Before we begin: required software installations

Visual Studio Code (instructions: https://curriculum.dhinstitutes.org/installations/microsoft-visual-studio-code/windows/)

For Windows users, install Git and Git Bash (instructions: https://curriculum.dhinstitutes.org/installations/git-and-git-bash/windows/)

The following curriculum is adapted from https://curriculum.dhinstitutes.org/

What Is the Command Line?

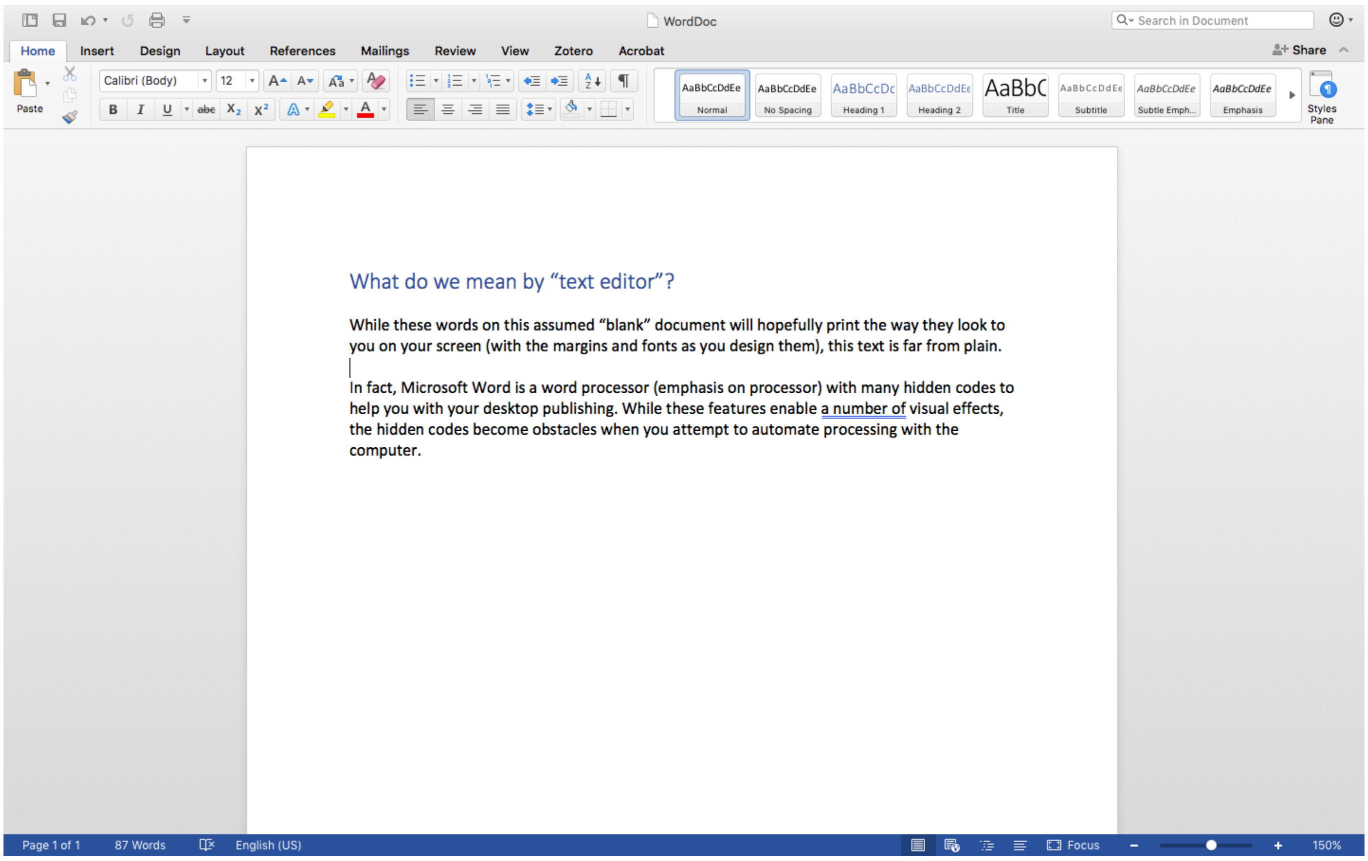
If asked to show someone who has never seen a computer how to do something on your computer, many of us would explain what a screen and a cursor are, and then show how to point and click on icons. This approach relies on a graphical user interface, or GUI (pronounced "gooey!"). Today we're going to explore another way to make your computer do things: through the command line. Instead of pointing and clicking, we'll be typing in either git bash (Windows) or terminal (macOS) to tell the computer directly what task we'd like it to perform.

The command line is a text-based way of interacting with your computer. You may hear it called different names, such as the terminal, the shell, or bash. In practice, you can use these terms interchangeably. (If you're curious, though, you can read more about them in the glossary.) The shell we use (whether terminal, shell, or bash) is a program that accepts commands as text input and converts commands into appropriate operating system functions.

The command line (of computers today) receives these commands as text that is typed in.

Plain text vs. formatted text

As scholars working with computers, we need to be aware of the ways plain text and formatted text differ. Words on a screen may have hidden formatting. Many of us learned to use a word processor like Microsoft Word and don't realize how much is going on behind the words shown on the screen. For the purposes of communicating with the computer and for easier movement between different programs, we need to use text without hidden formatting.



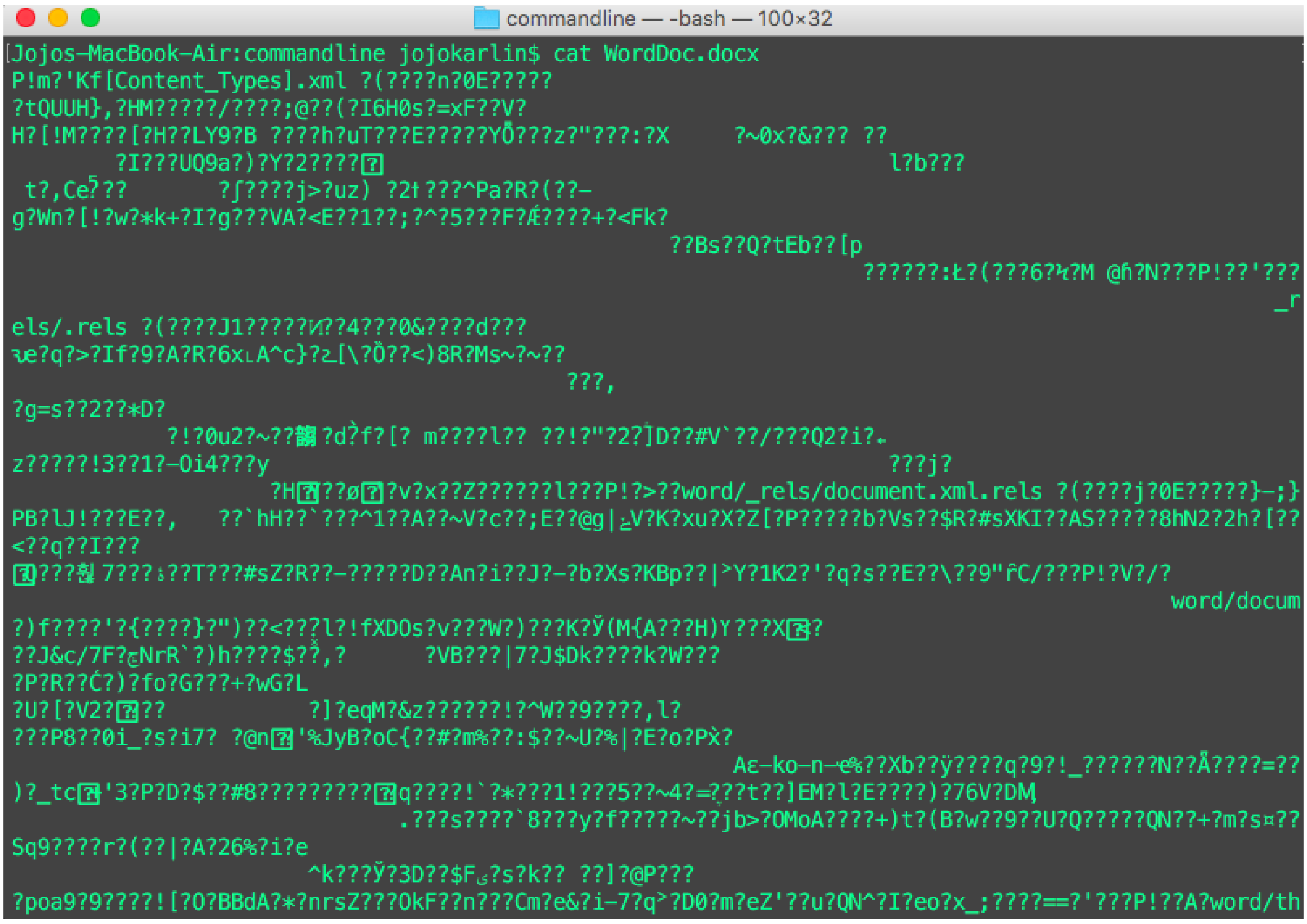
What do we mean by "text editor"?

While these words on this assumed "blank" document will hopefully print the way they look to you on your screen (with the margins and fonts as you design them), this text is far from plain.

In fact, Microsoft Word is a word processor (emphasis on processor) with many hidden codes to help you with your desktop publishing. While these features enable a number of visual effects, the hidden codes become obstacles when you attempt to automate processing with the computer.

Users with visual disabilities, click here to download the Word file.

If you ask the command line to read that file, this Word `.docx` file will look something like this



```
[Jojos-MacBook-Air:commandline jojokarlin$ cat WordDoc.docx
 P!m?'Kf[Content_Types].xml ?(????n?0E?????
 ?tQUUH},?HM?????/????;@??(?I6H0s?=xF??V?
 H?[!M????[?H??LY9?B ????h?uT???E?????YŐ???z?"???:?X     ?~0x?&??? ??
         ?I???UQ9a?)?Y?2????⍰                              l?b???
  t?,Ce??         ?ʃ????j>?uz) ?2ɫ???^Pa?R?(??-
 g?Wn?[!?w?*k+?I?g???VA?<E??1??;?^?5???F?Ǽ????+?<Fk?
                                                         ??Bs??Q?tEb??[p
                                                                          ??????:Ł?(???6?ʮ?M @ɦ?N???P!??'???
                                                                                                          _r
 els/.rels ?(????J1?????ɳ??4???0&????d???
 ɿe?q?>?If?9?A?R?6xʟA^c}?ƻ[\?Ŏ??<)8R?Ms~?~??
                                                ???,
 ?g=s??2??*D?
              ?!?0u2?~??鬸?d?f?[? m????l?? ??!?"?2?]D??#V`??/???Q2?i?.
 z?????!3??1?-Oi4???y                                    ???j?
                        ?H⍰??ø⍰?v?x??Z??????l???P!?>??word/_rels/document.xml.rels ?(????j?0E?????}-;}
 PB?lJ!???E??,    ??`hH??`???^1??A??~V?c??;E??@g|ƻV?K?xu?X?Z[?P?????b?Vs??$R?#sXKI??AS?????8hN2?2h?[??
 <??q??I???
 ⍰???칊 7???ꜟ??T???#sZ?R??-?????D??An?i??J?-?b?Xs?KBp??|˃Y?1K2?'?q?s??E??\??9"ȓC/???P!?V?/?
                                                                                              word/docum
 ?)f????'?{????}?")??<???l?!fXDOs?v???W?)???K?ў(M{A???H)Y???X⍰?
 ??J&c/7F?ȼNrR`?)h????$??,?      ?VB???|7?J$Dk????k?W???
 ?P?R??Ć?)?fo?G???+?wG?L
 ?U?[?V2?⍰??            ?]?eqM?&z??????!?^W??9????,l?
 ???P8??0i_?s?i7? ?@n⍰'%JyB?oC{??#?m%??:$??~U?%|?E?o?Px?
                                                           Aɛ-ko-n-e%??Xb??ÿ????q?9?!_??????N??Å????=??
 )?_tc⍰'3?P?D?$??#8?????????⍰q????!`?*???1!???5??~4?=???t??]EM?l?E????)?76V?DM,
                                  .???s????`8???y?f?????~??jb>?OMoA????+)t?(B?w??9??U?Q?????QN??+?m?s¤??
 Sq9????r?(??|?A?26%?i?e
                          ^k???ў?3D??$Fى?s?k?? ??]?@P???
 ?poa9?9????![?O?BBdA?*?nrsZ???OkF??n???Cm?e&?i-7?q˃?D0?m?eZ'??u?QN^?I?eo?x_;????==?'???P!??A?word/th
```

Users with visual disabilities, click here to download the text file.

