Northeastern University
DS2000 – Programming with Data
Spring 2019, Derbinsky & De Vries

Practicum 1: Installing Software & Hello, World!

There are two goals for this week's practicum: (1) to install all the software you will need for the rest of the class, and, (2) to write, run, and test your first Python program!

1 Software Installation

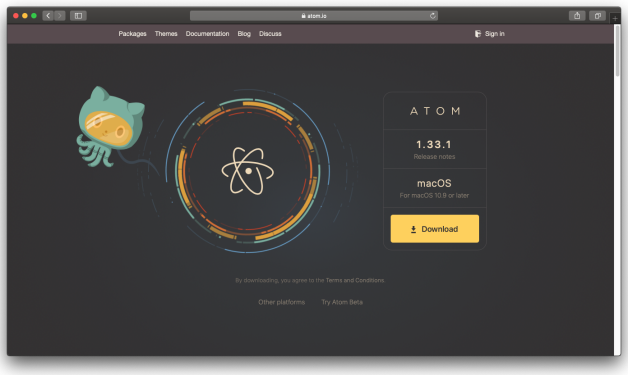
We will install two pieces of software: Atom, a program that is useful for viewing files & writing Python programs, and Anaconda, a common packaging of Python with other fun tools.

1.1 Atom

Atom is one of many "text editors," or programs that allow you to view & edit files containing text. It enjoys the following advantages. It is free, cross-platform (i.e., it works on MacOS, Windows, & Linux), extensible (i.e. the Atom community can write plugins to extend its functionality), and useful for writing Python programs (e.g., it has 'syntax coloring' and visually displays whitespace characters). You are welcome to use other editors if you so choose, but Atom will be the default used in class.

1.1.1 Download

To begin, visit the Atom homepage: `https://atom.io`

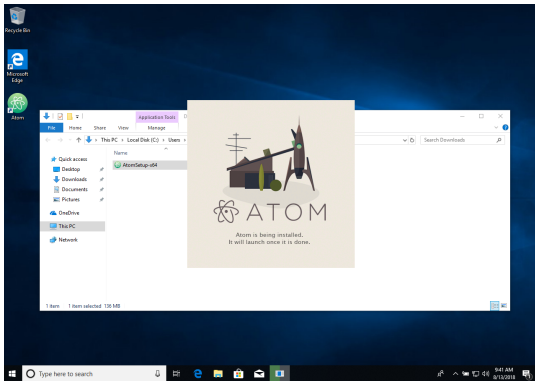


Now click the "Download" button to get the latest version for your platform.

1.1.2 Install

Once downloaded...

- On MacOS, double-click the downloaded file to uncompress ("unzip") it, then drag the resulting App to your Applications folder
- On Windows, double-click to install



1.1.3 Show Invisibles

Python is very sensitive to the type of whitespace you use (either spaces or tabs). Atom can show these visibly to you, which is helpful. To enable visual display...

- On MacOS, click the `Atom` menu, then `Preferences`
- On Windows, click the `File` menu, then `Settings`

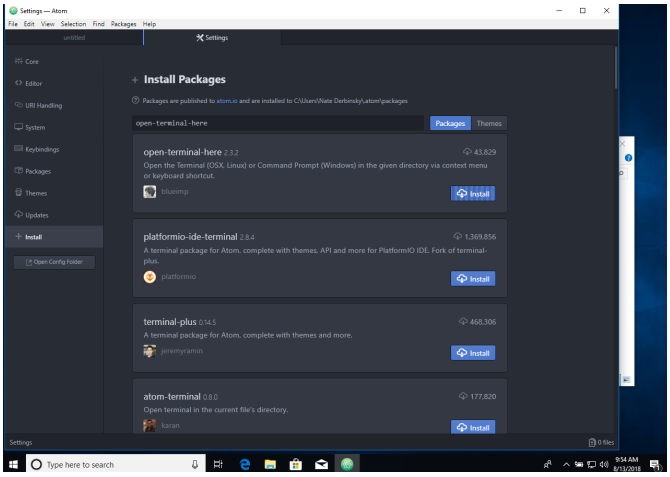
Click the `Editor` tab on the left. Scroll down and check the box labeled "Show Invisibles" – now close the `Settings` tab.

