Scientific Computing

Mon, Feb 2

Announcements

* Homework 2 due Fri, Feb 13, 11:59pm
  pdf + zip file on D2L
  start early!!
  covers Greedy Algorithms

Don't forget to keep track of and cite any external resources you use - friends, websites, AI, etc.

Office Hours:

Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

Topic 4 - The Unix Command Line

Unix was an OS framework developed in the 70s that is a precursor to the OSes of today (everything except Windows).

Mac and Linux have terminals where you can still issue Unix commands, and "Git for Windows," which we installed is a Unix terminal emulator.

The VS Code terminal is also a Unix emulator.

Today we'll cover just the very basic commands to navigate and manipulate the file/folder system.

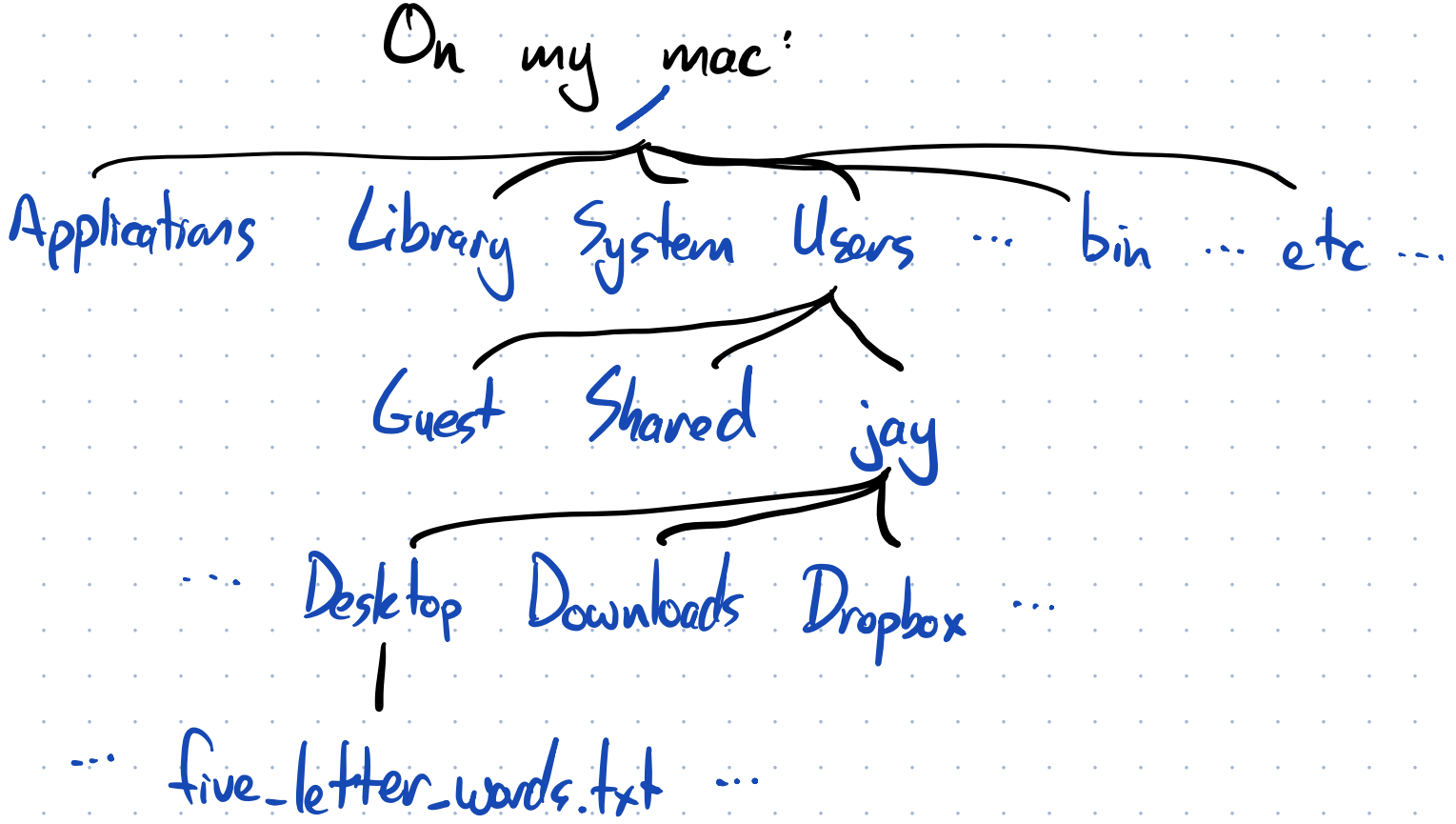
Why is this useful:

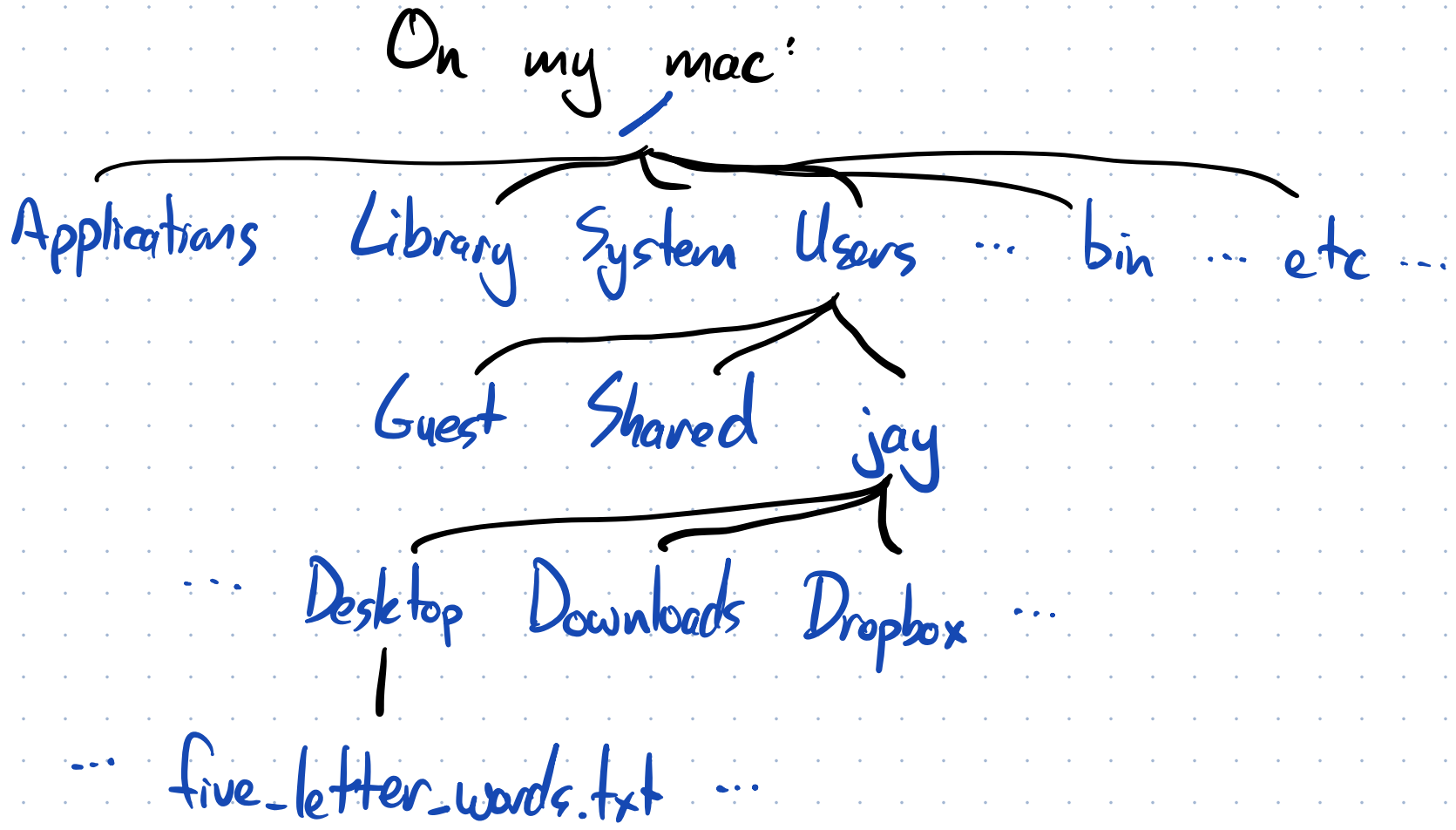
- On your own machine, a lot of it can be done with the GUI, but sometimes this is more efficient (e.g., view the first 10 lines of a 250mb text file).
- Especially though: any work you do on a server you connect to via SSH (e.g., my research server Ada).

