Lab 1: Getting Started in Python

Note: This first lab practical is heavily-based on the first few exercises of Zed Shaw's Learn Python the Hard Way (5th Edition). This is an excellent book with clear instructions for some of the challenging set-up issues so why reinvent the wheel!

Part 0: Setting Up Python

[1]The goal of this part of the practical is to get python (specifically python3) correctly running on your computer. This will include your whole "programming environment" and hopefully everything you'll need for the rest of the course.[2] There are a lot of alternative programming tools and programmers tend to customise their environments heavily. However, while you learn the basics I highly recommend setting the relatively simple environment described below.

What you'll need is the following (there will be step-by-step instructions later on):

1. Jupyter is a programming and data analysis environment for python (and julia, R, etc) that is widely used in science.
2. Python (latest version is 3.13.1). However, the exact version doesn't matter so long as it's newer than version 3.10. Versions of Python (and other software) use numbers to indicate their age, and the position of the numbers determines how much changed between versions. The general rule is that the first number means "major change," the second number means "compatible changes," and the third number means only bug or security fixes. That means if you have version 3.8 and version 3.10, then they should be compatible. If you have versions 3.10.1 and 3.10.2, then there are only minor fixes. On the other hand version 2.7 IS NOT compatible with this course (and should no longer be used in general).
3. A *basic* programming text editor. There are many very complicated text editors that have lots of advanced features but to start out we will use something simple.
4. A Terminal emulator. This is a text based command interface to your computer (as featured in movies as the classic hacker "typing green text into a black screen"). These are very powerful tools for interacting with your computer that we will use later on.

You should have most of the other things you'll need on your computer already, so let's install each of these requirements for your Operating System (OS).

Concise instructions (for more experienced students)

[1] Most of these instructions are lightly modified from Zed Shaw's "Learn Python the Hard Way, 5th Edition"

[2] As students in a CS class you are automatically given a CSID, so if we have a complete nightmare getting things working on your computer we can always https://timberlea.cs.dal.ca:8000

The instructions in this exercise are designed to install most of the things you need for the rest of the course, but if you want to get going quickly with the least amount of work then install:

1. Anaconda to get your python.
2. Jupyter to write and run some code. *This comes with Anaconda so you don't need to install it separately.*
   - On Windows the best way to run Jupyter is to hit the windows key and type `anaconda prompt`. This will start a Terminal with Anaconda. Inside `Anaconda Prompt` type `jupyter-lab`. Later you'll do a better install method that makes this easier.
   - On Linux it should be the same command in your Terminal.
   - On OSX you can either type that command in the Terminal or start the app like normal.

This will give you enough for today's practical, but eventually you'll hit exercises that need the Terminal and Python from the "command line."

Step-by-Step Windows Instructions

Installation on Windows is a bit of a pain so we are going to use a special automated install script from Zed Shaw's Learn Python the Hard Way 5th edition. The install uses two PowerShell scripts to install all the software you need, and is easy to use. First, hit your Windows key to open the Start Menu, then type "powershell" to open a PowerShell instance (*not* PowerShell ISE). Once that's opened enter these two commands:

```
irm https://learncodethehardway.com/setup/base.ps1 -outfile base.ps1
powershell -executionpolicy bypass .\base.ps1
```

Be sure you stay at your computer while it runs. You'll be asked to enter your Administrator password for various installers, and some of them time out if you don't do it quickly enough.

You should now *close this PowerShell window* and open a new one. In the new PowerShell window you repeat the process for `python.ps1`:

```
irm https://learncodethehardway.com/setup/python.ps1 -outfile python.ps1
powershell -executionpolicy bypass .\python.ps1
```

This script will install Official Python *and* Anaconda (due to some issues with anaconda alone on windows). When it is ready to install Anaconda it will give you an important message to read about forcing Anaconda to add Python to the PATH. **Be sure you read it and do as it says, no matter what Anaconda tells you, add it to the PATH!**

When Anaconda is done installing you want to close that PowerShell and open one more PowerShell window. In this new window type:

```
conda init powershell
```

If you are told that `conda` doesn't exist then you either didn't start a new PowerShell window, or you didn't tell Anaconda to add itself to the PATH. Uninstall Anaconda and rerun `powershell -executionpolicy bypass .\python.ps1`, but this time check the box that says add it to the PATH. Then run `conda init powershell` one more time.