1.1.4 Add a Package

Atom has many "packages" that have been written to extend its abilities. One useful package for this class will be `open-terminal-here`. To install it...

- On MacOS, click the `Atom` menu, then `Preferences`
- On Windows, click the `File` menu, then `Settings`

Click the `Install` tab on the left. Now type `open-terminal-here` in the textbox and click the `Install` button when it appears in the list of results. When installation is complete close the `Settings` tab.



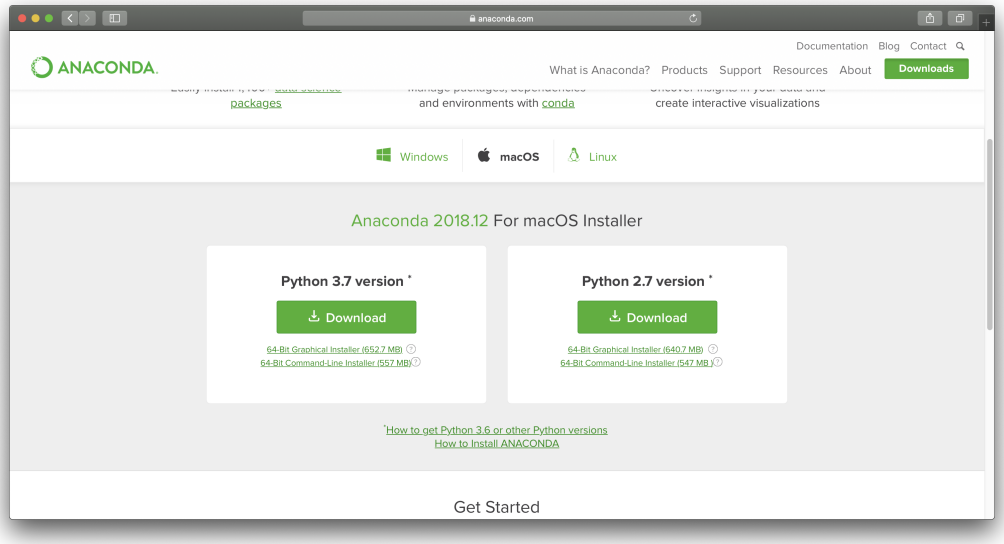
Feel free to explore and install additional packages throughout the semester!

1.2 Anaconda

Python is a language that comes in many flavors and "distributions." In this class, we will use Anaconda, which is quite common amongst Data Scientists. Importantly, we will use **version 3** of Python – version 2 is still somewhat common, but will NOT work in this class.

1.2.1 Download

To begin, visit the Anaconda download page: `https://www.anaconda.com/download/`



Choose your platform and click download for **Version 3.7**. This may take a while ...

1.2.2 Install

Once downloaded, double-click the file to begin installation. For the most part, simply click "Next" to complete the process. Notes...

- **Windows**: make sure to choose the "Add Anaconda to my PATH environment variable" – this will make it easier to use Python and create parity with your MacOS peers
- You will NOT need `VS Code` for this class, so feel free to install that or not
- Try to avoid spaces in the install path (this matters more on Windows, and may require installing for "All Users")
- The webpage has platform-specific install instructions ("How to install ANACONDA") if you get stuck :)
- It's a big package, so give your computer some time for the install

2 Hello, World!

Now that your software is installed, we will test Python by writing and running a very simple "Hello, World!" program.[1]