(Software Carpentry website has great resources.)

File system:

Files in a computer are stored in a hierarchy. The very top is called "/" in unix-like systems, and usually "C:\" in Windows.





So, every file has a full address:

/Users/jay/Desktop/five_letter_words.txt

Demo: Open your terminal or Git for Windows or VS Code

(1) pwd - "present working directory", or where in the file system you are.

(2) ls - "list", display the files in the current folder

Most commands have extra arguments you can pass to change the behavior (sometimes tons of them). "flags"

Ex: `ls -l` - list the files in the current folder with extra information

To see the full "manual page" for a command, do `man command`, e.g., `man ls`. Press "q" to exit.

You can also tell ls, and many other commands, to only act on some files, using "*" as a symbol that means "anything":

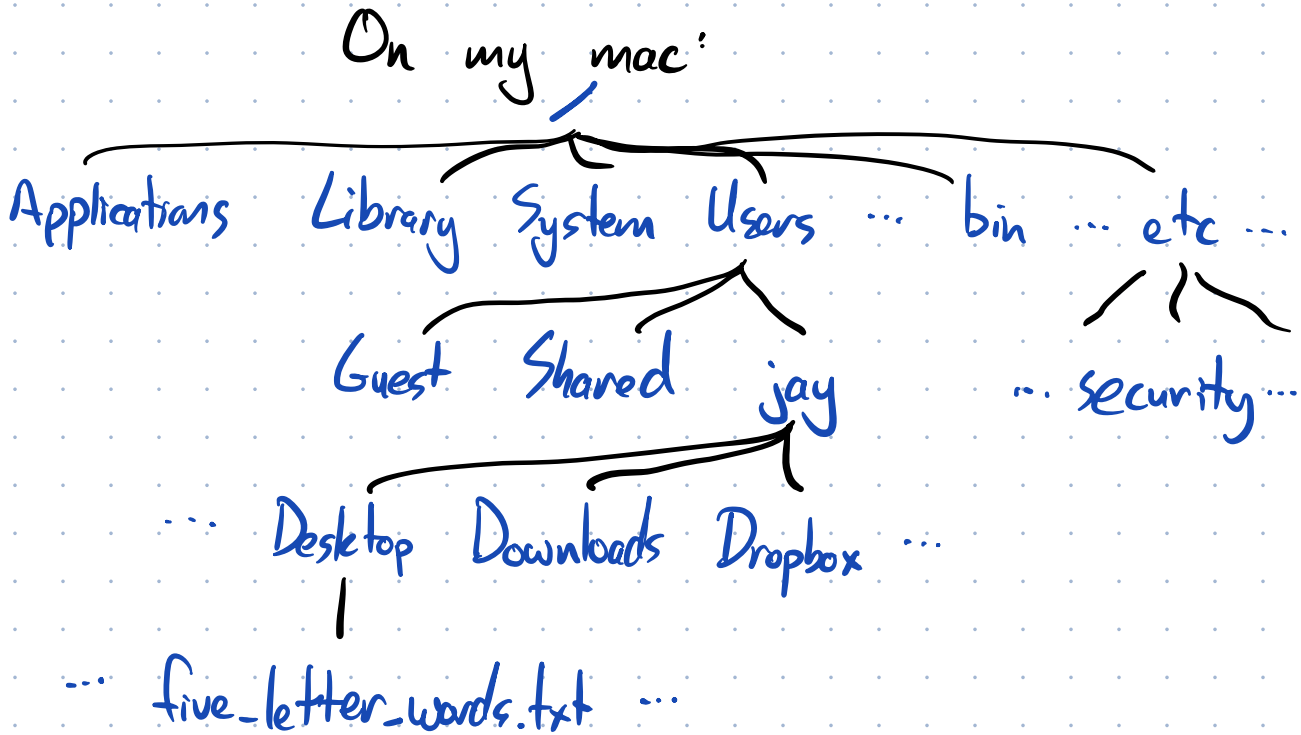
```
ls -l *.txt
