New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Getting Started with Python in a Terminal Window (Mac and Linux/Unix)

We have two choices when ***running*** Python programs:
1) Python 2 (installed by default)
2) Python 3 (must be installed)

We have two choices when ***writing*** Python programs:
1) the Atom text editor; or
2) another text editor or IDE like PyCharm

This step-by-step guide shows you how to install Python (if you will be using Python 3, the official class version), how to get around the Mac/Unix file system, how to write and save a Python script using a text editor, and how to execute the script within a Unix terminal window. If you know how to do these things and you're comfortable with your editor or IDE, you don't need these instructions.

If you *don't* know how to do these things, ***please follow all steps below before our first class session***.

Python on the Macintosh requires that we access the Mac's underlying Unix system, so if you are running Linux or Unix, you can follow these same general steps -- you'll only need to find a Terminal window that corresponds to the one described here.

Note: we will be using Python 3.x (latest version, 3.6) - examples used in class will not run in Python 2.x. However, the differences at an introductory level are minimal: if you prefer to learn Python 2.x because your company prefers it, it is a simple matter to convert Python 3.x examples so they can run on Python 2. In addition, all of our course materials will be presented in both versions so you can compare and/or switch at will.

1. Install Python (Python 3 users only). Python 2 comes installed on the Mac by default. If you wish to learn Python 3 in this class (the official class version), it must be installed. (For Linux / Unix, the available versions may depend on the specific OS you are running. At first, assume that Python is installed; if you run into problems (such as the error message **python3 is not a recognized program** or similar), then you will need to install Python.)

   Instructions for the Mac:
   a. Visit **python.org**
   b. Click Downloads > Mac OSX > Latest Python 3 Release
   c. Scroll down to **Mac OS X 64-bit/32-bit installer**
   d. A file should download; double-click the file and follow the instructions

e. Once installed, open a *new* **Terminal** window (see steps 2.a.i-iv below) and type the following text at the **$** prompt:

```
python3 -V
```

You should see **python 3.6.0** or similar number displayed. If you don't, make sure you've opened up a *new* Terminal window, and that you're typing the above correctly. If you still don't see **python 3.6.0** or similar version, **please let me know**.

2. Open a Terminal window. The Terminal program is the way we talk to Mac or Unix. We'll use Terminal to execute our Python scripts, and locate, name and move files.

   a. If on the Mac:
      i. Search for **Terminal** by clicking on the magnifying glass in the upper right corner of the screen and typing **Terminal** into the text window that appears.
      ii. In the list that drops down, click on the Terminal icon (usually, "Top Hit"). A small white window with text appears.
      iii. (You can make Terminal accessible from the "Dock" (the icons at the bottom of the screen) by searching with the magnifying glass as above, then selecting **Show all in Finder** (a window of listings appears), and draggging the black **Terminal** icon down from the Finder window into the dock. The other icons will make room for it. (This is not moving **Terminal**, just creating a shortcut.))
      iv. Inside the Terminal window, you'll see some text followed by a **$** and a small rectangular cursor. This is the *Unix prompt.* It is Unix telling you that it's ready for your commands:

```
Last login:  Fri Jun 12 14:39:46 on ttys001
myuser:~ myuser$
```

(In these examples, ***myuser*** will correspond to your own username on your system.)

   b. If you're running Unix or Linux directly (i.e., not the Mac), you must find your Terminal window yourself -- every system will be a little different and might even name the Terminal program differently. Email me if you're having difficulty.

   The Terminal window lets us move around in the file system and execute Python code.

3. Find your way around the file system. Unix provides a text-based command system. In a Graphical User Interface environment like Mac or Windows, we click on things and drag them around. In the Unix command-line environment, we use text commands to determine your current directory, what is in a given directory, and to get to where you want to be. *Spend a few minutes each day working with these commands and you'll be familiar with the environment after just a few days.*

Your purpose in this section is to be able to move around the folders/directories on your system, see what is in a directory, and create a directory. In other words, what you would do normally by using the Finder folder windows.

a. Present Working Directory (**pwd**). At any given moment, your Terminal session is "sitting" in a particular folder. To see what folder you are in, execute this command: **pwd**. In these examples, the commands to type are in bold:

```
myuser:~ myuser$ pwd
/Users/myuser
myuser:~ myuser$                # the prompt reappears for the next command
```

(Keep in mind that ***myuser*** refers to whatever your home directory name is.)

