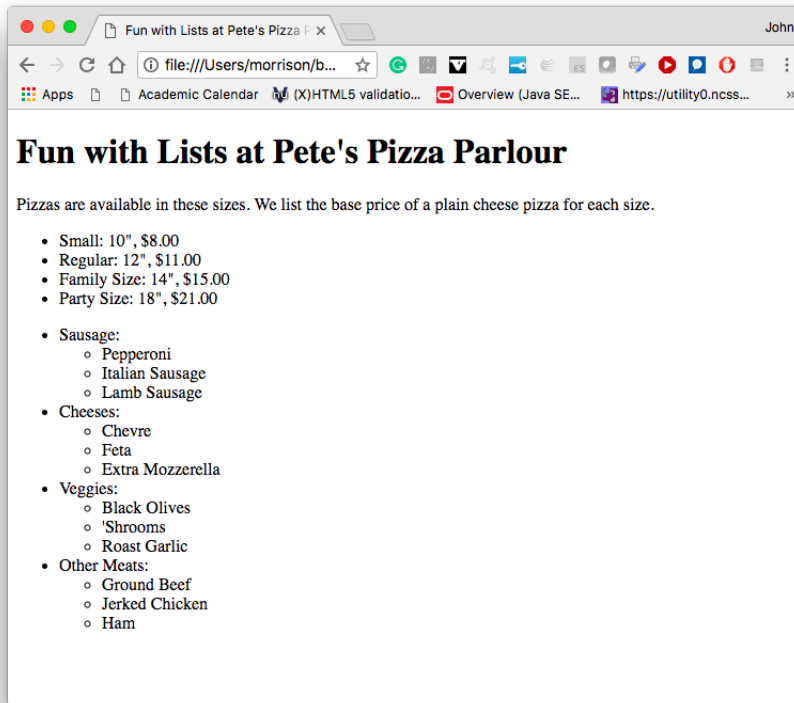
Start with a shell file. Name this file `tagLab1.html` and save it in your text editor in the directory for this class. Then, in a browser tab, choose File → Open..., or hit control-o (command-o on a mac), and open the file with the browser. You will see the shell file displayed.

For this first exercise, you should follow the hints provided in the text. Use W3Schools to learn about anything unfamiliar. As you edit, save then refresh the browser and you will see your progress.

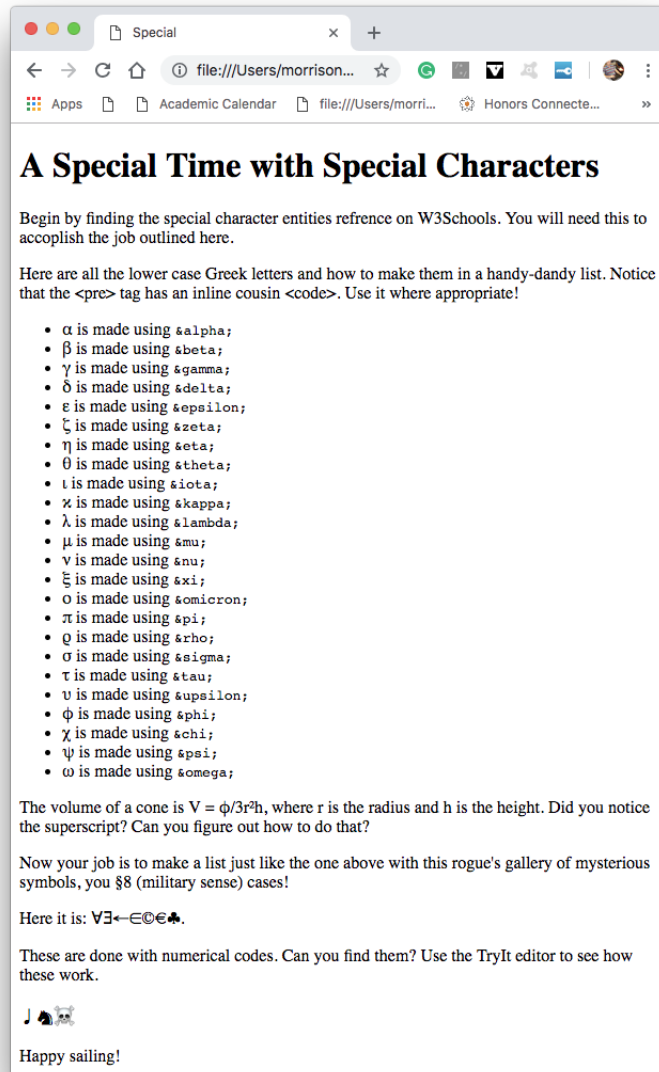
Reproduce this page as seen in the screenshot below.