After that, close PowerShell for the final time, hit your Windows key, and type "terminal" into your start menu. You should see a new application `Terminal`. Run this and you should be able to do the test at the end of this exercise.

macOS

The setup for macOS is almost the same as Windows, except you don't need a separate Terminal program as macOS already has one:

1. Anaconda This is the easiest way to get a working Python install. When you install it make sure to add it to your `PATH`, even if it warns you that this will cause problems.
2. Geany is a great little text editor for programming. It supports almost everything you need, is small, runs on low power computers, and runs on nearly every OS. I *highly* recommend this for when you're starting out.
3. Terminal is already installed on your Mac, but you will want to add it to your Dock. Hit `command-space` and type "terminal" to start it. Once it's running, right click on the icon in your Dock and select the option to keep it in your Dock.
4. Jupyter will be your primary programming environment for a while. *This comes with anaconda so you don't need to install it.* You can think of Jupyter as an editor and terminal all in one program. There's even psychopaths who do all of their work only in Jupyter. It's a great tool for interactively working on ideas and doing data analysis.

Linux

I love linux and exclusively used it… however, any new operating system has a learning curve and you are already going to be navigating the challenge of learning how to program. Therefore, I don't recommend installing linux for this course! The following instructions are just in case you already use it.

The Linux required software is almost exactly the same, but Linux distro will have its own way to install and manage software.

1. Anaconda This is the easiest way to get a working Python install. When you install it make sure to add it to your `PATH`, even if it warns you that this will cause problems.
2. Jupyter will be your primary programming environment for a while. You can think of Jupyter as an editor and terminal all in one program. There's even psychopaths who do all of their work only in Jupyter. It's a great tool for interactively working on ideas and doing data analysis. *Jupyter comes with Anaconda so you don't need to install Jupyter separately.*
3. Geany is a great little text editor for programming. It supports almost everything you need, is small, runs on low power computers, and runs on nearly every OS. I *highly* recommend this for when you're starting out.
4. Whatever terminal program your version of Linux already has will be fine.

Testing Your Setup

> *NOTE* Before we continue, I'm going to use the word *Terminal* whenever I mean "start PowerShell" if you're on Windows, or "start Terminal" when you're on macOS/OSX. Just remember, every time I say "*start your Terminal*" it's just a placeholder for `PowerShell` on Windows or `Terminal` on OSX.

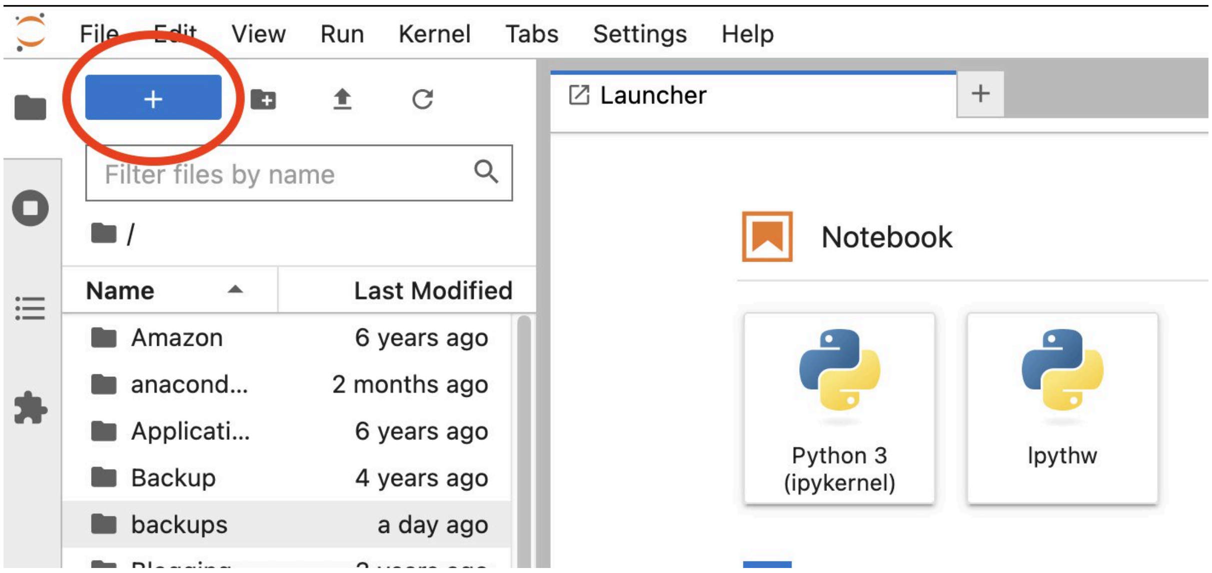
Once you have everything installed go through these steps to confirm that everything is working:

1. Start your Terminal (PowerShell on Windows, Terminal on OSX, etc.) and type this command exactly, spaces and all: `mkdir csci2202`.
2. Once that works you have a directory `csci2202` that you can place your work.
3. Go into that directory with the command `cd csci2202`. This command "moves" your terminal into that directory so your work is saved there.
4. A "directory" is also called a "folder" on Windows and macOS. You can make the connection between the "directory" in your terminal and the "folder" you normally see by typing `start` on Windows or `open` on macOS. This opens the current directory into a graphical folder window you're used to normally seeing. If you're ever lost, type that.
5. The `start` command (`open` on macOS) works like double-clicking that thing with your mouse. If you are in the terminal and want to "open" something, just use this command. Let's say there's a text file named `test.txt` and you want to open it in your editor. Type `start test.txt` on Windows or `open test.txt` on macOS.
6. Now that you can open your Terminal and open things while in your Terminal, you'll want to start your editor. This is Geany if you've been following instructions. Start it and create a file named `test.txt` and save it in the `csci2202` directory you made. If you can't find it remember you can open it from the Terminal with `start` (`open` on macOS) and then use that folder window to find it.
7. Once you've saved the file in the `csci2202` directory you should be able to type `ls test.txt` in your Terminal to see that it's there. If you get an error then either you're not in the `csci2202` directory and need to type `cd ~/csci2202` *or* you saved it in the wrong place. Try again until you can do this.
8. Finally, in the Terminal type `jupyter-lab` to start up Jupyter and make sure it works. It should open your web browser and then you'll see the Jupyter app inside your browser. It's kind of like a little website on your personal computer.

Think of these tasks as a kind of puzzle to solve. We'll spend a lot more time discussing how to navigate a filesystem in later labs.

Part 2: Printing and Jupyter-Lab

Open **jupyter-lab** and create a new notebook using the big blue plus sign:

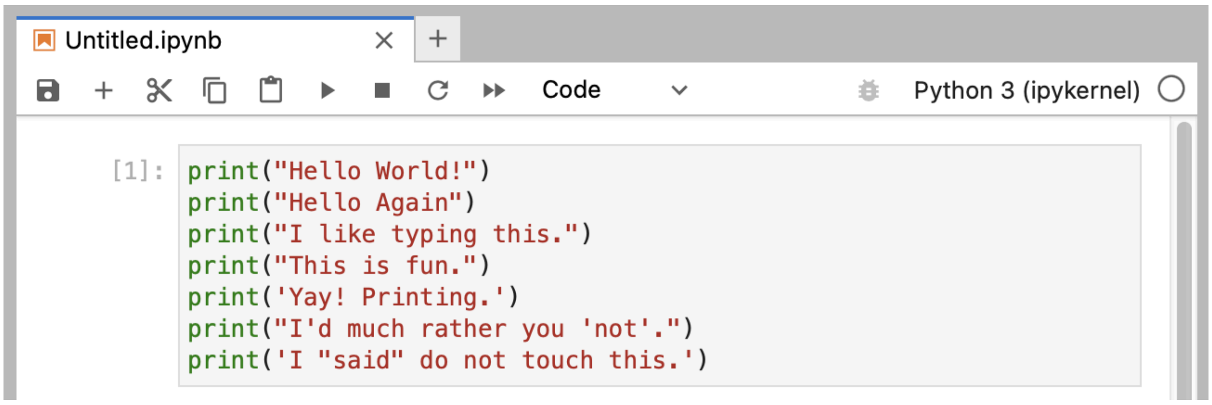


If you don't see the + sign then you may have opened **jupyter** instead of **jupyter-lab** (jupyter is only part of jupyter-lab)

Then type the following into the first bit of the notebook (called a **cell**):

```
print("Hello World!")
print("Hello Again")
print("I like typing this.")
print("This is fun.")
print('Yay! Printing.')
print("I'd much rather you 'not'.")
print('I "said" do not touch this.')
```

Your Jupyter notebook cell should look something like this:



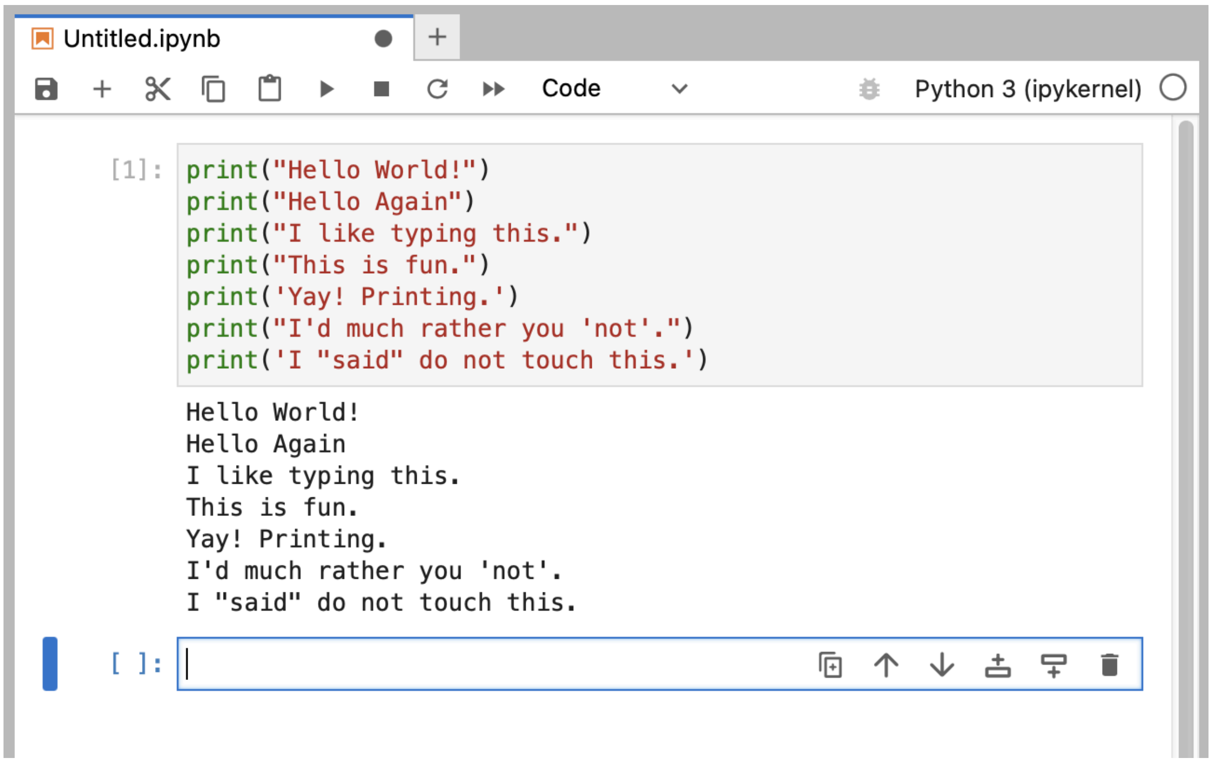
```
print("Hello World!")
print("Hello Again")
print("I like typing this.")
print("This is fun.")
print('Yay! Printing.')
print("I'd much rather you 'not'.")
print('I "said" do not touch this.')
```

Don't worry if your Jupyter window doesn't look exactly the same; it should be close though. You may have a slightly different window header, maybe slightly different colors, and the left side of your Jupyter window won't be the same, but will instead show the directory you used for saving your files. All of those differences are fine.