Word documents which look like "just words!" are actually comprised of an archive of extensible markup language (XML) instructions that only Microsoft Word can read. Plain text files can be opened in a number of different editors and can be read within the command line.

Plain Text

For the purposes of communicating with machines and between machines, we need characters to be as flexible as possible.

Plain text shows its cards—if it's marked up, the markup will be human readable. Plain text can be moved between programs more fluidly and can respond to programmatic manipulations. Because it is not tied to a particular font or color or placement, plain text can be styled externally.

A counterpoint to plain text is rich text (sometimes denoted by the Microsoft rich text format `.rtf` file extension) or "enriched text" (sometimes seen as an option in email programs). In rich text files, plain text is elaborated with formatting specific to the program in which they are made.

Plain text has two main properties in regard to rich text:

> plain text is the underlying content stream to which formatting can be applied. Plain text is public, standardized, and universally readable.

Default Recommendation for a Text Editor

For our workshops, we will be using Visual Studio Code. Not only is Visual Studio Code free and open source, but it is also consistent across macOS, Windows, and Linux systems.

You will have downloaded Visual Studio Code according to the instructions on the installations page. We won't be using the editor a lot in this tutorial, so don't worry about getting to know the editor now. In other workshops we will discuss syntax highlighting and version control, which Visual Studio Code supports. For now we will get back to working in the command line itself.

Evaluation

What is the difference between a plain text document and a rich text document? (Select all that apply)

- Plain text contains no formatting, only line breaks and spacing.*
- Plain text cannot be marked up.
- Rich text is styled text, *i.e.,* plain text completed by information such as font size, format, and colors.*
- One can't determine whether there is a difference betweeen the two without looking at their content.

Why is the Command Line Useful?

Initially, for some of us, the command line can feel a bit unfamiliar. Why step away from a point-and-click workflow? By using the command line, we move into an environment where we have more minute control over each task we'd like the computer to perform. Instead of ordering your food in a restaurant, you're stepping into the kitchen. It's more work, but there are also more possibilities.

The command line allows you to...

- Easily automate tasks such as creating, copying, and converting files.
- Set up your programming environment.
- Run programs you create.
- Access the (many) programs and utilities that do not have graphical equivalents.
- Control other computers remotely.

In addition to being a useful tool in itself, the command line gives you access to a second set of programs and utilities and is a complement to learning programming.

What if all these cool possibilities seem a bit abstract to you right now? That's alright! On a very basic level, most uses of the command line are about **showing information** that the computer has, or **modifying or making** things (files, programs, etc.) on the computer.

In the next section, we'll make this a little more clear by getting started with the command line.

Getting to the Command Line

macOS

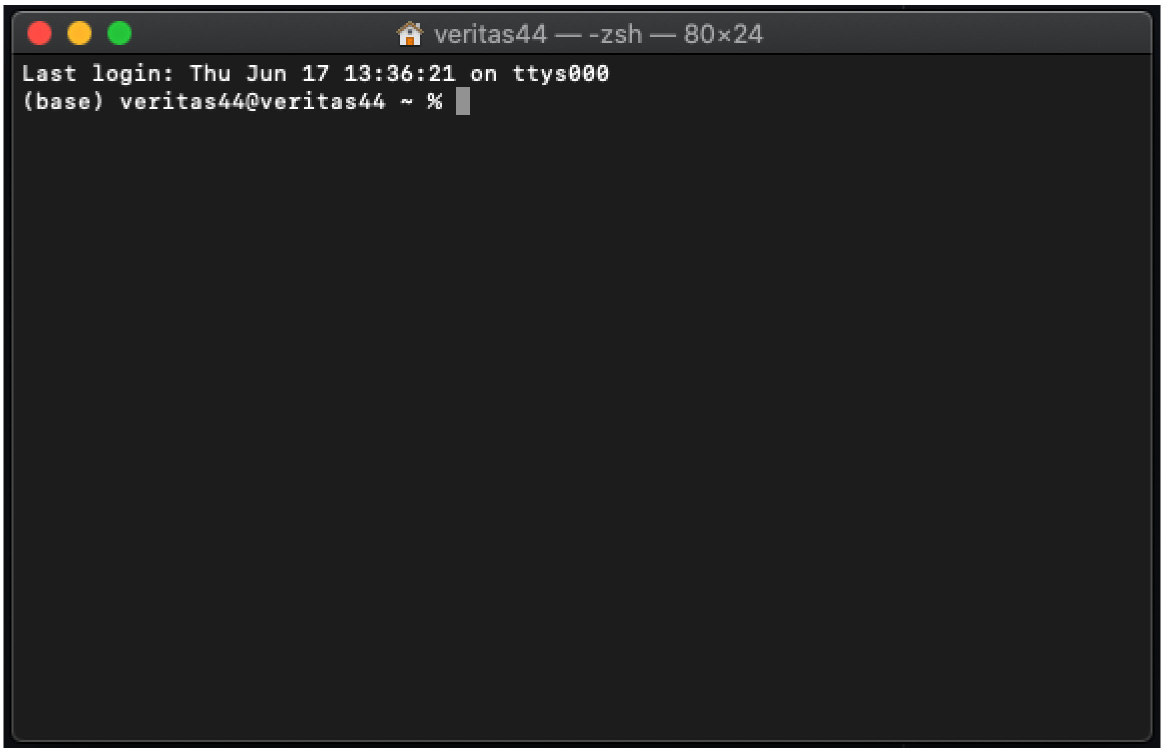
If you're using macOS:

1. Click the Spotlight Search button (the magnifying glass) in the top right of your desktop.

2. Type `terminal` into the bar that appears.

3. Select the first item that appears in the list.

4. When the Terminal pops up, you will likely see either a window with black text over white background or colored text over a black background.



```
Last login: Thu Jun 17 13:36:21 on ttys000
(base) veritas44@veritas44 ~ %
```

Please note: You can change the color of your Terminal or BashShell background and text by selecting `Shell` from the top menu bar, then selecting a theme from the menu under `New Window`.

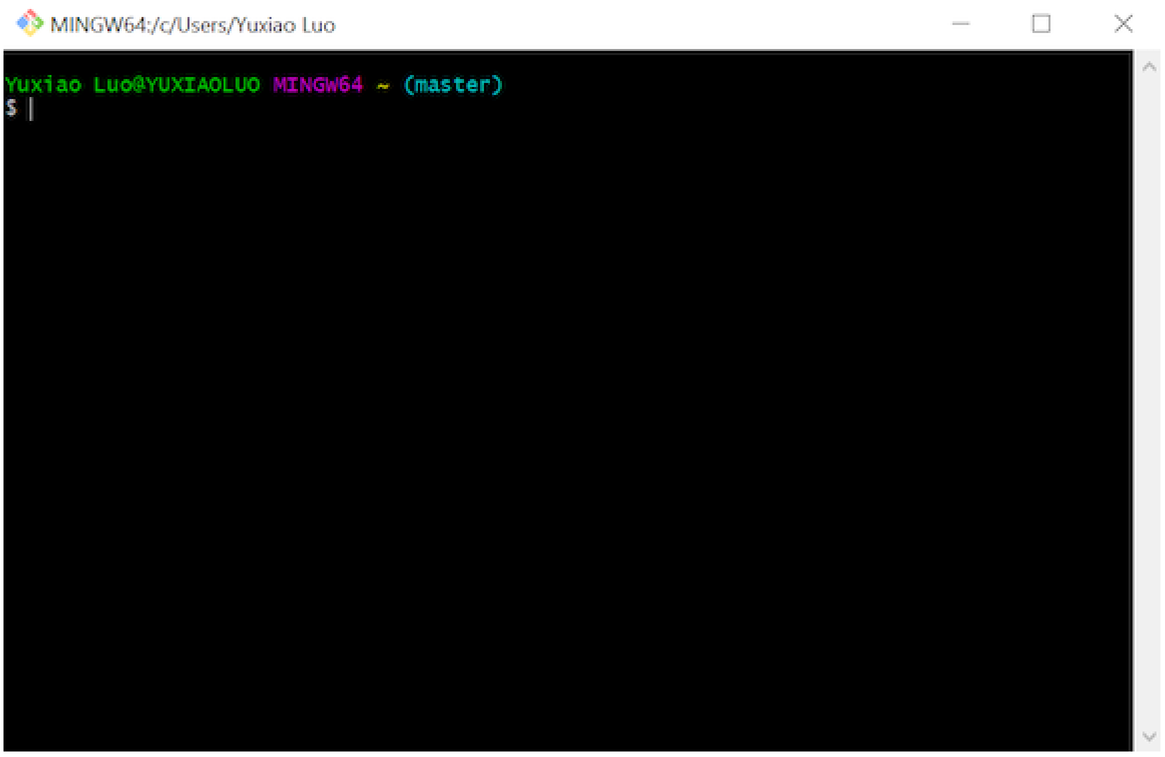
Bonus points: if you really want to get the groove of just typing instead of pointing and clicking, you can hold the `command (⌘)` key while and press `space` to pull up Spotlight search, start typing `Terminal,` and then hit `enter` to open a terminal window. This will pull up a terminal window without touching your mousepad. For super bonus points, try to navigate like this for the next fifteen minutes, or even the rest of this session—it is tricky and sometimes a bit tiring when you start, but you can really pick up speed when you practice!

Windows

We won't be using Windows's own non-UNIX version of the command line. Instead, we will use Git Bash. If you haven't installed it yet, you can follow these instructions. The reason we use Git Bash as the command line on Windows is that it makes you able to run the same commands as you would on a computer running macOS or Linux. Git Bash includes core utilities available on Linux that are not available on Windows.

1. Look for Git Bash in your programs menu and open.
2. If you can't find the git folder, just type `git bash` in the search box and select `git bash` when it appears.
3. Open the program.
4. When the terminal pops up, you will likely see either a window with black text over white background or colored text over a black background.You know you're in the right place when you see the `$`.

Note that the sign for you being in the right place might also be a `%` or a `#` depending on your operating system.



```
Yuxiao Luo@YUXIAOLUO MINGW64 ~ (master)
$ |
```

Bonus points: if you really want to get the groove of just typing instead of pointing and clicking, you can press `windows` to open the Start menu, start typing `git bash` and then hit `enter` to open a git bash window. This will pull up a command window without touching your mousepad.