Now you can try your hand at nested lists. Here is an important hint: each nested list is part of the list item it lives in, so place your `</li>` carefully. Make sure your code validates.



Finally, you will now learn about HTML special characters entities. W3Schools has a reference on these. They are of the form `&string;` and they will render as special characters in the browser window. Remember to validate as you go.



## 9 Presenting Data and Making Tables

There are three basic ways to present data in HTML: ordered lists, unordered lists, and tables, which are the subjects of this section. From the exercises in the last section, you should now know about ordered and unordered lists.

We quickly summarize these here. Ordered lists are, by default, enumerated with Arabic Numerals. They are delimited by the `ol` tag. Items in unordered lists are set off with a bullet character, `•`, by default; they are delimited by the `ul` tag. List items are each delimited by the `li` tag. Text may be placed directly inside of a `li` tag.

Below we show some code for each. An unordered list of presidents listed backwards looks like this.

```
<ul>
  <li> Joe Biden</li>
  <li> Donald Trump</li>
  <li> Barack Obama</li>
  <li> George W. Bush</li>
  <li> William Clinton</li>
  <li> George H. Bush</li>
</ul>
```

An ordered list of the same presidents looks like this.

```
<ol>
  <li> Joe Biden</li>
  <li> Donald Trump</li>
  <li> Barack Obama</li>
  <li> George W. Bush</li>
  <li> William Clinton</li>
  <li> George H. Bush</li>
</ol>
```

Tables allow us to present data in rows and columns. We begin here by listing the important tags for ready reference.

- `<table>` This tag bounds the table element. You may use the `style` attribute to control the table's width. It is best to use a percentage to set width, as in `style="width:50%"`, rather than using pixels or other units. Remember, you have no control over the size of the client's browser window.
- `<thead>` This can delimit an element for the table's header and you can attach style rules to it.

- `<tbody>` This can delimit an element for the table’s body and you can attach style rules to it.
- `<tfoot>` This can delimit an element for the table’s footer and you can attach style rules to it.
- `<tr>` This tag bounds a table row element. Tables are set row-by-row. Table datum and table header elements go inside of table row elements.
- `<th>` This tag bounds a table header element. By default, text is centered and boldface in a table header.
- `<td>` This tag bounds a table datum element. By default, text is plain and left-justified. For both table header and table row elements, the attribute `colspan` can be used to make a cell span more than one column and `rowspan` can be use to make a table cell span more than one row.

The general structure of a table looks like this. Any table cell can be a header or a regular datum cell.

```

<table>
<tr><th>First Header</th><th>Second Header</th> .... </tr>.
<tr><td>First Datum</td><td>Second Datum</td> .... </tr>.
.
.
.
</table>

```

It is up to you to make sure you have the correct number of table cells in each row or the right-hand side of your table will be “ragged” and aesthetically unappealing.