The Unix/Mac filesystem, like that of Windows, is organized into folders within folders (i.e., directories within directories). In other words, directory A may contain directories B and C, and C may contain directory D, etc. -- we sometimes call it a filesystem "tree".

In Unix, the folders are notated with a forward slash. Therefore the above **pwd** output, **/Users/*myuser***, shows that we are in the ***myuser*** directory (remember, this will have your name or some other name, not ***myuser***), which is inside the **Users** directory, and the **Users** directory is at the root (the root can always be identified by a path that starts with the forward slash). **pwd** means the *present working directory*, and this value will change when we move to a different directory. (The tilde ~ is another way of noting the user's home directory - not important to understand right now.)

b. List directory (**ls**). The **ls** command shows the contents of the present working directory. You'll see folders and files there - these are folders and files that are inside the present working directory (i.e., inside **/Users/*myuser***). The initial directory/folder that you start in when you start Terminal is known as the *home directory*. Every user has a home directory, and this does not change.

```
myuser:~ myuser$ ls
a_file.txt      Movies
Desktop         Music
Documents       mydir
Downloads       Pictures
Library         Public
mixfile.txt
myuser:~ myuser$
```

The files and folders are mixed together, and in alphabetical order (your list of files will of course be different.) . They are also often shown in multi-column format. How can you tell which are folders and which files? You can use **ls -l** to see a longer listing, but we're going to move around a little first.

c. Change directory (**cd**). We change our present working directory with this command. Simply specify the directory you'd like to move in *relative to where you are.* So if we're in

/Users/*myuser*, ane we've seen that the **Desktop** folder is inside ***myuser*** (as we did in the listing above), we can get there with this command:

```
myuser:~ myuser$ cd Desktop
myuser:Desktop myuser$
```

Note that after we enter a new directory, the prompt changes - it's there to let us know the name of the directory we're in. But **Desktop** alone doesn't tell us the whole story about our location -- it's just the name of the directory. We can again check to see exactly where we are in the filesystem with **pwd**:

```
myuser:Desktop myuser$ pwd
/Users/myuser/Desktop
myuser:Desktop myuser$
```

Note that we have moved into the **Desktop** folder, so it is included in the **pwd** path. Now an **ls** will show the folders and files found in **Desktop**. Try it!

To change into the parent directory: simply use the shortcut for "parent", a double dot (..):

```
myuser:Desktop myuser$ cd ..
myuser:~ myuser$ pwd
/Users/myuser
```

To go back to the home directory: **cd** by itself will take us there. Let's move into the Desktop directory with **cd Desktop** and then back into the home directory with **cd**:

```
myuser:~ myuser$ cd Desktop
myuser:Desktop myuser$ pwd
/Users/myuser/Desktop
myuser:Desktop myuser$ cd
myuser:~ myuser$ pwd
/Users/myuser
```

d. Create a new directory to hold python scripts. **cd** into **Desktop**, then use the **mkdir** command to create a directory *inside the present working directory*.

```
myuser:~ myuser$ cd                          # making sure we're at home
myuser:~ myuser$ cd Desktop                  # move to Desktop
myuser:Desktop myuser$ mkdir python_scripts  # create a dir on Desktop
myuser:~ myuser$ cd python_scripts           # move into the new dir
myuser:python_scripts myuser$ pwd            # confirm we're there
/Users/myuser/Desktop/python_scripts
myuser:python_scripts myuser$ cd ..    # move back into the parent dir
myuser:Desktop myuser$ ls                    # confirm it's there
a_nice_file.txt
Dropbox
python_scripts                # (not bolded on your screen)
z_nice_file.txt
myuser:~ myuser$
```

(Note that when any command is successful (and no output is expected), the command prompt simply reappears -- there is no confirmation that a command succeeded. This is standard behavior for Unix. Instead, if there is a problem with your command, you'll see an error message.)