ls -l p*
```

(3) cd - "change directory", move to a different place in the file structure.

cd [directory]

If you directory starts with "/", you are specifying an absolute path, exactly where a folder is.

If it doesn't start with "/", you are specifying where it is relative to your current location.




```
> cd /etc/security
> pwd
    /etc/security
> cd /Users/jay
> pwd
    /Users/jay
> cd Dropbox
> pwd
    /Users/jay/Dropbox
```

cd /Users/jay/Dropbox

To go from /Users/jay to /etc/security: cd ../../etc/security

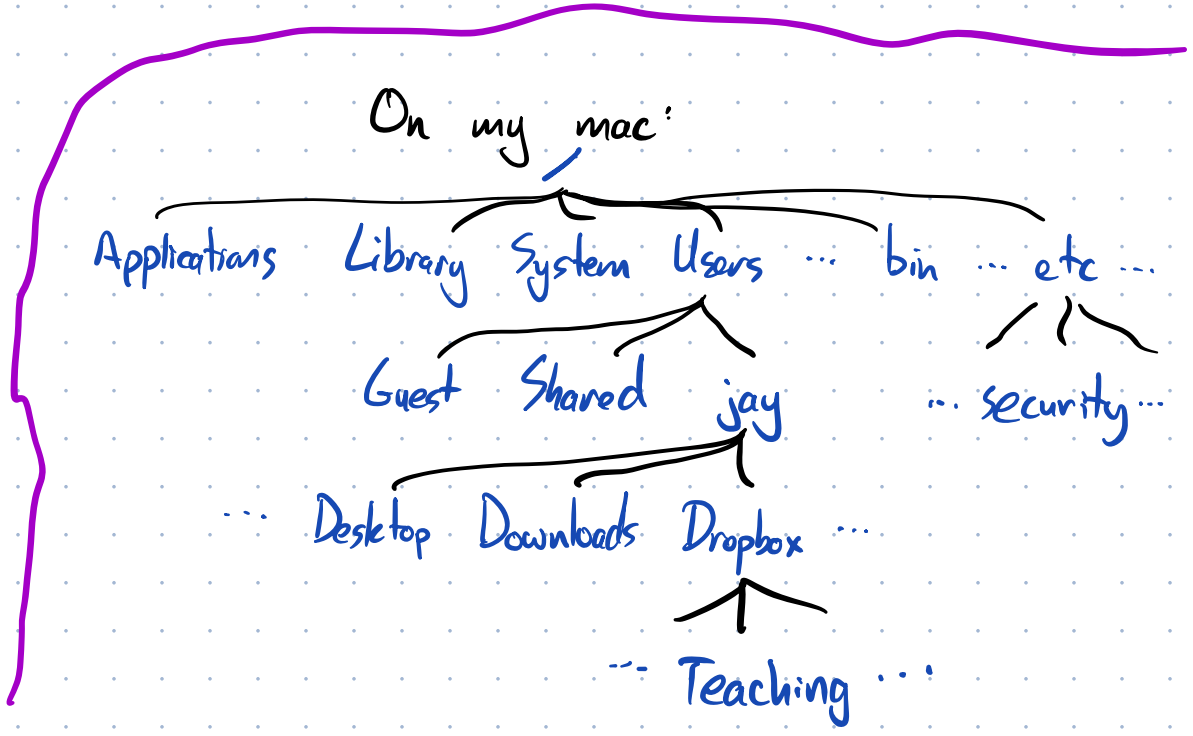
".." = up

Shortcuts: "." - current folder (not helpful here)

".." - up one level

"~" - user's home folder

```
> cd ~/Dropbox/Teaching
> pwd
    /Users/jay/Dropbox/Teaching
> cd ../../Desktop
> pwd
  /Users/jay/Desktop
```



You can use the "tab" key to complete a command or filename if it's unique, or press it twice to list possibilities.