**We are going to cheat!** It is hard to work with unstyled tables because they appear to be a mess. To alleviate this problem, we will use the following CSS file to put rules on our tables and to make table header cells be yellow. Place this code in a file named `table.css`.

```

table, th, td
{
    border: solid 1px black;
    border-collapse: collapse;
}
table
{
    margin-left: auto;
}

```

```

    margin-right:auto;
}
th, td
{
    padding:.5em;
}
th
{
    background-color:#FFFF00;
}

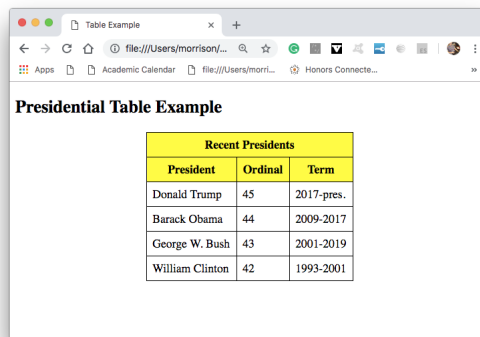
```

In the head of your HTML document, add this line.

```
<link rel="stylesheet" href="table.css">
```

The table styles will then be in force. Make sure the CSS file and your table example are in the same directory.

Let us now create this table.



The screenshot shows a web browser window titled 'Table Example'. The address bar shows a file path: file:///Users/morrison/... The browser displays a page with the heading 'Presidential Table Example'. Below the heading is a table with the following content:

Recent Presidents		
President	Ordinal	Term
Donald Trump	45	2017-pres.
Barack Obama	44	2009-2017
George W. Bush	43	2001-2019
William Clinton	42	1993-2001

To begin, we put in our table tags.

```
<table>
</table>
```

Now put in the top row. Note the use of the colspan property to get the top of the table to span all three columns.

```
<table>
  <tr><th colspan="3">Recent Presidents</th></tr>
</table>
```



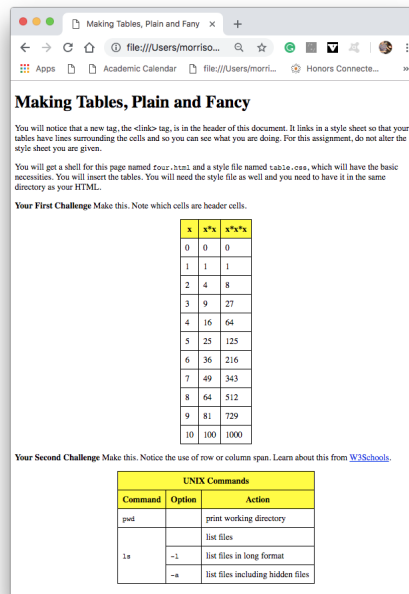
Next goes the header row.

```
<table>
  <tr><th colspan="3">Recent Presidents</th></tr>
  <tr><th>President</th><th>Ordinal</th> <th>Term</th></tr>
</table>
```

Finally, populate the rest of the cells.

```
<table>
  <tr><th colspan="3">Recent Presidents</th></tr>
  <tr><th>President</th><th>Ordinal</th> <th>Term</th></tr>
  <tr><td>Joe Biden</td><td>46</td> <td>2021-pres..</td></tr>
  <tr><td>Donald Trump</td><td>45</td> <td>2017-2021</td></tr>
  <tr><td>Barack Obama</td><td>44</td> <td>2009-2017</td></tr>
  <tr><td>George W. Bush</td><td>43</td> <td>2001-2009</td></tr>
  <tr><td>William Clinton</td><td>42</td> <td>1993-2001</td></tr>
  <tr><td>George H. Bush</td><td>41</td> <td>1989-1993</td></tr>
</table>
```

**Programming Exercise** Make this page.



## 10 Sniffing Around for More, and How to Avoid the Bad Kids on the Block

If you are looking at a website and see something interesting, use the View Source feature to see its HTML. You can attempt to reverse-engineer it and replicate it for yourself.