Command Prompt `$`

`$`, which we will refer to as the "command prompt," is the place you type commands you wish the computer to execute. We will now learn some of the most common commands.

When you see the `$`, you're in the right place. As noted above, however, the sign varies somewhat between systems, and sometimes the sign is a `%` or a `#`. We call the sign the *command prompt*; it lets us know the computer is ready to receive a command.

In the following lessons, we will refer to the command prompt using a `$`. Just make a note now of your sign, if it differs from the dollar sign. You will be able to follow along just fine as long as you understand that they all are different ways of knowing that you are "at the *command prompt*."

Prefatory Pro Tips

Before we get started, I wanted to give you a couple of tips of things to keep in mind.

First, go slow at first and check your spelling! *Keep this in mind!* If at first something doesn't work, check your spelling! Unlike in human reading, where letters operate simultaneously as atomistic symbols and as complex contingencies (check Johanna Drucker on the alphabet), in coding, each character has a discrete function including (especially!) spaces.

Second, keep in mind that the command line and file systems on macOS and Unix are usually pre-configured as cAsE-pReSeRvInG—so capitalizations also matter when typing commands and file and folder names.

Third, while copying and pasting from this handy tutorial may be tempting to avoid spelling errors and other things, we encourage you not to! Typing out each command will help you remember them and how they work.

Now, we are ready to get started.

Navigation

Getting started: know thyself

You may also see your username to the left of the command prompt `$`. Let's try our first command. Type the following and press `enter` on your keyboard:

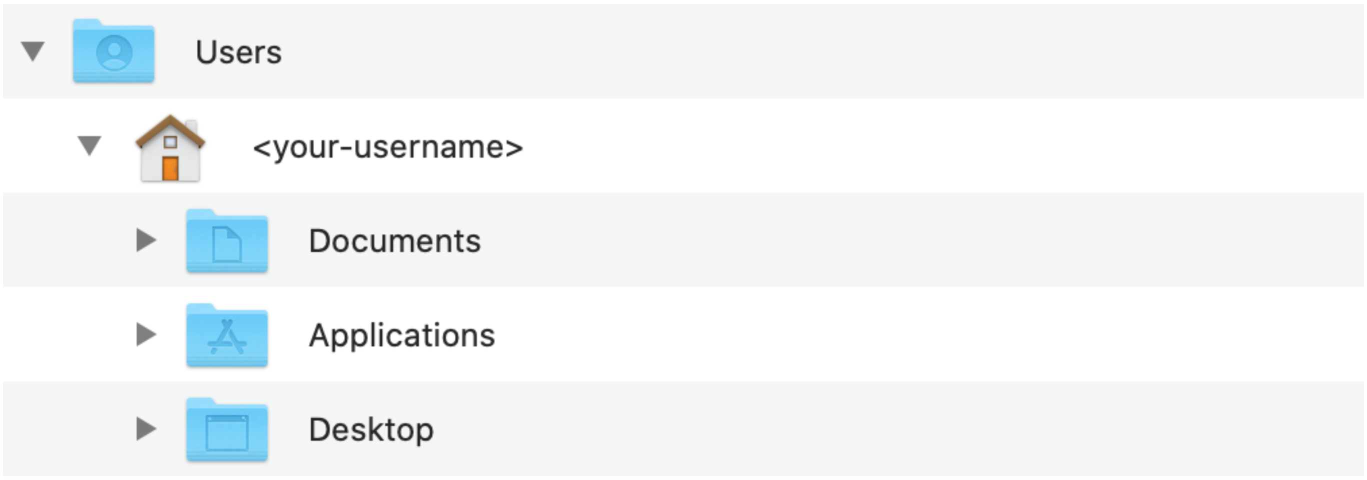
```
$ whoami
```

The `whoami` command should print out your username. Congrats, you've executed your first command! This is a basic pattern of use in the command line: type a command, press `enter` on your keyboard, and receive output.

Orienting Yourself in the Command Line: Folders

OK, we're going to try another command. But first, let's make sure we understand some things about how your computer's filesystem works.

Your computer's files are organized in what's known as a hierarchical filesystem. That means there's a top level or `root` folder on your system. That folder has other folders in it, and those folders have folders in them, and so on. You can draw these relationships in a tree:



The root or highest-level folder on macOS is just called `/`. We won't need to go in there, though, since that's mostly just files for the operating system. On Windows, the root directory is usually called `C:`. (If you are curious why `C:` is the default name on Windows, you can read about it here.)

Note that we are using the word "directory" interchangeably with "folder"—they both refer to the same thing.

OK, let's try a command that tells us where we are in the filesystem:

```
$ pwd
```

You should get output like `/Users/your-username`. That means you're in the `your-username` directory in the `Users` folder inside the `/` or root directory. This directory is often called the "home" directory.

On Windows, your output would instead be `C:/Users/your-username`. The folder you're in is called the working directory, and `pwd` stands for "print working directory." "Print" as a word can be somewhat misleading. The command `pwd` won't actually print anything except on your screen. This command is easier to grasp when we interpret "print" as "display."

Now we know "where" we are. But what if we want to know what files and folders are in the `your-username` directory, a.k.a. the working directory? Try entering:

```
$ ls
```

You should see a number of folders, probably including `Documents`, `Desktop`, and so on. You may also see some files. These are the contents of the current working directory. `ls` will "list" the contents of the directory you are in.

Wonder what's in the `Desktop` folder? Let's try navigating to it with the following command:

```
$ cd Desktop
```

The `cd` command lets us "change directory." (Make sure the "D" in "Desktop" is capitalized.) If the command was successful, you won't see any output. This is normal—often, the command line will succeed silently.

So how do we know it worked? That's right, let's use our `pwd` command again. We should get:

```
$ pwd
/Users/your-username/Desktop
```

Now try `ls` again to see what's on your desktop. These three commands—`pwd`, `ls`, and `cd`—are the most commonly used in the terminal. Between them, you can orient yourself and move around.

One more command you might find useful is `cd ..` which will move you one directory up in the filesystem. That's a `cd` with two periods after it:

```
$ cd ..
```

When you're done, come back to your "home" folder with

```
$ cd ~
```

(That's a tilde `~`, on the top left of your keyboard.)

Evaluation

What command do you run if you are trying to identify where in the filesystem you are currently located/working?

- $ `ls`
- $ `pwd`*
- $ `cd`
- $ `whoami`

When and why would you want to use the command line as opposed to your operating system's GUI?

Keywords

Do you remember the glossary terms from this section?

- Filesystem
- GUI

- Root

Creating Files and Folders

Creating a File

So far, we've only performed commands that give us information. Let's use a command that creates something on the computer.

First, make sure you're in your home directory:

```
$ pwd
/Users/your-username
```

Let's move to the `Desktop` folder, or "change directory" with `cd`:

```
$ cd Desktop
```

Once you've made sure you're in the `Desktop` folder with `pwd`, let's try a new command:

```
$ touch foo.txt
```

The `touch` command is used to create a file without any content. This command can be used when you don't have any data yet to store in it.

If the command succeeds, you won't see any output. Now move the terminal window and look at your "real" desktop, the graphical one. See any differences? If the command was successful and you were in the right place, you should see an empty text file called `foo.txt` on the desktop. Pretty cool, right?

Handy Tip: Up Arrow

Let's say you liked that `foo.txt` file so much you'd like another! In the terminal window, press the `up arrow` on your keyboard. You'll notice this populates the line with the command that you just wrote. You can hit `enter` to create another `foo.txt,` (note - `touch` command will not overwrite your document nor will it add another document to the same directory, but it will update info about that file.) or you could use your left/right arrows to move the insert cursor around on the screen so you can, for instance, change the file name to `foot.txt` to create a different file.

As we start to write more complicated and longer commands in our terminal, the `up arrow` is a great shortcut so you don't have to spend lots of time typing.

Creating Folders

OK, so we're going to be doing a lot of work during the Digital Humanities Research Institute. Let's create a `projects` folder on our desktop, where we can keep all our work in one place.

First, let's check to make sure we're still in the `Desktop` folder with `pwd`:

```
$ pwd
/Users/your-username/Desktop
```

Once you've double-checked you're in `Desktop`, we'll use the `mkdir` or "make directory" command to make a folder called `projects`:

```
$ mkdir projects
```

Now run `ls` to see if a projects folder has appeared. Once you confirm that the projects folder was created successfully, `cd` into it.

```
$ cd projects
$ pwd
/Users/your-username/Desktop/projects
```

OK, now you've got a projects folder that you can use throughout the Institute. It should be visible on your graphical desktop, just like the `foo.txt` file we created earlier.

Evaluation

What does the `up arrow` command do?

- It quits the Terminal/GitBash.
- It undoes my last command.
- It inserts my last command.*
- It shows me what folder I am working in.

Creating a Cheat Sheet

In this section, we'll create a text file that we can use as a cheat sheet. You can use it to keep track of all the awesome commands you're learning.

Echo

Instead of creating an empty file like we did with `touch`, let's try creating a file with some text in it. But first, let's learn a new command: `echo`.

```
$ echo "Hello from the command line"
Hello from the command line
```

Redirect (`>`)

By default, the echo command just prints out the text we give it. Let's use it to create a file with some text in it:

```
$ echo "This is my cheat sheet" > cheat-sheet.txt
```

Now let's check the contents of the directory:

```
$ pwd
/Users/your-username/projects
$ ls
cheat-sheet.txt
```

OK, so the file has been created. But what was the `>` in the command we used? On the command line, a `>` is known as a "redirect." It takes the output of a command and puts it in a file. Be careful, since it's possible to overwrite files with the `>` command.

If you want to add text to a file but *not* overwrite it, you can use the `>>` command, known as the redirect and append command, instead. If there's already a file with text in it, this command can add text to the file *without* destroying and recreating it.

`Cat`

Let's check if there's any text in `cheat-sheet.txt`.

```
$ cat cheat-sheet.txt
This is my cheat sheet
```

As you can see, the `cat` command prints the contents of a file to the screen. `cat` stands for "concatenate," because it can link strings of characters or files together from end to end.

A Note on File Naming

Your cheat sheet is titled `cheat-sheet.txt` instead of `cheat sheet.txt` for a reason. Can you guess why?

Try to make a file titled `cheat sheet.txt` and observe what happens.

Now imagine you're attempting to open a very important data file using the command line that is titled `cheat sheet.txt`

For your digital best practices, we recommend making sure that file names contain no spaces—you can use creative capitalization, dashes, or underscores instead. Just keep in mind that the macOS and Unix file systems are usually pre-configured as cAsE-pReSeRvInG, which means that capitalization matters when you type commands to navigate between or do things to directories and files. You may also want to avoid using periods in your file names, as they sometimes can prompt you to confuse them with system files or file extensions (e.g., the full name of a PDF file is usually `file.pdf`).

Using a Text Editor

The challenge for this section will be using a text editor, specifically Visual Studio Code (install guide here), to add some of the commands that we've learned to the newly created cheat sheet. Text editors are programs that allow you to edit plain text files, such as `.txt`, `.py` (Python scripts), and `.csv` (comma-separated values, also known as spreadsheet files). Remember not to use programs such as Microsoft Word to edit text files, since they add invisible characters that can cause problems.