In your listing of **Desktop** you may see a lot of files (depending on how tidy or untidy your Desktop may be!). But you certainly should see **python_scripts** directory listed this is the directory you created inside the **Desktop** directory.

Because **Desktop** represents your actual Desktop, you should also see this directory represented as a graphical folder on your Desktop.

4. Write a Python program using a text editor.

   a. Edit files using **Atom**. Atom is a versatile, simple and free text editor designed for software developers, developed by the awesome folks at Github. You are free to choose a different editor or IDE, but we will be using Atom in class.

      i. Install **Atom** by downloading it from the above link and following the install instructions.
      ii. Open **Atom**. A new document is opened.
      iii. Edit the file so it looks like this:

```
#!/usr/bin/env python3    # make sure this is the very first line, flush left!

print('hello, world!')
```

The first line is called the **shebang** line, which is used by the Unix operating system (i.e., not Windows) to identify your program as a Python program. When running on Unix / Mac this line *must be the first line of the file.* This line instructs the Unix shell to run your script as a python script.

      iv. Save the file in your **python_scripts** directory.

Remember, in the previous section you created this directory inside your home directory, at **/Users/*myuser*/Desktop/python_scripts** (where ***myuser*** is your home directory name).

Hit **command-S** or click **File > Save** from the menu at the top. If you don't see **python_scripts** in the file save window, click the little down-arrow to the right of the filename blank to navigate to the proper folder.

Save the file with the name **hello.py** in the **python_scripts** directory. Notice that when you save the file with the **.py** extension, the text of the file changes color. This is called *syntax highlighting,* and it shows that **Atom** is aware that this is a python program, identifying the various pieces of syntax and highlighting them to aid in

reading. You can use this highlighting to help identify the Python "parts of speech" as well as to identify syntax mistakes. (Overall color scheme and highlighting can be changed in **Atom's** preferences.)

It's crucial during this step that you're sure of the folder location where this file is being saved. Double check to be sure! You could easily save it in a different place and then lose track of its location. We decided to save it in the **python_scripts** directory, and that directory is in your **Desktop** folder. On a mac you can confirm a file's location by Ctrl-clicking on the filename and icon at the top of the open **Atom** window -- this will temporarily display the file's file path using icons and names. You should see your filename on top of (inside) the **python_scripts** folder, on top of (inside) the **Desktop** folder, on top of (inside) your home directory (which is marked by a little house icon), on top of (inside) the **Users** directory, usually inside the **Macintosh HD** system icon.

If you're not seeing the file in this location, you can choose **File > Save As…** to save the file in the proper location, or probably better, switch to the Mac's Finder, locate the file and drag it into the **python_scripts** folder in your **Desktop** directory.

If you can't find the **python_scripts** folder in your **Desktop** directory, you'll need to go back to your first Terminal window, issue the **cd** command, and repeat the steps starting with "Find your way around the file system".

It's also important that you not create a **Desktop** or **python_scripts** folder in a different location just so you can work with them. **Desktop** is a specific folder and you must not create another one or the locationing will become very confusing. The proper location should be: **/Users/*myuser*/Desktop/python_scripts** (where **myuser** is the name of your home directory).

v. Once you have verified that the file is saved in the proper location, leave **Atom** open. We will return to it shortly. Jump from here to the next numbered step (i.e., skip the **nano** editor instructions unless you intend to use **nano** instead of **Atom**).

b. Unix/Linux: edit files using **nano**, or your favorite text editor. **nano** is a command-line text editor that is usually available on Linux. It allows us to edit files directly and saves us having to use a separate text editor. The interface is simple and great for beginners. But, feel free to use your favorite text editor if you have one.

i. To start, *open a new Terminal window* (from Terminal, use **Cmd-N** to launch a new Terminal window). We'll continue to work with two windows simultaneously - one to edit the file (and keep it open for further edits) and another to execute the file (i.e., run the Python program).