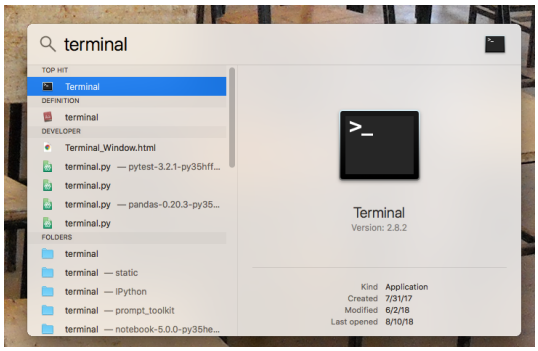
[1] `https://en.wikipedia.org/wiki/"Hello, World!" program`

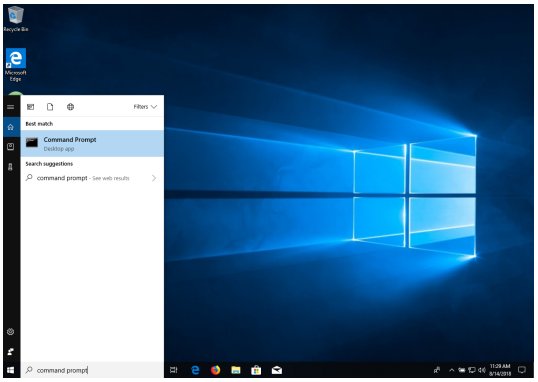
2.1 Interactive Python

Python comes with an "interactive console" tool built-in – this is a program that allows you to type commands and immediately see their results. It's useful for small programs, and for trying examples from the class readings, so we'll start here and then build up to how you'll ordinarily write/run programs (e.g., for your homeworks).

To begin, you'll need to launch your platform's command-line interface or CLI[2]...

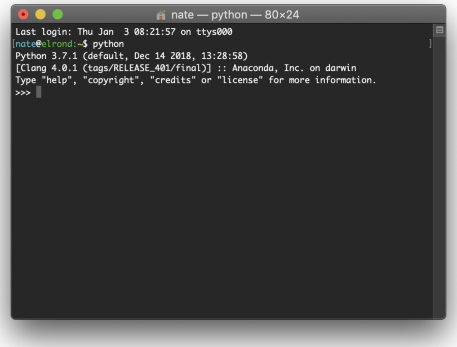
- On MacOS, double-click the `Terminal` application from the `Utilities` folder of `Applications`; alternatively, use `Spotlight` (the magnifying glass in the upper right corner) – type `Terminal` and press `return`/click it in the list.
- In Windows, the program is `Command Prompt`: either via search, Start Menu (typically under `Accessories` in old versions, or now `Windows System`), or using `Run` (Windows Key+`R`, type `cmd`, `OK`).

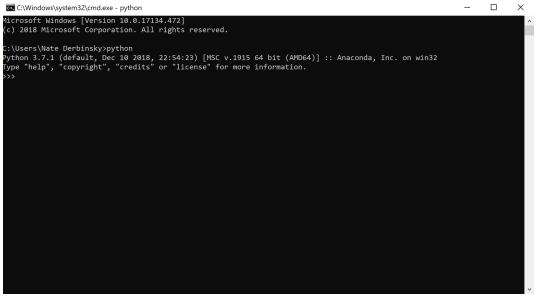




The CLI is a power user's best friend: it allows you to directly tell your OS (e.g. Windows, MacOS, Linux) exactly what to do via commands, as opposed to lots of points & clicks. In a CLI you are presented with a "prompt" that allows you to type a command, which will be executed once you press `return`.

For this part, type `python` (all lowercase; this matters!) and press `return` – you should see something like what is shown in the screenshots, indicating that you started the Python interactive prompt program.