Exercise

You *could* use the GUI to open your Visual Studio Code text editor—from your programs menu, via Finder or Applications or Launchpad in macOS, or via the Windows button in Windows—and then click `File` and then `Open` from the drop-down menu and navigate to your Desktop folder and click to open the `cheat-sheet.txt` file.

Or, you can open that specific `cheat-sheet.txt` file in the Visual Studio Code text editor directly from the command line! Let's try that by using the `code` command followed by the name of your file in the command line. (Please note the command `code` prompts your computer to open Visual Code only if you have correctly completed the software configuration during installation.)

Once you've got your cheat sheet open in the Visual Studio Code text editor, type to add the commands we've learned so far to the file. Include descriptions about what each command does. Remember, this cheat sheet is for you. Write descriptions that make sense to you or take notes about questions.

Save the file.

Once you're done, check the contents of the file on the command line with the `cat` command followed by the name of your file.

Solution

- Step 1

  ```
  $ code cheat-sheet.txt
  ```

- Step 2

  ```
  $ cat cheat-sheet.txt
  My Institute Cheat Sheet
  ```

```
ls
lists files and folders in a directory

cd ~
change directory to home folder

...
```

Evaluation

What does effect does the following command produce?

```
$ echo "Hello! My Name is Mark!" > introduction.txt
```

- It adds the line "Hello! My Name is Mark!" to the existing content of the `introduction.txt` file.
- It checks whether the content of the `introduction.txt` file contains the line "Hello! My Name is Mark!"
- It replaces the content of the `introduction.txt` file with the line "Hello! My Name is Mark!"*
- None of the above.

Creating Syllabus Content Using Markdown

Markdown allows us to format textual features like headings, emphasis, links, and lists in a plain text file using a streamlined set of notations that humans can interpret without much training. Markdown files usually have a `.md` extension. We'll be using **Markdown** to write a syllabus.

Markdown is a markup language for formatting text. Like HTML, you add markers to plain text to style and organize the text of a document.

Markdown has fewer options for marking text than HTML. It was designed to be easier to write and edit.

In Markdown, we insert headings with a single hash mark like this:

```
# My Syllabus Heading
```

A sub-heading (H2) heading uses two hash marks like this:

```
## Readings
```

The lessons of this workshop were originally written in markdown. You can see here what they look like in their raw, unrendered form.

Compare that with this—the source code for this lesson's web page, written in HTML here.

Markdown is also arguably more sustainable and accessible than formats like `.docx` because of its simplicity and related ability to be read across multiple platforms. Use of Markdown is also supported by document-conversion tools like Pandoc that can change a markdown file to an `.epub` with one command entered into your terminal.

Here are a few more key elements to get you ready to make your own syllabus in Markdown.

To provide emphasis, place asterisks around some text:

```
*This text will appear italicized.*
**This text will appear bold.**
```

For emphasis, you need to mark where it should start and where it should end, so you need astrisks at the beginning and end of whatever text is being emphasized.

To create a bulleted list, put a hyphen at the beginning of each list item:

```
- Reading one
- Reading two
- Reading three
```

To create a link, put the anchor text (the text you will see) in square brackets and the URL in parentheses, directly following the anchor text in brackets. Don't put a space between them:

```
I teach at [The University of Miami](www.miami.edu).
```

Paragraphs of text are denoted by putting a blank line between them:

```
This is a paragraph in markdown. It's separated from the paragraph below with a
blank line. If you know HTML, it's kind of like the <p> tag. That means that there
is a little space before and after the paragraph when it is rendered.

This is a second paragraph in markdown, which I'll use to tell you what I like
about markdown. I like markdown because it looks pretty good, if minimal, whether
you're looking at the rendered or unrendered version. It's like tidy HTML.
```

Creating a Syllabus file

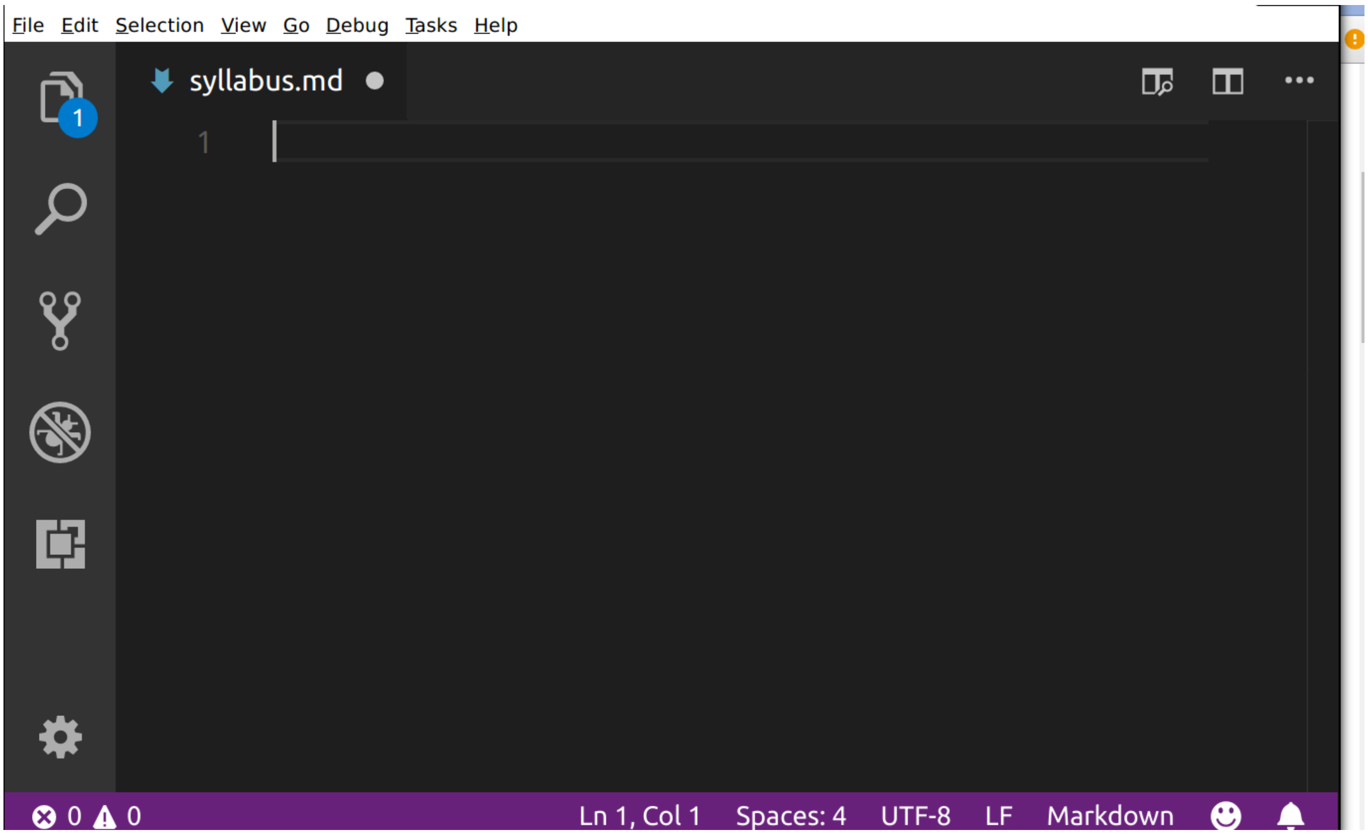
To create a plain text file, we're going to switch to our text editor, Visual Studio Code, to create and edit a file named `syllabus.md` and save it to our `projects` folder. The `.md` extension indicates that it is a Markdown file.

In terminal, check to make sure you are in your `projects` folder. (*Hint*: use `pwd` to see what directory you are currently in.)

Next, open the `syllabus.md` file in Visual Studio Code using:

```
$ code syllabus.md
```

You should see a window appear that looks similar to this:



If Visual Studio Code does not open when you use the `code` command in your terminal, open it using the Start Menu on Windows or Spotlight Search on macOS as you would any other software. Then click `File > Open File` and use the dialog to navigate to the `/Users/<your-name>/Desktop/projects/git` folder and create a `syllabus.md` file there.

We'll be typing our markdown into this file in the Visual Studio Code window. At any time, you can save your file by hitting `control + s` on Windows or `⌘ + s` on macOS. Alternatively, you can click the `File` menu on the top right, then select `Save` from the dropdown menu.

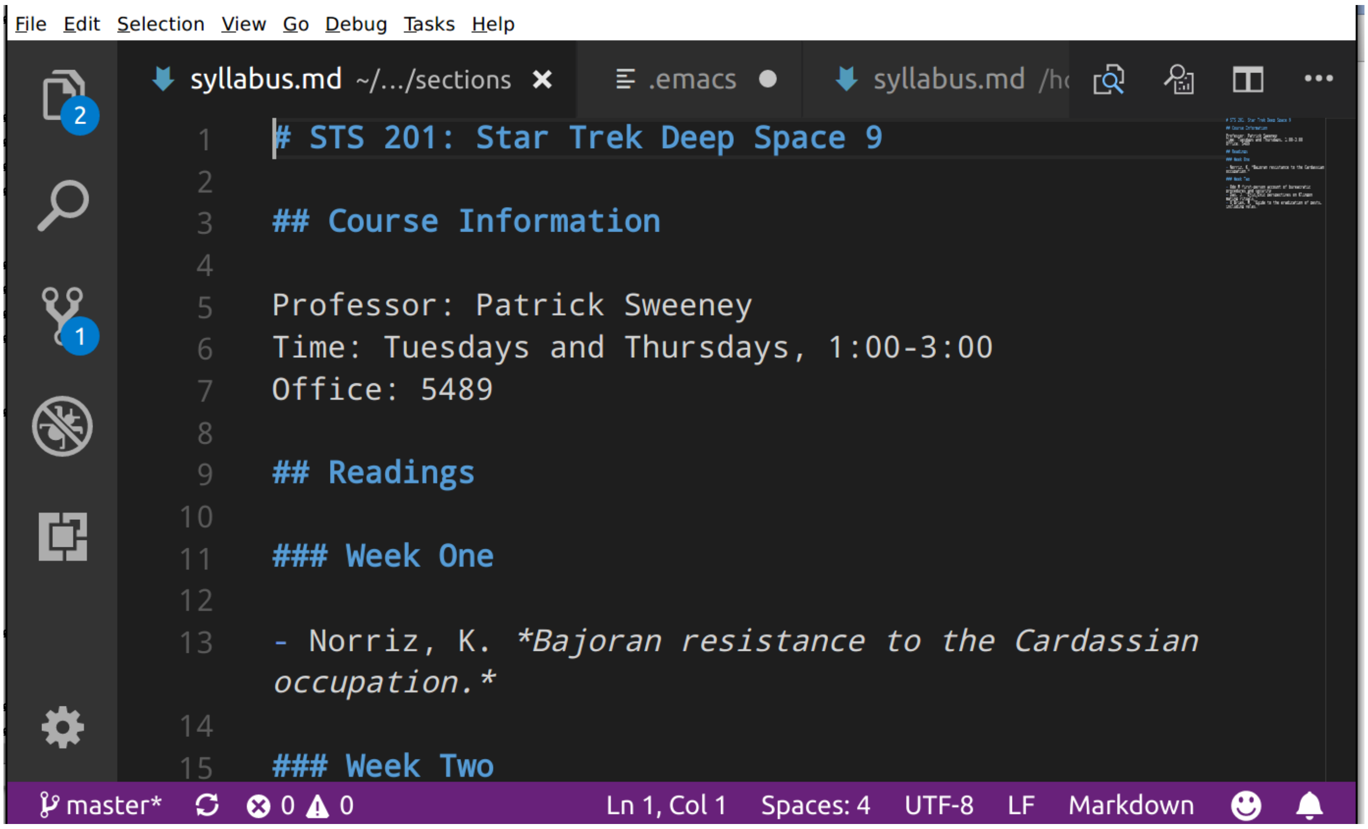
Exercise

Use these five elements—headings, emphasis, lists, links, and paragraphs—to create a syllabus. Have a main heading that gives the course title (one `#`), then subheadings for, at least, course info and readings. Use emphasis (`*`) for book titles and try to get a list in there somewhere.