(4) mkdir [name] - "make directory" (folder)

(5) mv [current location] [source location]
- move/rename file or folder

(6) cp [source file] [destination file]
- copy a file or folder
↳ requires flags! "-R"

(7) rm [file or folder]

- "remove"/delete a file or folder
  - requires "-r" flag

⚠ Warning! This is dangerous!
"rm -rf /" will just delete all your system files. They don't go in a trash/recycle bin and can't be recovered. ⚠

(8) cat [file name] - print a whole file to the screen

(9) head [file name] - print first 10 lines of a file

(10) tail [file name] - print last 10 lines

"-n" to change the number for head and tail, like

head -n 20 [file]

(11) less [file name] - open a file in a way you can scroll around but not edit

"q" to quit

Note: you can use the up and down arrows to scroll through your history of commands.

You can do anything on a terminal.
A few non-beginner things.

(12) nano [filename] opens a terminal-based file editor. Keyboard shortcuts for everything, [ctrl] + [key] on Mac. You may need to install nano on some platforms.

(13) touch [filename] just puts a blank file in the current folder

You can write whole programs ("bash scripts") to do anything.

When you run Python, its "active directory" is whatever your p.w.d. is in the terminal window where you're running it.

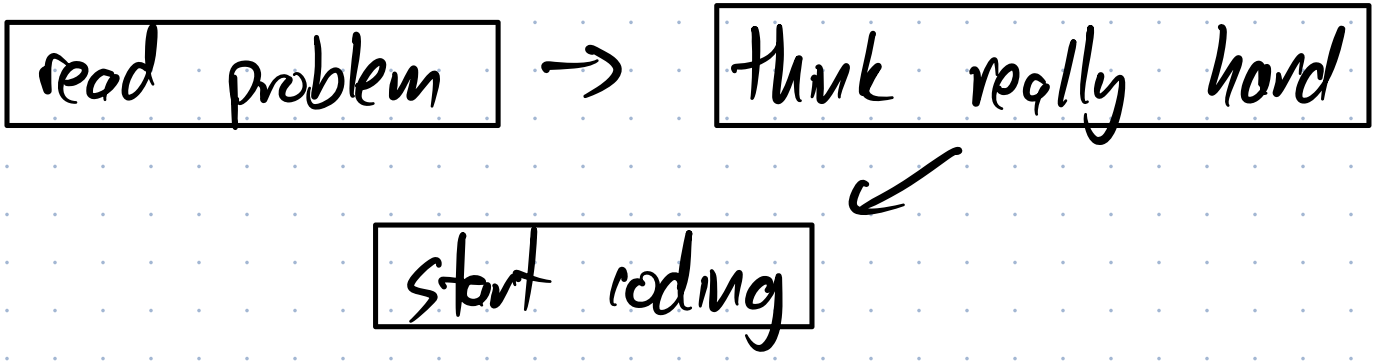
Example:

```
with open("five_letter_words.txt", "r") as f:
    words = f.readlines()

#do any words have no vowels?
vowels = ["a", "e", "i", "o", "u"]
print([w for w in words if not any(v in w for
                                   v in vowels)])
```

Topic 4 - The Coding Process

Hardest Approach:



Too many steps at once, all in your head

Better Process:

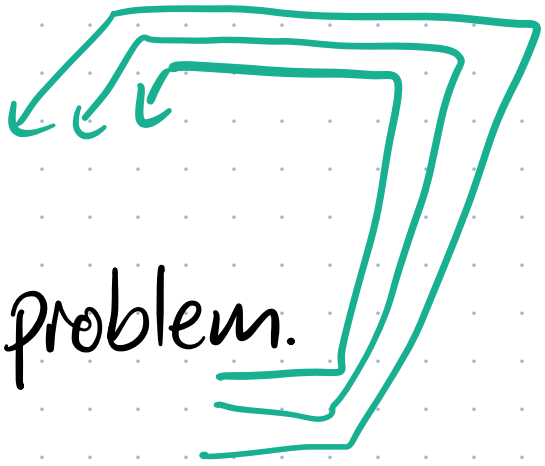
1) Read the problem.