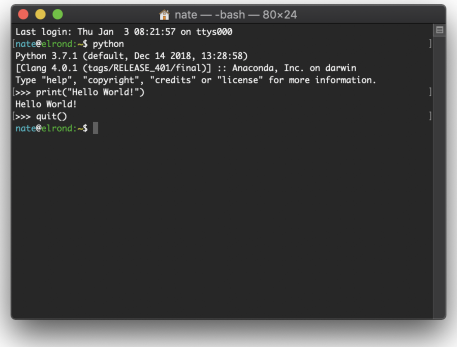


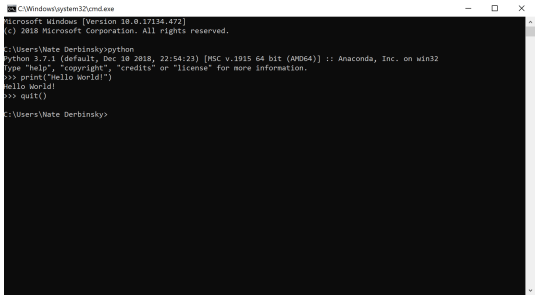


Now all commands we type will be interpretted as Python statements. For example, to tell Python to print something to the screen, type the following and press `return`: `print("Hello World!")`

[2] `https://en.wikipedia.org/wiki/Command-line_interface`

Python should have executed your command and output the result back to you. Feel free to use this program to try other commands, particularly while reading the course text. To exit when you are finished, type the following and press `return`: `quit()`





You will now be back in the Windows or Mac CLI, which you can close.

As mentioned above, the interactive console is great for small programs & testing commands, but not great for even moderate sized programs & submitting work for class. This is why our next goal is to write Python code in a file, and then tell Python to execute the contents of this file. To do this we'll need to take a quick detour to understand some basics about files, file systems, and navigating file systems using your platform's CLI.

2.2 A Quick Aside: File Systems

To run a Python program, it is usually best to be in the same folder (or "directory") as that program. Thus, the goal of this section is for you to have a basic idea of how files & folders are organized (known as the "file system"), and how to navigate this organization (to then run a Python program).

On most modern computers, files are kept within a hierarchy of folders described by its "path" (e.g. `C:\Users\Nate Derbinsky\hello.py` or `/Users/nate/hello.py`). The end of the path is the name of the file, and the pieces leading up to it (separated by \ on Windows, `/` on MacOS) indicate folders within folders that lead to that file. On MacOS & Linux/Unix computers, the root is `/`. On Windows each drive (e.g. `C`, `D`) serves as the root for its own hierarchy. For example...

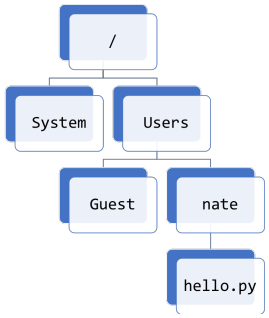
- On MacOS, the path `/Users/nate/hello.py` indicates that `hello.py` is within the "nate" folder within the "Users" folder of the root.
- On Windows, the path `C:\Users\Nate Derbinsky\hello.py` indicates that `hello.py` is within the "Nate Derbinsky" folder within the "Users" folder on the "C" drive.

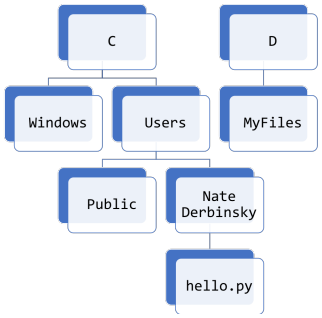
As shown above, it can be useful to visualize these paths as a tree.

OK, back to Python – let's say we know the path of a file and we want to get there, how do we do it via the CLI? So here's the core set of CLI commands you'll need to learn:

1. Where am I in the file system? (MacOS: `pwd`; Windows: `cd`)
2. What's in my folder? (MacOS: `ls`; Windows: `dir`)
3. Change to another folder (MacOS: `cd folderName`; Windows: `cd folderName`)

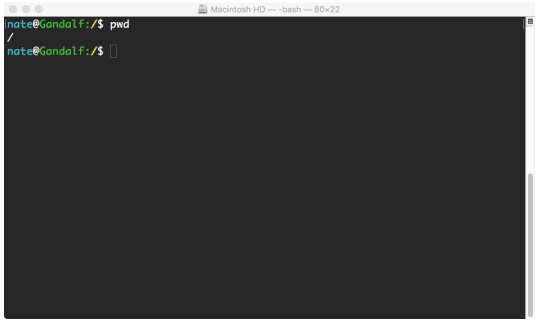
Let's drill down into these a bit.