If you want to learn more, we highly recommend this Markdown cheatsheet to learn additional markdown elements and add some extra features like images, blockquotes, or horizontal rules.

Example

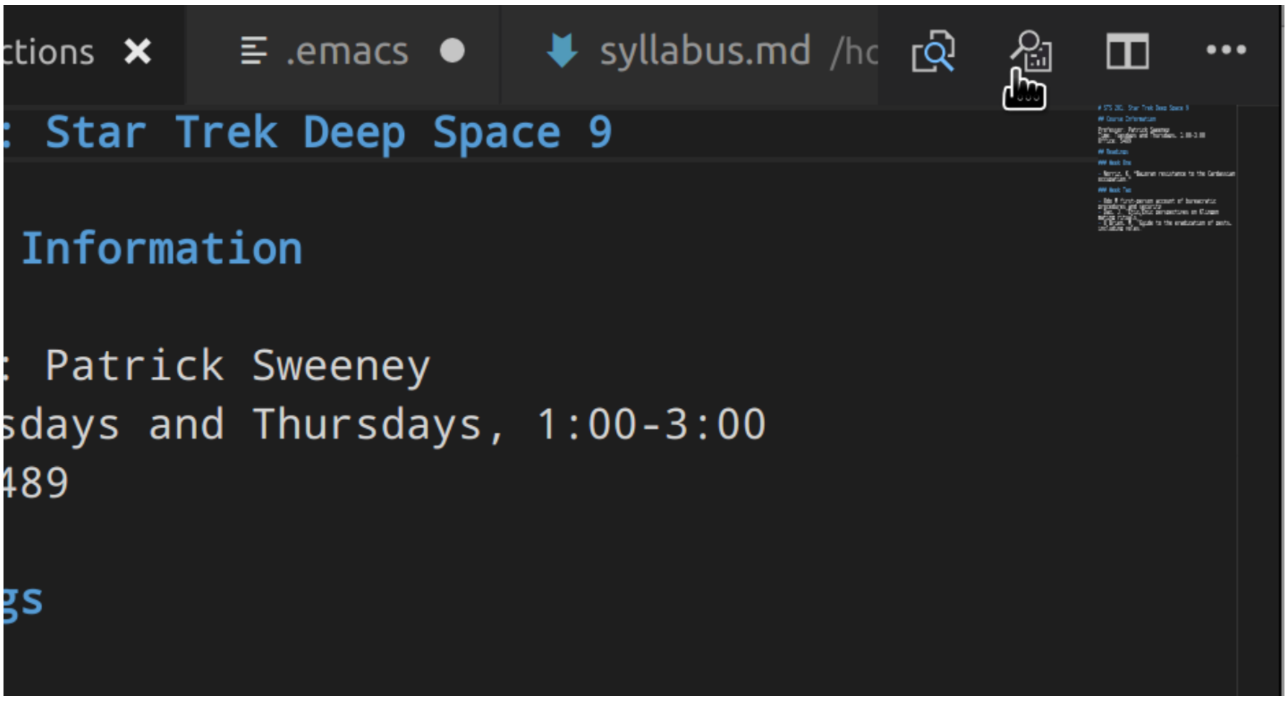
You can look at an example syllabus in raw text form here. When it's rendered by GitHub, it looks like this. When editing the markdown file in Visual Studio Code, it might look like this:


File Edit Selection View Go Debug Tasks Help
syllabus.md ~/.../sections
.emacs
syllabus.md /hc
master*
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Markdown

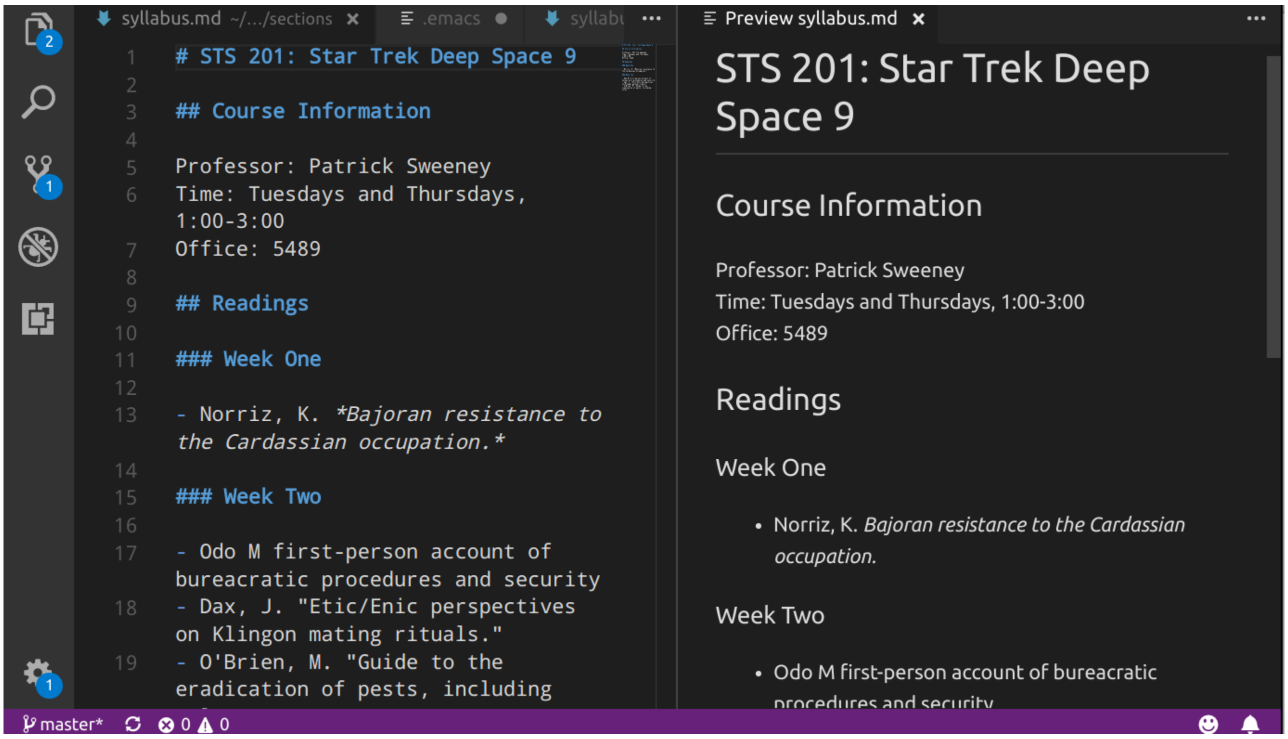
```
# STS 201: Star Trek Deep Space 9

## Course Information

Professor: Patrick Sweeney
Time: Tuesdays and Thursdays, 1:00-3:00
Office: 5489

## Readings

### Week One

- Norriz, K. *Bajoran resistance to the Cardassian occupation.*

### Week Two
```

Tips

1. Visual Studio Code also has a preview feature for your markdown. Hit the preview button on the top right while editing your markdown file:



You'll get two side-by-side panels. Your markdown file will be on the left, and your rendered preview will be on the right:



```
# STS 201: Star Trek Deep Space 9

## Course Information

Professor: Patrick Sweeney
Time: Tuesdays and Thursdays, 1:00-3:00
Office: 5489

## Readings

### Week One

- Norriz, K. *Bajoran resistance to the Cardassian occupation.*

### Week Two

- Odo M first-person account of bureacratic procedures and security
- Dax, J. "Etic/Enic perspectives on Klingon mating rituals."
- O'Brien, M. "Guide to the eradication of pests, including
```

STS 201: Star Trek Deep Space 9

Course Information

Professor: Patrick Sweeney
Time: Tuesdays and Thursdays, 1:00-3:00
Office: 5489

Readings

Week One

- Norriz, K. *Bajoran resistance to the Cardassian occupation.*

Week Two

- Odo M first-person account of bureacratic procedures and security

2. Remember to save your work—regularly!—with `control` + `s` on Windows or ⌘ + `s` on macOS.

Evaluation

Which best describes Markdown:

- a software installed on my local machine
- a language for formatting plain text files*
- a language that can be read and rendered by some web-based platforms*
- a version control software
- a cloud-based software
- refers to project folders as "repositories"

Introduction to HTML

Websites seem like these magical things that appear when we open our web browser (i.e. Chrome, Firefox, Safari). We know that websites are hypertext, meaning that we can click between links, travelling from page to page until we find what we need. What may be less obvious about websites is that, fundamentally **websites are plain text documents**, usually written in HTML or another web-based markup language, such as XML or XHTML.

Fun fact: **More than 90% of all websites (whose markup language we know) use HTML** (w3techs.com).

Hypertext Markup Language (HTML)

HTML is a markup language used to write web-based documents. It enables us to provide web browsers with information about the *content* of a document. We can, for example, indicate that some part of our document is a paragraph, image, heading, or link. The browser uses this information when displaying the document for users.

Markup Language vs. Programming Language

HTML is a *markup* language, not a programming language. Programming languages are used to transform data, by creating scripts that organize an output of data based on a particular input of data. A markup language is used to control the presentation of data.

For a practical example of this difference, we can think about tables. A programming language can help you search through a table, understand the kinds of data the table includes, find particular data points, or transform its content into other kinds of data, such as frequencies. A markup language would instead determine the content, layout, and visual presentation of the table.

Fundamentally, then, a script or program is a set of instructions given to the computer. A document in a markup language determines how information is presented to a user.

NOTE—Markup vs Markdown: Markdown and HTML are both types of markup languages; Markdown is a play on words. Markup languages help format content.

Cascading Style Sheets (CSS)

CSS is usually used in conjunction with HTML. HTML tells the browser what the different parts of a document *are*. CSS tells the browser what the parts of the document should *look like*. It is essentially a set of rules that are applied when rendering an HTML document. Its name—Cascading Style Sheets—refers to the fact that there is an order of precedence in how the browswer applies CSS rules to a document. More specific rules overwrite less specific rules.

Where Does the Internet Come In?

Together, these languages can be used to write and style a website using a text editor (such as Visual Studio Code) directly from your computer. No internet access needed.

However, internet access is necessary if you plan on making your website available to the public. Click here to learn how to get your website from your local computer onto the internet.

In our activities during this workshop we will focus on building locally-hosted websites. These are websites that you can open on your web browser, however, they only exist on your own device and are only accessible to you. Locally-hosted websites are not yet on the internet.

Evaluation

True or False: The primary difference between markup languages and programming languages is that markup languages are used to determine the format, appearance, and purpose of content, whereas programming languages are used to transform data.

- True*
- False

Keywords

Do you remember the glossary terms from this section?