Remember, however, it's a jungle out there and there are evil influences that will violate standards. You will see that the HTML reference you explored in the programming exercises above talks about deprecated tags. Don't use them. The reference will tell you how to achieve the desired effect correctly. Many of these tags attempt to style a document in HTML. Don't do this. Remember: HTML is about structure, CSS is about appearance. Keep them separate.

- The `<u>` tag has been used for underlining text. This just confuses that text with a link. Do not do this. The effect can be achieved with CSS, but it clearly should be avoided.
- The `<big>` makes the text inside bigger. Clearly this is presentational. You will learn to use the `font-size` property in CSS instead. The same goes for `<small>`.
- The `<blink>` and `<marquee>` tags are just too vile to countenance. Just do not do it.
- The `<font>` tag is presentational. Use CSS.
- The `<center>` tag is presentational. Use `text-align:center` instead in a local style sheet.

### 10.1 I'm Innocent, Don't Frame Me!

You might see pages out there that use frames, but they have become increasingly rare. Frames make a hash of things. They confuse search engines (You want your page seen, don't you?), prevent people from bookmarking stuff on your site the find useful, and stymie screen readers used by people with limited vision. That makes your page less accessible. You can tell by the tenor of this paragraph that frames are a bad idea. They are obviated by CSS.

### 10.2 Son, you have some undesirable attributes

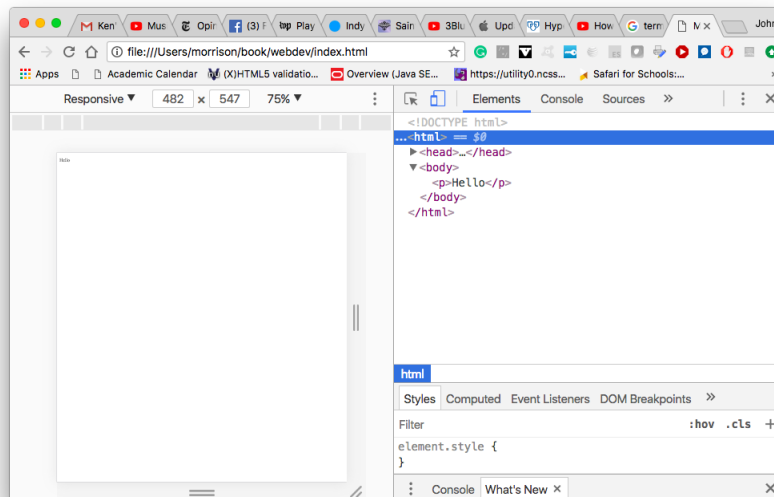
Another rogue's gallery is presented here; instead of tags, these are attributes you should not use. Most of this stuff can be done properly by using CSS. Again, we are trying to maintain the separation of style and structure.

- The `name` property is supplanted by `id`. It now only appears in form elements.

- The `text` property is supplanted the style command `color:someColor;`.
- The `bgcolor` property in the `body` tag is supplanted the style command `background-color:someColor;`.
- The `background` attribute in the `body` tag is supplanted the style command `background-image:url(someFileName);`.
- The `align` attribute is clearly presentational. Use CSS.
- The `target` opens a new tab on a user's browser; this is a breach of netiquette. It renders the back button inaccessible. Users can place your link in a new window themselves by right-clicking. You should leave this choice to the user.
- The `body` attributes `alink`, `vlink`, and `link` are all designed to control the colors of link text. This is clearly styling. In CSS, you will use `a:visited`, `a:active`, and `a:link` instead.