ii. The new Terminal window will probably start in the same directory as the previous one, but just in case **cd** home, then, **cd** into the same directory as the other window, so their **pwd**s match:

```
myuser:python myuser$ cd
myuser:~ myser$ cd python
myuser:python myuser$ pwd
/Users/myuser/python                    # confirmed - same location
```

iii. Still in the new Terminal window, execute **nano.** Type **nano hello.py**, where **hello.py** is the name of a new or existing file you'd like to create or edit. Nano will open in the full terminal window, showing an empty document (or the document's text if the file already exists). Now, type in your first Python program (shown in bold below **GNU nano 2.0.06**):

```
myuser:python myuser$ nano hello.py


================================================================================
GNU nano 2.0.6              File: hello.py                          Modified

#!/usr/bin/env python3    # make sure this is the very first line, flush left!

print('hello, world!')




^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

The first line is called the **shebang** line, which refers to the Unix shell and the *bang* (exclamation point) following the hash mark. When running on Unix this line *must be the first line of the file.* This line instructs the Unix shell to run your script as a python script.

Edit as you would a normal editor, with arrow keys and backspace. When you're done editing, save the file with **Ctrl-O** (i.e., hold down the Ctrl key and then type O) and accept the file name (you could also specify a different one) by hitting Return. **nano** will save (*Write Out*) the file under that name *in the present working directory*.

You can cut and paste from other applications into **nano** in the usual way, and you can cut and paste lines of text in **nano** itself using Ctrl-K and Ctrl-U. There is no built-in undo functionality that I know of.

For now, leave this **nano** window open - we'll come back to it soon.

5. Find the script in your original Terminal window. We need to again be sure that our saved script, and the **pwd** of our original Terminal window, are in the same place. We intended to save our script in the **/Users/*myuser*/Desktop/python_scripts** folder, and an easy way to confirm this is to just list the contents of the directory where we believe the script has been saved.

   a. Go back to your original Terminal window, i.e. click on that window.
   b. Do a **pwd** to verify that you're in the right directory:

```
myuser:python_scripts myuser$ pwd
/Users/myuser/Desktop/python_scripts
myuser:python_scripts myuser$
```

   c. If you're not in the right directory, go back to 'home' and **cd** back into it:

```
myuser:python_scripts myuser$ cd            # back to 'home'
myuser:~ myuser$ cd Desktop/python_scripts  # if this fails, start over
myuser:python_scripts myuser$
```

   d. Now do an **ls** and see that the script exists where you wrote it:

```
myuser:python_scripts myuser$ ls
hello.py                        # there it is!
myuser:python_scripts myuser$
```

   If you don't see the file you created, you can retrace your steps, or start over and follow the same path to the directory you wanted:

```
myuser:~ myuser$ cd                 # just making sure we're home
myuser:~ myuser$ cd Desktop
myuser:~ myuser$ cd python_scripts  # the dir we wrote the file to
myuser:python_scripts myuser$ pwd
/Users/myuser/Desktop/python_scripts
myuser:python_scripts myuser$ ls
hello.py                        # there it is
myuser:python_scripts myuser$
```

   If you again don't see the file you created, either go back to **Atom** and re-save the file so that you can see where it is saved, (or if on Unix/Linux, go back to the **nano** window, exit **nano** and do a **pwd** in that window to see where you wrote the file. Then either re-write the file in the correct place (i.e., **cd** into the right directory and re-launch **nano**), or use **cd** *in the first (non-nano) window* to to get to the directory where the file was saved.) If you can't find the file and get confused, you may have to start over - and follow these instructions as closely as possible.

Remember, in this new environment you have four commands to see where you are and what you have, and to go where you want to go:

```
ls              # list the current directory
pwd             # see what directory you're in
cd dirname      # go into a directory within this directory
cd ..           # go to the parent directory
cd              # go to the home directory
```

6. Change the script's permissions (*only needed once per script*). In order to execute the script directly, we need to give the file permission to be executed by anyone. In the first window:

```
myuser:python_scripts myuser$ ls
hello.py                          # confirmed:  it's here
myuser:python_scripts myuser$ chmod 755 hello.py
myuser:python_scripts myuser$       # no response means success
```

As noted, this need only be done once per script, the first time you create it. It doesn't have to be done again for a given script.

7. Execute the script. We can execute (run) the Python script by typing **python** and then the name of the script at the command line.

If we have a proper shebang line and have **chmod**ed the file, we can also execute the script directly. Strangely, though, we have to specify the directory first, even if we're in the same directory as the script. So, script execution in the same directory starts with **./**, because a single dot (**.**) means in the current directory:

```
myuser:python_scripts myuser$ ./hello.py
hello, world!
myuser:python_scripts myuser$
```

8. Lather, rinse, repeat. Now our continuing process of editing and running our programs will repeat some of the above steps. Here they are summarized, along with keyboard shortcuts to make the cycle go even faster -- *it is highly recommended that you develop a keyboard workflow.*

a. Return to your text editor and make changes to your file:
   i. For Mac, return to **Atom** and edit the file. Save the file (by default, in the same place it was created): ***Cmd-S***.
   ii. For Unix, return to **nano** and edit the file. Save the file with Ctrl-O. Don't exit **nano**.
b. Switch to the the first Terminal window (***Cmd-Tab***) and run the script (***up-arrow to retrieve the last command, then Return***).
c. Switch back to the **Atom** window (***Cmd-Tab***), re-edit and re-save the script (***Cmd-S***).
d. Switch back to the Terminal window (***Cmd-Tab***) and run the script (***up-arrow, Return***).
e. Now, repeat steps **c** and **d** until the script is debugged and complete.

9. Download and unzip the sample data into a folder called **python_data**. Starting in Session 2 we will begin working with sample data, which is located at the **source data** link on the website. All of this data is zipped into a file called **ALL_DATA.zip** at that link.

   a. Visit the class website at http://davidbpython.com/introductory_python and click on **source data**.
   b. Click on the **ALL_DATA.zip** to download. Your browser may ask if you want to open the file; you can if you wish, or you can double click the file once it is downloaded.
   c. (The file can be downloaded in other ways (for example, using the command-line **curl** utility). The full link to this file is http://www.davidbpython.com/introductory_python/data/ALL_DATA.zip)
   d. When downloaded, if it has not already been expanded into a **python_data** directory, double-click the file and wait until a **python_data** directory appears.
   e. Move the folder (it can simply be dragged using your graphical windows) onto your **Desktop**.
   f. Verify that your Desktop now has the two new folders that you created, side-by-side:

```
myuser:~ myuser$ cd                  # just making sure we're home
myuser:~ myuser$ cd Desktop
myuser:Desktop myuser$ ls            # confirm they're there
a_nice_file.txt
Dropbox
python_scripts                  # (not bolded on your screen)
python_data                     # (not bolded on your screen)
z_nice_file.txt
myuser:~ myuser$
```

10. Congratulations! Now, the mayhem can begin.

Debugging

Here are some problems that can arise when things go wrong (i.e., a step is skipped or done incorrectly - or, heaven forbid, an error in these instructions).

*I can't stress enough how important it is that you stay patient and read every error message. If you find yourself jumping around and guessing, you **must stop**, take a breather, and pay close attention to what you are seeing.*

(note that error messages are approximate)

No such file or directory: this can happen if one or more of the following is true:

- you created the **python_scripts** folder in the wrong directory
- you saved the **hello.py** in the wrong directory
- you are presently in the wrong directory.

Go back to the beginning and follow the instructions to the letter, and meticulously compare your output to my sample output. Make sure that the path to the folder you create is **/Users/*myuser*/Desktop/python_scripts**, that inside **python_scripts** is **hello.py**, and that your **pwd** is **/Users/*myuser*/Desktop/python_scripts** (where **myuser** is the name of your home directory).

Permission denied: you did not **chmod** the file, or possibly you saved the file in more than one location and **chmod**ed the wrong one. **cd** (to go home)**,** then **cd Desktop/python_scripts**, then **ls** to confirm the file is there, then **chmod 755 hello.py**, then **./hello.py** to execute.

[continued]

Bad interpreter : this means that your 'shebang' line is incorrect. It must be the first line of the file, be 'flush left' (meaning no spaces before the line begins) and should look like this:

```
#!/usr/bin/env python3
```

Follow this line to the letter. ***usr*** *has no* ***e****.*

Line 2: print: command not found (or similar): this is the same problem as above; it means that the shebang line is not at the very top left of your script.