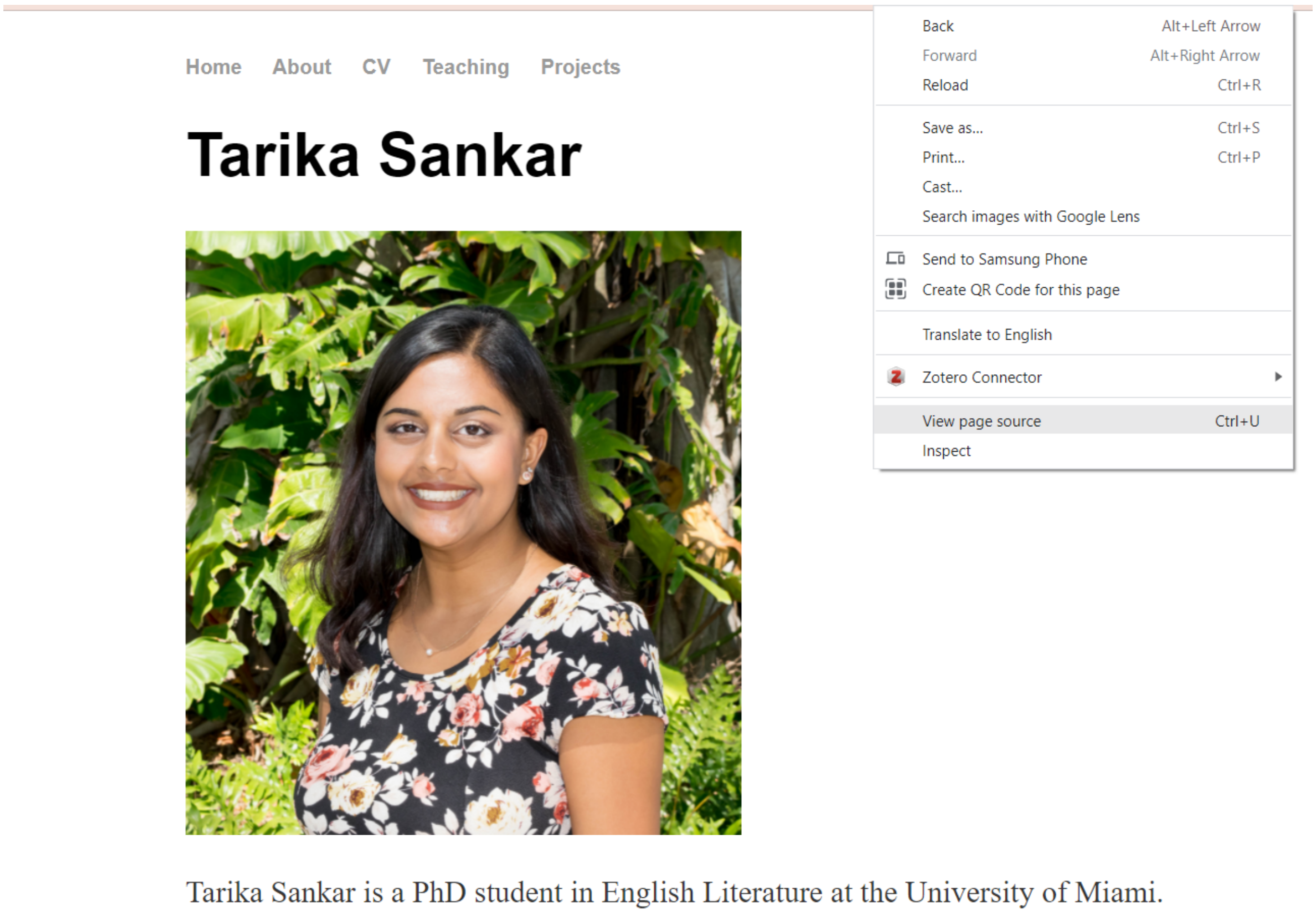
- CSS

- HTML
- Markdown
- Programming Language

Opening Activity

***Note*: please use Firefox or Chrome. Safari will not allow you to complete this activity.**

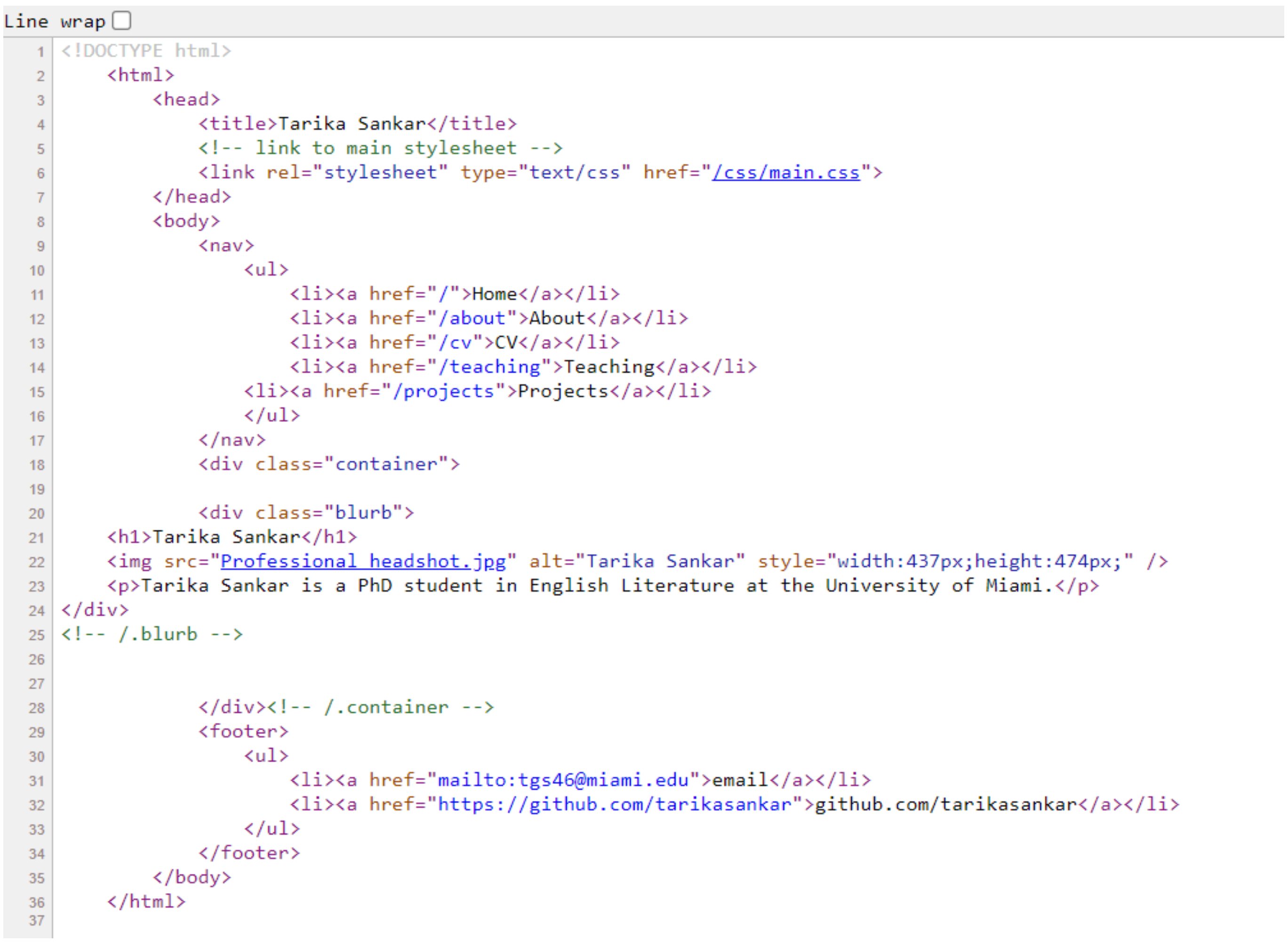
1. Open a web browser, preferably Firefox.
2. Go to any website. The example below is from tarikasankar.github.io.
3. Open the secondary menu (using a mouse, this would be the menu that opens when you right click on the page; on Mac computers, this is usually a two-finger tap on the track pad, or you can press the `control` button then click the track pad).
4. Select `View Page Source` from the dropdown menu.



Tarika Sankar is a PhD student in English Literature at the University of Miami.

What You Are Seeing

A second tab should open in your browser displaying the underlying code of the page. This is the code that is used to make and render the page in your web browser.



```
<!DOCTYPE html>
    <html>
        <head>
            <title>Tarika Sankar</title>
            <!-- link to main stylesheet -->
            <link rel="stylesheet" type="text/css" href="/css/main.css">
        </head>
        <body>
            <nav>
                <ul>
                    <li><a href="/">Home</a></li>
                    <li><a href="/about">About</a></li>
                    <li><a href="/cv">CV</a></li>
                    <li><a href="/teaching">Teaching</a></li>
                <li><a href="/projects">Projects</a></li>
                </ul>
            </nav>
            <div class="container">

            <div class="blurb">
    <h1>Tarika Sankar</h1>
    <img src="Professional headshot.jpg" alt="Tarika Sankar" style="width:437px;height:474px;" />
    <p>Tarika Sankar is a PhD student in English Literature at the University of Miami.</p>
</div>
<!-- /.blurb -->


            </div><!-- /.container -->
            <footer>
                <ul>
                    <li><a href="mailto:tgs46@miami.edu">email</a></li>
                    <li><a href="https://github.com/tarikasankar">github.com/tarikasankar</a></li>
                </ul>
            </footer>
        </body>
    </html>

```

In this workshop, we are going to learn how to read and write this code, and render it in the browser on your local computer. At the end we will discuss the next steps for how you could host your new website, making it available for browsing by others via the internet.

Basic Template for HTML

Below is a basic template for an empty HTML Document.

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

    <head>
        ...
    </head>

    <body>
        ...
    </body>

</html>
```

HTML documents start with a `DOCTYPE` declaration that states what version of HTML is being used. This tells the browser how to read the code below it to render the page. If the webpage were written with a different markup language (i.e. XML, XHTML), it would tell you here.

After the `DOCTYPE`, we see the start of the **Root Element**. EVERYTHING—all content—that you want presented on this page and all information about how you want that information to be organized and styled goes in the root element, and it is demarcated by `<html>` and `</html>`.

The root element begins by indicating which language the document is written in; and in this basic template, `en` tells us and the computer that we are writing in English.

Within the root element of the basic template above, you'll notice the two main sections of all HTML documents: a head section (demarcated by `<head>` and `</head>`) and a body section (demarcated by `<body>` and `</body>`).

The **head section** contains basic information about the file such as the title, keywords, authors, a short description, and so on. This is also where you will link to your CSS stylesheet which describes how you want the page styled—colors, fonts, size of text, and positioning of elements on the page.

The **body section** contains the content of the page, including paragraphs, images, links, and more, and indicates how this content is to be structured on the page.

Exercise

Create a folder called `htmlpractice` in your projects folder (`~/Desktop/projects/htmlpractice`). Inside that folder, create a new text file and save it as `index.html`.

Let's use the command line to create the new folder and file:

1. Open your terminal.
2. Navigate to your projects folder using this command:

   ```
   $ cd ~/Desktop/projects
   ```

3. Create a new folder:

   ```
   $ mkdir htmlpractice
   ```

4. Use your Visual Studio Code text editor to create a file called `index.html`: `code index.html`.
5. Paste the template above (starting with `<!DOCTYPE html>`) into the new file.

The `index.html` file is your default homepage for the website we are creating. This is an industry standard, because web browsers tend to recognize the `index.html` page as the opening page to the directory that is your website. See here for more explanation.

Once you've created your new file, open it with a web browser using your graphical user interface:

On macOS, click on the Finder in your dock (the apps at the bottom of the screen) and click on Desktop on the left. From there, navigate to `projects`, then `htmlpractice`. Alternately, you can click the projects folder icon on your Desktop and find it from there. If you're using a Mac and would prefer to use the command line, you can also type `open index.html` from within your `htmlpractice` folder.

On Windows, click the `projects` folder icon on your desktop. Navigate to `projects`, then `htmlpractice`. Double click the `index.html` file. If it does not open in a browser, right click the `index.html` icon and select "Open with..." from the menu. Select Firefox or Google Chrome from the app list that appears.

What Happens?

When you open the empty template, you'll see only a blank web page. Open your secondary menu (right click on Windows, hold `control` and click with macOS) and view the page source.

What Should Happen When I Open Each of my Two New Files?

When you "View Page Source," you should see the code for the basic template.

No content renders on the page, because there is no content in the template at this time.

Evaluation

Which one of these two HTML commands is also known as the "root element"?

- `<!DOCTYPE html>`
- `<html lang="en">`*

Keywords

Do you remember the glossary terms from this section?

- Root Element

Tags and Elements

Tags and elements are the structuring components of html webpages.

Elements identify the different parts of a page, such as paragraphs, headings, titles, body text, images and more. Elements are demarcated by tags which enclose the content of an element (ex. `<head>` and `</head>` are tags that denote the head element of your page).

Tags demarcate elements in one of two ways. As with the paragraph element below, an element can have an opening and a closing tag, with the content in between.