Better Process:

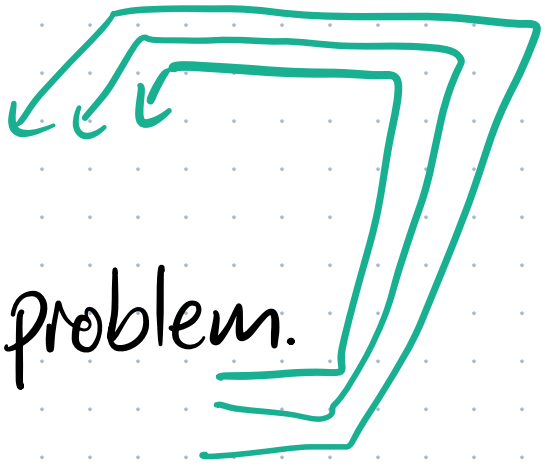
1) Read the problem.
2) Think about the problem.

Better Process:

1) Read the problem.
2) Think about the problem.



Better Process:



1) Read the problem.
2) Think about the problem.

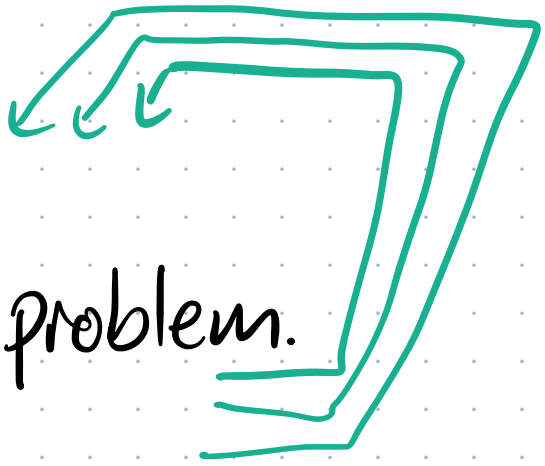
3) Do some examples by hand to see if you understand the problem.

   Leg. Longest Collatz sequence:

   $20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

   length 8)

Better Process:



1) Read the problem.
2) Think about the problem.

3) Do some examples by hand to see if you understand the problem.

   Leg. Longest Collatz sequence:

   $20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

   length 8)

4) Think about how you could solve it.

   What steps did you do by hand?

Better Process:

1) Read the problem.
2) Think about the problem.
3) Do some examples by hand to see if you understand the problem.
   e.g. Longest Collatz sequence:
   $20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$
   length 8)
4) Think about how you could solve it.
   What steps did you do by hand?
5) Write out, by hand, in words, the steps of your algorithm (pseudocode)

Ex: (Collatz)

```
set longest_chain = 0
set longest_num = 0
loop over "num" from 1 to 1 million:
    [compute length of chain for num
      if length > longest_chain:
          longest_chain = length
          longest_num = num
answer is longest_num
```

This part is many steps! You can think about it separately, or even as a function.

6) Now start coding!

As you code:

7) "Rubber Ducking"

Talk to a rubber duck, out loud, explaining what you're doing in each line, and <u>why</u>

8) Pause often to test a few lines of code at a time before writing more.

→ Do they do what you thought?

→ Is your loop looping over the right thing? (print something to check!)

→ Does the list you built contain the things you thought?

If it's not working:

9) Debug it! Think of small test cases.
(1 to 10 instead of 1 to 1 million)

Add lots of print statements to see what values the variables hold in each step and see if they are what you expect.

When it's working:

10) Test it! Test the small examples from step 3.
Do you get the expected answer?