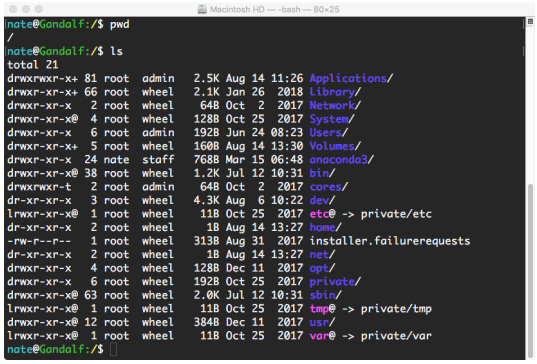
2.2.1 Getting Your Current Path

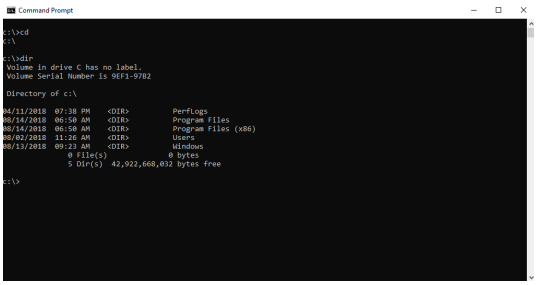
The first command simply tells you your current folder path...



2.2.2 Seeing What's in Your Current Folder

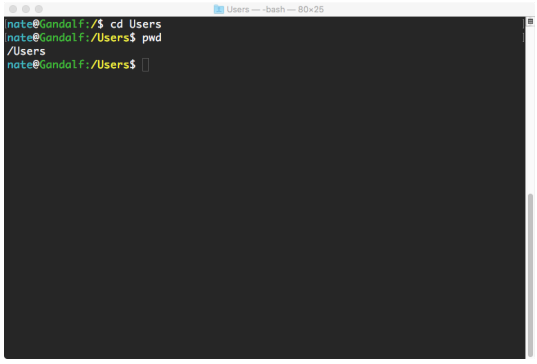
The second command tells you what files and folders are in your current folder...



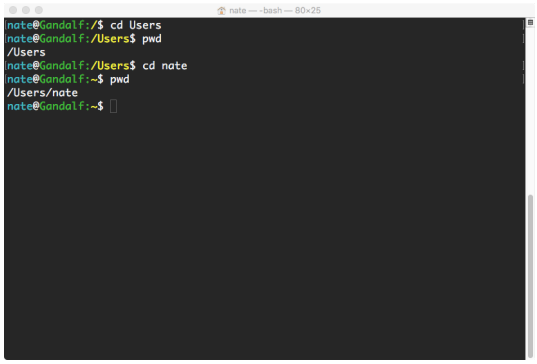


2.2.3 Changing Your Current Folder

Finally, let's use a sequence of CLI commands to get to the same folder as our `hello.py` file...



Note that capitalization matters (and hence the capital "U")...

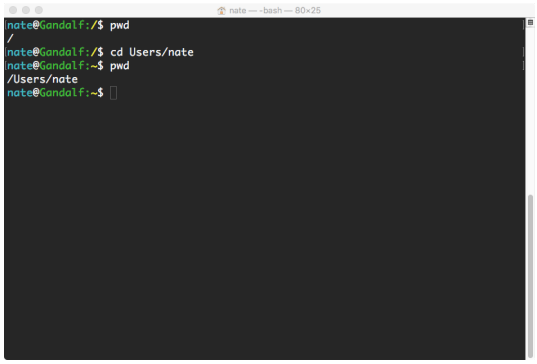


Note #1: folders that have a space should be surrounded by double quotes (hence `"Nate Derbinsky"`).