```
<p>This is a paragraph.</p>

<p>
    This is also a paragraph.
</p>
```

Elements which have an opening and closing tag can have other elements inside them. Inside the paragraph element below is a `<strong>` element, which emphasizes the included text by making it bold.

```
<p>
    When I came home from school, I saw he had <strong>stolen</strong> my
chocolate pudding.
</p>
```

Other elements have self-closing tags as with the `<img>` (image) element below. These tags are also called **void tags**.

```
<img src="image.jpeg" />
```

These elements don't require a separate closing tag. Closing tags aren't needed because you wouldn't add content inside these elements. For example, it doesn't make sense to add any additional content inside an image. It is common practice to end void tags like the one above with a `/` to mark the end of it.

Below is HTML that adds alternative text to an image—or text that describes the image. This information added is an attribute—or something that modifies the default functionality of an element.

```
<img alt="This is an image" src="image.jpeg" />
```

Adding alternative text to an image, as was done in this example, is vitally important. That information makes the image more accessible to those viewing your site. For instance, users with poor vision who may not be able to see your image will still understand what it is and why it's there if you provide alternative text describing it.

If you look back at the basic template in your `index.html` file, you'll see that the main sections of your file have opening and closing tags. Each of these main elements will eventually hold many other elements, many of which will be the content of our website.

Evaluation

Which one of the following statements is correct:

- Elements have opening and closing tags.*
- Tags have opening and closing elements.

Keywords

Do you remember the glossary terms from this section?

- Tag
- Elements

Paragraphs and Headings

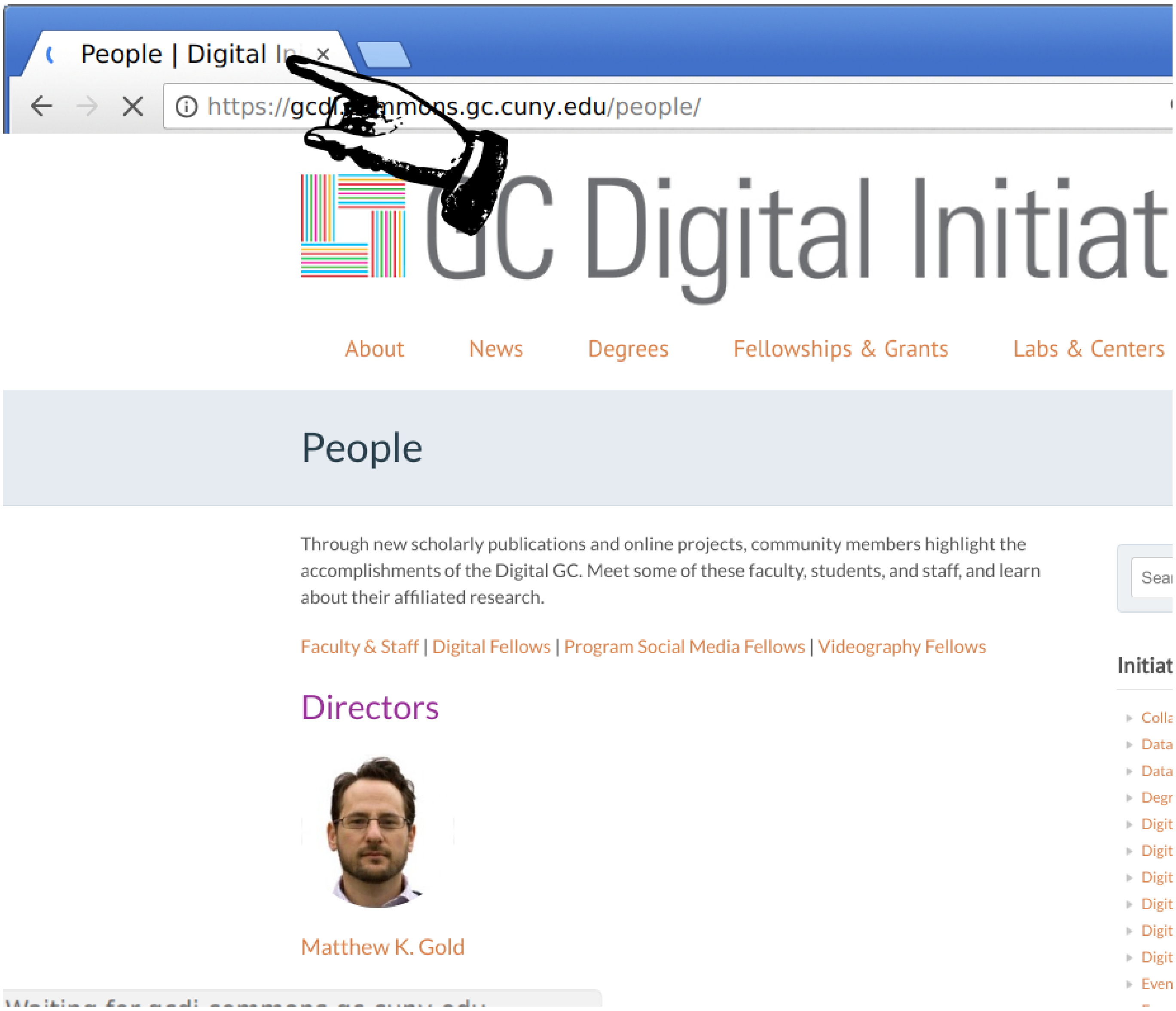
Paragraphs and headings are the main textual elements of the body of your webpages. Because these contain content that you want to organize and display on your webpage, these are entered in the body element.

The `<h1>`, `<h2>`, `<h3>`, etc. tags denote **headings** and **subheadings**, with `<h1>` being the largest and `<h6>` the smallest.

The `<p>` tags denote **paragraphs**, or blocks of text.

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

    <head>
        <title>A boring story</title>
    </head>

    <body>
        <h1>
            Cleaning my boiler
        </h1>
        <p>
            When I got to my basement that day, I knew that I just had to clean my
boiler. It was just too dirty. Honestly, it was getting to be a hazard. So I got
my wire brush and put on my most durable pair of boiler-cleaning overalls. It was
going to be a long day.
        </p>
    </body>

</html>
```

Note that the `<title>` is in the `<head>` element, which is where information about the webpage goes. The title doesn't appear on the page, but instead elsewhere in the browser when the page is displayed. For example, in Chrome, the title appears on the tab above the navbar.



People

Through new scholarly publications and online projects, community members highlight the accomplishments of the Digital GC. Meet some of these faculty, students, and staff, and learn about their affiliated research.

Faculty & Staff | Digital Fellows | Program Social Media Fellows | Videography Fellows

Directors

Matthew K. Gold

Note also that the elements and tags used in HTML have *meaning*. They provide information about the structure of a web page, showing how its parts work together. Those who make use of assistive technologies such as screen readers rely on this semantic information to navigate the page. Thus, it's important to use elements such as headers only when the information being marked calls for it. Making text large and/or bold for visual effect should be done using CSS. The Mozilla Developer Network has some good introductory information on semantic HTML.

Exercise

Using your text editor, add the following to your `index.html`:

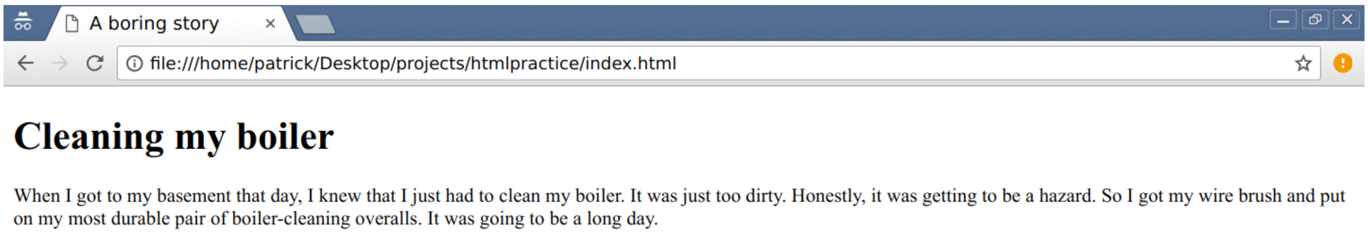
- Title
- Heading
- Paragraph

Then, re-save the file. Open it in your browser again or refresh the page if still opened.

What do you notice about how the information is organized in the webpage? In other words, where are the title, heading, and paragraph text?

What Should You See?

The heading should appear at the top of the page, followed by the paragraph text. The heading text should be larger. The title should appear in the browser window tab.


A boring story
file:///home/patrick/Desktop/projects/htmlpractice/index.html

Cleaning my boiler

When I got to my basement that day, I knew that I just had to clean my boiler. It was just too dirty. Honestly, it was getting to be a hazard. So I got my wire brush and put on my most durable pair of boiler-cleaning overalls. It was going to be a long day.

Evaluation

If I wanted to indicate that "About" is a subheading in my page, which element should I use?

- `<head>`
- `<h2>`*

Links

Links are the foundation of the World Wide Web, and thus are an important component of most websites. Hyperlinking text enables users to move between the different webpages on your site (sometimes in the form of a menu or navigation bar), or connect to other resources or information on other websites.

The `<a>` tag, or **anchor tag**, creates a link to another document. You can use the `<a>` tag to link to other documents or webpages you created for the same site or to documents located elsewhere on the web. You can also use it to link to a particular location on a page—we'll see an example of this in the section on Classes and IDs.

Option One: Relative Links

Relative links take the current page as an origin point and search for other files within the same folder or directory. This method is useful for creating links to pages within your own site.

The following appears as a link to the `about.html` page in the same folder as `index.html`:

```
<a href="about.html">About</a>
```

On your webpage it will appear as:

> About

This link is asking the browser to look for a file titled `about.html` in the same folder. If a file named `about.html` is not in the same folder, clicking the link will result in a `404` ("Page Not Found") error.