### 10.3 Exploring HTML files

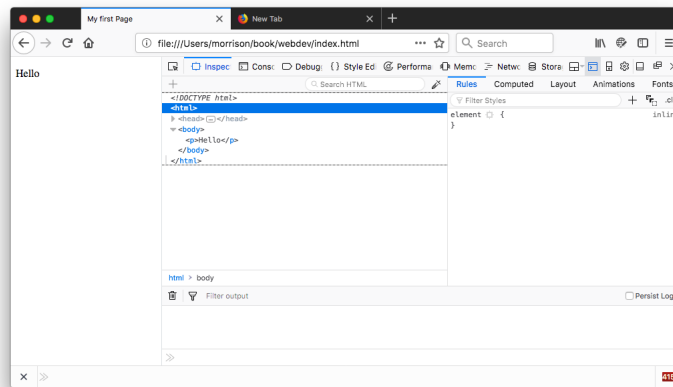
Open your index file on your server or local machine in Chrome. Then right-click on it and choose **Inspect** from the menu. You will see this.



The Elements tab allows you to explore your document and they will help you to visualize the document tree. Right now, it's pretty minimal but we will soon change that. Mouse over the paragraph element. The browser will highlight it and you will see stuff about margins, and padding. You will see that the paragraph element goes all the way across the page. This is so because the

paragraph is an example of a *block-level* element, which lives on a vertical chunk of a page.

You can also do this in Firefox. Click on the ≡ (aka “Hamburger Menu”) in the upper right hand corner of your browser window and choose the menu item Web Developer, then in the submenu choose Inspector. You can also right-click on the page and choose Inspect from the drop-down window. You will see this.



Click on the little triangle next to `body` in the Elements tab. Watch the element close. Click again to reopen it. You will be able to do this to explore the structure of your page.

Now let us introduce a grammatical error and see how Firefox handles it. Move the `meta` tag so it is the first line of the `head` element. Then, switch the closing tags for `title` and `head` so they close in the wrong order. Reload the page. It still looks the same. Now open the `head` element. Look inside the `title` element. You will see that the browser has corrected your bad HTML. Despite our incorrect HTML, the browser renders the document correctly and interprets it correctly. Now remove the error, save your index page, and reload the document. You will see that peace is restored to the kingdom.

Next, let us explore the body. Open it; you will see that its contents consist of a single paragraph with the word “Hello” in it. The indentation makes it clear that the `p`(paragraph) element lives inside the `body` element, which lives inside the `html` element. The `html` element is the it root element, since it contains all other elements on the page.

As you build your web pages, use this tool to explore the hierarchy of your documents. It can help you to avert errors. You should add some more elements to your page, save it, and explore them. These tools can help you debug pages that are not formatting to your liking. In combination with the validator, they constitute a very useful toolkit.

**A Programming Project** Here are two CSS files you will need. This is the now-familiar `tables.css`

```
table
{
    border-collapse:collapse;
    border: solid #000000 1px;
    margin-left:auto;
    margin-right:auto;
}
td, th
{
    border-collapse:collapse;
    border: solid #000000 1px;
    padding:.5em;
}
th
{
    background-color:#FFFF00;
}
```

You will also need this one, `hughJass.css`. These make table cells big.

```
tr, td
{
    height:75px;
    width:75px;
}
```

Use two `link` tags to include them on your page. You will then reproduce this.

Your job is to replicate me!

File | /Users/morrison/20... | (X)HTML5 validati... | The Future of Agi... | Overview (Java S...

## Replicate Me, Part 0

**Warning** I need to be valid. You need to subject me to the tender mercies of [Sister Mary Rapaknuckle](#) so I don't contain any crummy HTML.

**Resources** Here are places to look for help in crating me.

- [W3Schools](#) has tutorials and references on all the stuff we will do in this class. Do not neglect to use the "Try it" mechanism. It makes experimenting easy and it gives instant feedback!
- The [Mozilla Developer Network](#) has loads of great material that will help you in this class.
- The [Net Ninja](#)s has loads of great video tutorials on HTML; visit the search page and type HTML...

### Paths Through NCSSM's Computer Science Curriculum

Here's the skinny on learning to program at NCSSM.