Note #2: if you want your platform to help finish typing a file/folder name, start it then press the `tab` button for it to auto-complete based on the files/folders it finds.

Note #3: if you want to go "up" a level (e.g. `/Users/nate` to `/Users`, or `C:\Users` to `C:\`), two dots (`..`) means "the directory above" – so `cd ..` means go one level up from my current directory.

Note #4: you could change multiple folders using a single command...

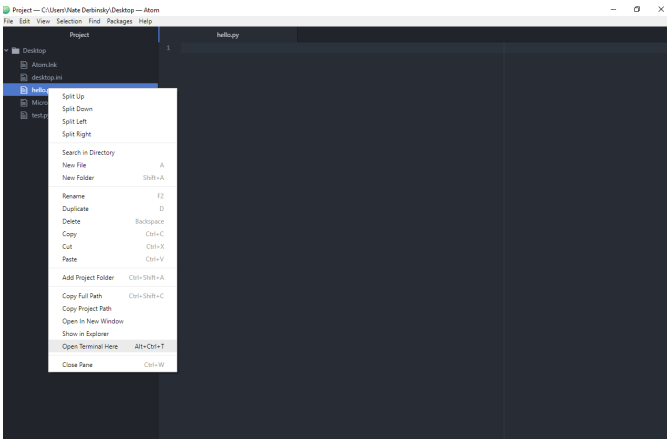


2.3 Writing & Running a Python Program

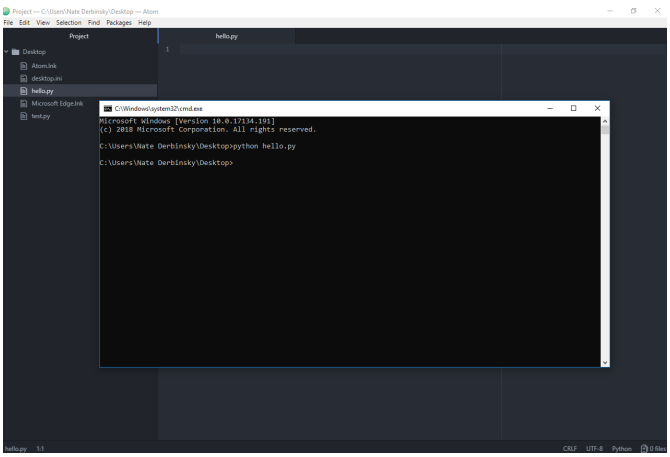
Alright, back to Python :)

If you haven't already, download `pr01_starter.zip` and uncompress/unzip it. Now open `hello.py` in Atom – you may need to right-click, choose "Open With", and then choose Atom.

You will use Atom to write the code of your program and your CLI to run the resulting program (sequence of statements). To run using the CLI, first either use the commands we just covered to change to the directory with `hello.py` OR (as a nifty shortcut, thanks to the `open-terminal-here` package we installed earlier) right-click `hello.py` from the left pane in Atom and choose `Open Terminal Here`.

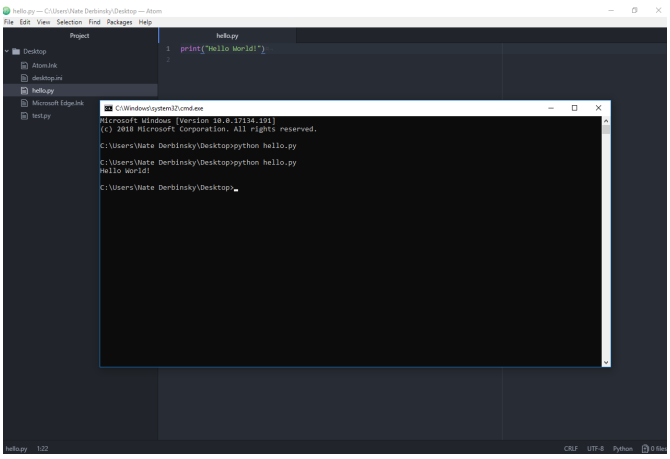


Now in the CLI, type the following and press `return`: `python hello.py`



Don't be too disappointed with the (lack of) output – we haven't told Python to do anything!