Option Two: Absolute Links

An absolute link includes information that allows the browser to find resources on other websites. This information includes the site domain—such as `google.com`—and often the protocol—such as `http` or `https`.

```
<a href="https://www.google.com">Google</a>
```

On your webpage it will appear as:

> Google

This pathway is directing your browser to look online for this text document at the URL address provided.

More on Links

Each example above includes an `href`—a hypertext reference—which is an example of an **attribute**. Attributes offer secondary information about an element.

The `<a>` tag, or anchor tag, creates a link. The text within the `<a>` and `</a>` tags, the anchor text, is what a visitor to the site will see and can click on. The `href=` attribute tells the browser where the user should be directed when they click the link.

There is another technical difference between the two options above.

Relative vs. Absolute Links: When to Use Which One

Use relative links when referring to pages on your own site. The main advantage of using relative links to pages on your site is that your site will not break if it is moved to a different folder or environment.

Exercise

1. Create a new text file called `about.html` in your `htmlpractice` folder. Copy over the HTML from your `index.html` file, but change the text in the `<h1>` element to "About."
2. In your `index.html` file, add a relative link leading to your "About" page.
3. Also add a relative link from your "About" page to your `index.html` page. In this link, call your `index.html` page "Home" (Reminder: `index.html` is the default homepage)
4. Lastly, include an absolute link to a page of your choosing. Remember that an absolute link includes the protocol (for example, `http:`) and also a domain (for example, `cuny.edu`), such as

`http://cuny.edu/about`.
5. Re-save your text files and reopen or refresh them in your browser.

Check If It Worked

When your pages are updated, you should be able to navigate from your "Home" page to your "About" page, and vice versa. You should also be able to navigate to the external web page.

Evaluation

Which one of the following options is a relative link?

- `<a href="https://www.nytimes.com/">The New York Times</a>`
- `<a href="digitalProject.html">Digital Project</a>`*

Keywords

Do you remember the glossary terms from this section?

- Attributes

Images

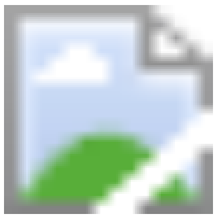
Images are another important component of websites. Sometimes these just help bring your website to life, but other times they can help communicate information to users.

Images are created with the `<img>` tag. Similar to the `<a>` tag, `<img>` requires an attribute, in this case `src`. The `src` attribute stands for "source" and communicates secondary information to your browser that identifies and locates the image. Unlike many other tags, the `<img>` tag does not need to be closed, making it an example of a void tag.

The following element pulls in an image located in the same folder as the `.html` file:

```
<img src="scream.jpeg" />
```

The same rules apply here as with the `href` attribute: if the image is not located in the same folder as the document you are writing in, the browser won't find it. If the browser cannot find an image resource, you will see a broken image icon, such as this one from Chrome:



Note: Some sites use a lot of images. When this is the case, it can be helpful to keep images in a separate folder within your site's structure. To enable the browser to find an image in that case, just add the directory in front of the file name. For example, if you have a folder named images in the same folder as your index.html file, and scream.jpeg is in that folder, you'd change the void tag above to `<img src="images/scream.jpeg" />`.

Making Images Accessible

As briefly noted earlier, alternative text, or alt text, is descriptive "text associated with an image that serves the same purpose and conveys the same essential information as the image" (see Wikipedia Manual of Style/Accessibility/Alternative Text for Images for more), and is important for ensuring content conveyed by images is accessible to all.

To add alternative text to an image, you add an additional attribute, `alt` followed by your descriptive text. For example:

```
<img src="filename.png" alt="Text in these quotes describes the image" />
```

For more information about using alt text, see what the Social Security Administration has to say.

What Images May I Use On My Site?

If you're planning to use images that you did not take or make yourself, you'll need to use "public domain" or "open license" images.

This guide by the OpenLab at City Tech includes more information on licensure and a list of places where you can find reuseable images.

Exercise

Download and save an image from the web, or move an image from your computer into the same folder as your `index.html` file.

Tip: Give the file a simple name. Also, the name **cannot** have spaces. A good practice is to use either dashes or underscores where there would otherwise be spaces. For example: `this-is-an-image.jpg` or `this_is_an_image.jpg`.

Using the code above as a reference, add that image into your `index.html` file, re-save the file, and re-open or refresh the page in your browser. Your image should now appear on the page.

Evaluation

True or False: Does including "alt text" in websites improve their accessibility?

- True*
- False

Conventions

As we’ve gone through the different components of creating a webpage, you likely have noticed some common conventions or industry standards for creating a webpage using HTML. Can you guess any of these?

Here are a few examples:

- Some tags are self-closing, while others require a closing tag. Self-closing tags are called void tags, and are generally self-closing because you wouldn't need or want to add another element within a tag. They also generally end with a forward slash (`/`) to mark the end of the tag.
- Use lower case. While HTML is not case sensitive, it makes scanning the code easier, and makes it look more consistent.
- Your code should be nested. This is not a technical necessity either—blank space has no meaning in html. However, this makes it easier to scan the code quickly, which is particularly helpful when you run into errors!

Exercise: Create a website

In this exercise, we will begin creating a website for your personal portfolio or course. Using the tags we've just reviewed, and two additional ones (see below), we will make a barebones website that provides information about your academic profile and/or courses.

The first step is to create a new folder called `website` in your `projects` folder on your desktop. Create an `index.html` as well as an `about.html` file inside that folder. These will be the landing page of your site, and a supplemental page that provides information about your Digital Humanities Institute's organizers.

Add HTML to your `index.html` file. This page should include the following:

- Doctype
- Root element
- Head and a body
- Title for the page
- One heading
- One paragraph
- One image with alt text
- A menu or navigation bar that links to your Home and About pages

Think about the order of your content as you assemble the body of your page. Don't worry about getting the content just right. The important aspect of this exercise is to review the structure of a webpage, and practice creating a webpage.

Additional Tags

Here are two additional tags that might come in handy in assembling your page:

To **make a list**, you open and close it with the `<ul>` tags, and each item is an enclosed `<li>` tag:

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

The HTML above will produce an unordered (bulleted) list. To create an ordered (numbered) list instead, just substitute `<ol>` and `</ol>` for `<ul>` and `</ul>`.

(This may come in handy when making your menu or navigation bar.)

To **make a line break** or give space between different elements:

```
<br />
```

CSS Basics

CSS stands for Cascading Style Sheets. This language works in coordination with HTML, but is its own language with its own rules and terminology. In contrast to HTML, which is responsible for the content of the page, CSS is responsible for the presentation of the page.

Examples of what CSS can help you determine include:

- What background color you want to use for the page or a paragraph.
- What font or font size you want for your headings or your normal text.
- How large you want the images, and whether you want them aligned center, left, or right.
- Where elements appear on the page.
- Whether elements are visible to a user or not.

Evaluation

Is CSS a markup language or a programming language?

- Markup Language*
- Programming Language

Integrating CSS and HTML

In order for CSS to inform the style of the content on the page, it must be integrated with your HTML. CSS can be integrated into your HTML in three ways:

1. inline
2. internal
3. external (*recommended*)

Option 1: Inline

Inline styling adds CSS directly into the HTML of a page to adjust the style of particular parts of a page.

For example, if you want the text of your first paragraph to be red, but the text of your second paragraph to be blue:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>About</title>
    </head>

    <body>
        <p style="color: red">
            Content of paragraph
        </p>
        <p style="color: blue">
            Content of paragraph
        </p>
    </body>
</html>
```

Option 2: Internal

Internal styling also adds CSS directly into the HTML, but keeps it separate from the content code of the page by adding it into the head using the `<style>` tag. When using internal styling you are providing styling rules for the entire page. For example, if you want all headings to be blue:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>About</title>
        <style>
            h1 {
              color: blue;
            }
        </style>
    </head>

    <body>
        <h1>
            Heading One
        </h1>
        <p>
            Content of paragraph
        </p>
        <h1>
            Heading Two
        </h1>
        <p>
            Content of paragraph
        </p>
    </body>
</html>
```

Option 3: External (Recommended)

External styling creates a completely separate document for your CSS that will be linked to your HTML in the head section of your HTML document using the code below. This separate document is called a *stylesheet* and is often named `style.css`. The document is linked through a void `<link>` tag that lives inside the parent `<head>` tag. Its `href` attribute is a relative link to the document somewhere in relation to the document that references it.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>CSS Example</title>
        <link rel="stylesheet" href="style.css" />
    </head>

    <body>
        ...
    </body>
</html>
```

Best Practices

It's best practice to use Option 3, external styling, for a number of reasons:

1. It helps us remember what each language focuses on: HTML is for *content*, CSS is for *styling*. (This is sometimes referred to as the principle of "separation of concerns")
2. It helps us maintain consistency across the various pages of our site as *multiple HTML files can link to the same stylesheet*.
3. Because multiple HTML files can link to the same CSS file, it's not necessary to write the same CSS code multiple times. Once suffices. (This is sometimes referred to as the "Don't Repeat Yourself" principle, or simply *DRY*.)

Option 3, external styling, is preferred by most web developers because it's more manageable and because it lends itself to greater consistency across the entire site.

Exercise

Create a blank stylesheet using the command line (following option 3, external styling, described above). In your `index.html` document, link to your style sheet and re-save the file.

To link your stylesheet with your `index.html` file, insert the following code into the head element of that `index.html` file:

```
<link rel="stylesheet" href="style.css" />
```

Evaluation

Is the following code-snippet an example of inline styling or internal styling?

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Homepage</title>
        <style>
            h1 {
              font-family: monospace;
            }
            p {
             font-family: fantasy;
            }
        </style>
    </head>

    <body>
        <h1>
            Online Library for All!
        </h1>
        <p>
            Free books here!
        </p>
    </body>
</html>
```

- Inline Styling
- Internal Styling*

Rule Sets

CSS is based on selectors and declarations, which together form rule sets (or just "rules"). Rule sets comprise an external styling file with a `.css` extension. Here is the contents of a sample `.css` file:

```
h1 {
    color: orange;
    font-style: italic;
}

p {
    font-family: sans-serif;
    font-style: normal;
}

#navbar {
    background-color: yellow;
    padding: 80px;
}

.intro {
    font-family: arial;
    background-color: grey;
    color: dark-grey;
}
```

The first rule (which starts with the `h1` selector) applies to all `<h1>` tags on each page where your HTML document links to your stylesheet, and changes the font style and display of those headings.

The lines within the curly braces (i.e. `{ ... }`) are called **declarations**, and they change the formatting of the elements in the HTML document. Each line in the declaration sets the value for a **property** and ends with a semicolon (`;`).

Note the different syntax being used to select items for for styling with rule sets. The bottom two selectors are used to apply rule sets to **IDs** and **classes**. In general, adding classes and IDs to HTML elements allows for more specific styling—more on these soon!

Exercise

Copy and paste the CSS rules above into your `style.css` file and re-save the file. Then open or refresh your `index.html` file in your browser and see what happens.

What should happen?

The formatting of the text on your page should change accordingly. Your `<h1>` should be orange and italic, for example.

What are some other rules you might set for different HTML elements? Do a quick Google search for a CSS rule that changes the appearance of your page, such as putting a border around an element.

Evaluation

How do we associate a CSS file with an HTML page?

- By including a link to the CSS file in the HTML page's `<head>` element.*
- By putting the CSS file in the same folder as the HTML page.

Keywords

Do you remember the glossary terms from this section?

- CSS Selectors
- Class
- ID