1. If you have never programmed before, take one of these.
  - Web Development
  - Scientific Programming
  - Introduction to Robotics
  - Cryptography, a joint Math/CS course
 If you have, go to the next step. Be warned we will blithely assume you know how to program! We get into the deep end of the pool fast.
2. Take CS 424, Procedural Programming. This is an intermediate-level course in procedural programming. You build a lot of skill in this class. It involves a change of language from this class, likely Python or Ruby.
3. Take advanced offerings. We show the sequences here.
  - C: Learn the C language and a low-level interface to programming.
  - From C take Data Structures or Algorithms. This is a "booken or."

### HTML Special Character Entities

Here is a soupçon of some higfalutin' French! Auf English wir haben Cologne; auf Deutsch est is Köln. You will need a reservation to sit at table d' hôte at Madeline's Petite Paris. And you will need to have a mañana attitude on Jimmy Buffet's beach. We need to have a coordinated response to these challenges.

Here is a little sets education. When  $x$  belongs to a set  $A$ , we write  $x \in A$ ; otherwise we write  $x \notin A$ . If  $A$  and  $B$  are sets, and every element of  $A$  is present in  $B$ , we write  $A \subseteq B$ .

### We shall no Longer Table the Issue

No **goldbricking** Stack the gold bricks. You will need to link in the style file the file [table.css](#) and the file [hughJass.css](#).

S1	L1	S2
L2		L3
S3	L4	S4

### To the Weight Room for a Little Definition.

Learn about definition lists then do this.

**Arms**

- flat curls, 50 with 20 lbs each hand
- hammer curls, 50 with 20 lbs each hand
- King Kong's, 50 with a 100 lb stack
- Militaries on a ball, 50 with 30 lbs each hand
- Bench on a ball, 55 lbs each and, 100 reps
- Flees on a ball, 25 lbs each hand, 50 reps

**Legs**

- Squats, 40 lbs each hand, 50 reps
- Hamstring curls, 60 lbs, 100 reps
- Call raises at the barre: 50 each, first position, second position, toes pointed in

After a few weeks, you will develop some definition.

## 11 Terminology Roundup

Here is the new vocabulary we introduced in this chapter. You should know these terms by the time you finish reading this chapter.

**attribute** This is a property-value specification for a tag that modifies the tags default behavior. Example:

```
<meta charset="utf=8">
```

This specifies the character set of a page; in this case you are using English characters.

**base URL** This is the address of the main index file in your `public_html` directory.

**block-level** Block level elements specify vertical segments of a page. Examples include `p`, `ul`, and `ol`.

**browser** This is a piece of software that acts as a virtual computer. It understands the languages HTML, CSS, and JavaScript, and uses them to format and display interactive web pages. **CSS** This is a cascading style sheet, which governs page appearance.

**client** This is your computer that is browsing the web.

**client-side language** These are languages that are run by the client's browser: HTML, JavaScript, and CSS. This is as opposed to a server-side language such as PHP, which programmatically generates a web page's HTML.

**DNS cache** This is a storage place for translations of domain names to IP numbers. Your local machine and your ISP maintain DNS caches for recently-visited sites.

**Hypertext Transfer Protocol(HTTP)** This is the system by which HTML is transferred from a server to your local client.

**delimit** This means to tell where something begins and ends.

**directory** This is what we call a folder.

**element** This is material between two matching pairs of tags.

**file transfer protocol** This is a general purpose protocol for downloading all types of files from a server.

**host** This is a machine on the web that serves up web pages or files.

**IP Number** This is a unique number [Internet Protocol] assigned to any machine when it working on the Internet. IP numbers typically have an appearance similar to this 192.168.45.6. This number can vary from session to session.

**index page** This is the main page in a subdirectory of a website. Its name is typically `index.html`. On sites using PHP, you will see `index.php`.

**inline element** This is an element describing a texticographical region of a page. Inline elements may not appear directly in the body of a document; they

should be enclosed in some block-level element.