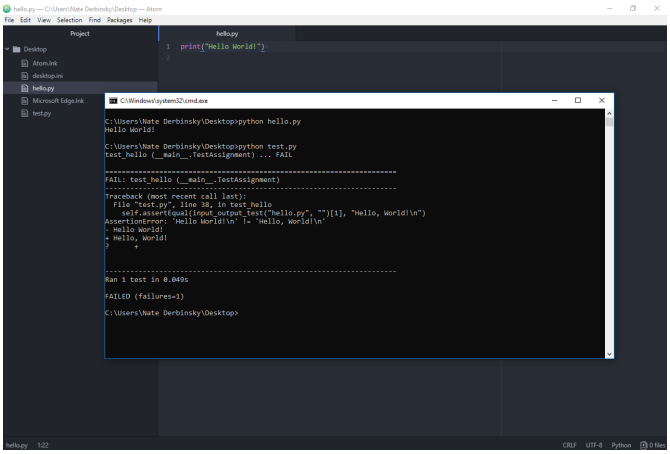
Now, let's add the same print code we used in the interactive console into the file using Atom, save the file, then type the following in the CLI and press `return`: `python hello.py`



Woohoo – you have just written and run your first Python program!!!

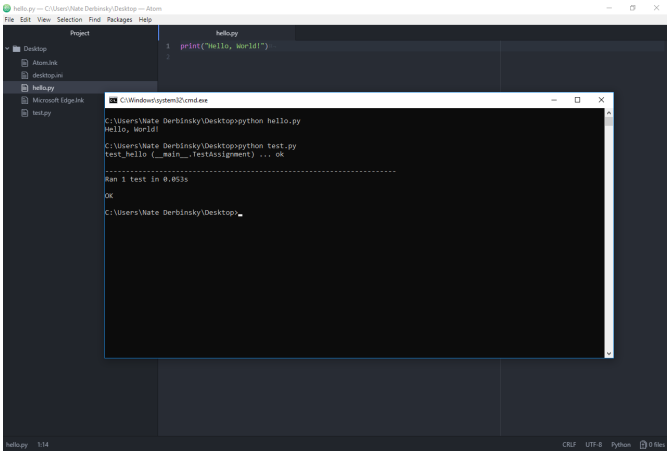
2.4 Testing Your Program

Throughout this semester, we will supply you tests you can run *on your computer* to give you feedback on your work. These tests will always be in a `test.py` file, along with "starter code" you will be provided. To get feedback on your work, simply run this program like any other…type the following in the CLI and press `return`: `python test.py`



It's not the friendliest of output, but the program is telling you that it tried to run `hello.py` and *failed* 1-out-of-1 tests that it ran. That's because it was expecting to see "Hello, World!" but instead got "Hello World!" – notice how picky these tests are, one missing/extra/incorrect letter and it will fail you! Computers are not terribly clever.

So let's fix the program: add the comma in the correct place, save, re-run the program, and finally re-run the test...



Woohoo - all tests pass! In general, passing all tests is not a guarantee of a perfect score (we might have extra tests we run, and will also talk about good style later in the course), but it's a very good sign. **DO NOT** change the tests to pass your program – you're just fooling yourself (since we'll run the unaltered tests during grading).

2.5 Submit Your Work!

Lastly, you should submit your work to Blackboard. First, make a `zip` file of your work (never include `test.py`; in this case the only program you wrote was `hello.py`, but in general you might have more)...

- On MacOS, select your work, right-click and choose `Compress hello.py`
- On Windows, select your work, right-click, `Send To`, `Compressed (zipped) folder`

The resulting file is what you should upload to Blackboard.

NOTE: sometimes mistakes happen and you upload the wrong file – we recommend you download the submitted work, unzip, inspect the contents, and run them to simulate what we will do when we grade your work.