**JavaScript** This is a computer language that gives web pages behavior and interactivity.

**local style sheet** This is a list of style commands that can be applied directly to any element on a page. This list is the value for the **style** property of the element. Example:

```
<p style="color:red;background-color:yellow">Horse</p>
```

This gives you a horse of a different color.

**MAC Number** This is a unique, permanent ID assigned to a hardware device that is to be connected to the internet.

**markup language** This is a language that specifies structural elements on a page, which are subsequently interpreted and displayed by a piece of software. Examples: HTML and  $\LaTeX$  (mathematics).

**matching tags** This refers to a set of opening and closing tags that jointly delimit an element.

**packet** When a server receives an HTTP request, it breaks up the HTML, CSS, JavaScript and media on a web pages into little bits called packets. Each packet knows the IP address it came from, the IP address it is destined to, and how, with all of the other packets, to reassemble itself on the client's machine. These are the "bits" of MikeTV.

**parse** To extract meaning from symbols. The browser parses your HTML document to create the document tree.

**process** This refers to any program running on your computer.

**prompt** This is a signal by a computer program that it is ready to accept a command. The machine is "prompting" you it is ready to do work.

**proper nesting rule** This refers to the fact that elements on a page are either disjoint (do not overlap) or one element is entirely nested inside of the other. This is enforced by the HTML rules that say tags must be closed in the reverse order that they open, and that a closing tag must be matched by an opening tag of the same type.

**property** This is something attached to a tag that determines its behavior. For example in the tag

```

```

the properties are **src**, the file containing the image, and **alt**, the alt-text. The text is quotes after each property is the property's value.



**public subtree** This is the portion of your server account that is visible to the web. It lives in a directory typically named `public_html`.

**Robert Fulghum Rule** If you opened it, close it when you are done. If you took it out, put it back where it came from when you are finished with it. This comes from his book *All I Really Needed to Know I Learned in Kindergarten*.

**server** This is a computer offering content on the web.

**tag** This is a token that delimits an element in an HTML document.

**token** A token is an atom of meaning; if you take it apart it loses its original meaning. For example, a tag is a token. The tag `<html>` is an irreducible unit of meaning: it says, “begin an HTML document.”

**type** Tags have types; it is just the first piece of contiguous text inside of the `<` that begins a tag.

**text element** This is an inline element consisting of just text. Text elements are self-terminating.

**texticographical** This describes the left to right, up to down flow of text on a page.

**uniform resource locator** This is a location on the Internet. It can be absolute, in which case it begins with `http://` or `https://`, or it can be relative, in which case it is on the same server, and it might have just a file name or a system path.

**user agent** This is the default CSS used by the browser.

**value** This is something attached to a tag’s property. For example in the tag

```

```

the properties are `src`, the file containing the image, and `alt`, the alt-text. The text is quotes after each property is the property’s value.

**virtual computer** This is a computer created in software. Your browser is an example of one.

**web server** This is a program that serves content to the web. Typical examples include Apache and NGinX. These program act as users on the server. They receive HTTP requests, fetch the requested material, turn it in to packets, and send it off.

**References** The Net Ninja videos, [5], present the basics of HTML in a clear, succinct way. Videos 1-13 cover the material of this chapter and add a few new things in for you.

## References

- [1] Roald Dahl. *Charlie and the Chocolate Factory*. Alfred A. Knopf, Inc., 1964.
- [2] Apache Foundation. Apache website. <http://www.apache.org>.
- [3] Patrick Griffiths. Html dog. <https://htmldog.com>.
- [4] Mozilla. Mozilla tag reference. <https://developer.mozilla.org/en/HTML/Element>.
- [5] Net Ninja. The net ninja html. <https://www.youtube.com/playlist?list=PL4cUxeGkcC9ibZ2TSBaGGNrgh4ZgYE6Cc>.
- [6] Inc. Refsnes Data. W3schools. <http://www.w3schools.com>.