When you create this cell, keep in mind these points:

1. I did not type the line numbers on the left. Those are printed in the book so I can talk about specific lines by saying, "See line 5..." You do not type line numbers into Python scripts.
2. I have the `print` at the beginning of the line, and it looks exactly the same as what I have in the cell. Exactly means exactly, not kind of sort of the same. Every single character has to match for it to work. Color doesn't matter, only the characters you type.

The Jupyter output will look like this after you hold `SHIFT` and hit `ENTER` (which I'll write as `SHIFT-ENTER`):



```
[1]: print("Hello World!")
     print("Hello Again")
     print("I like typing this.")
     print("This is fun.")
     print('Yay! Printing.')
     print("I'd much rather you 'not'.")
     print('I "said" do not touch this.')

     Hello World!
     Hello Again
     I like typing this.
     This is fun.
     Yay! Printing.
     I'd much rather you 'not'.
     I "said" do not touch this.

[ ]:
```

You may see different window appearance and layout, but the important part is that you type the command and see the output is the same as mine.

If you have an error it will look like this:

```
Cell In[1], line 3
  print("I like typing this.
        ^
SyntaxError: unterminated string literal (detected at line 1)
```

It's important that you can read these error messages because you will be making many of these mistakes. Even I make many of these mistakes. Let's look at this line by line.

1. We ran our command in the Jupyter cell with `SHIFT-ENTER`.
2. Python tells us that the cell has an error on line 3.
3. It prints this line of code for us to see it.
4. Then it puts a `^` (caret) character to point at where the problem is. Notice the missing `"` (double-quote) character at the end though?

5. Finally, it prints out a "SyntaxError" and tells us something about what might be the error. Usually these are very cryptic, but if you copy that text into a search engine, you will find someone else who's had that error, and you can probably figure out how to fix it.

Now try the following:

1. Make your script print another line.
2. Make your script print only one of the lines.
3. Put a # (octothorpe/pound/hash/mesh) character at the beginning of a line. What did it do? Try to find out what this character does.

Comments

Comments are very important in your programs. They are used to tell you what something does in English, and they are used to disable parts of your program if you need to remove them temporarily. Here's how you use comments in Python:

```
# A comment, this is so you can read your program later.
# Anything after the # is ignored by python.

print("I could have code like this.") # and the comment after is ignored

# You can also use a comment to "disable" or comment out code:
# print("This won't run.")

print("This will run.")
```

From now on, we will write code like this. It is important for you to understand that everything does not have to be literal. If my Jupyter looks a little different from yours, or if I'm using a text editor, the results will be the same. Focus more on the textual output and less on the visual display such as fonts and colors.

Part 3: Numbers and Math

Every programming language has some kind of way of doing numbers and math. We discussed the basics of integers, floats, and their operators in the lecture. Try typing in the following and modify them to work out what each of the operators below do:

```
print("I will now count my chickens:")
print("Hens", 25 + 30 / 6)
print("Roosters", 100 - 25 * 3 % 7)
print("Cockatrice", 2 // 4)
print("Now I will count the eggs:")
print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)

print("Now I'll count my ridiculous exercises:")
print(2**5)

# I can also combine expressions with print statements:
print("What is 3 + 2?", 3 + 2)
print("What is 5 - 7?", 5 - 7)
```

Q1: Fill in what you think each of these does for ints (e.g., + does addition):

- + plus
- - minus
- / slash
- // double slash
- % percent
- * asterisk
- ** double-asterisk

Now try them using floats:

```
print(3.1 + 4.2)
print(8.19 - 1.2)
print(4.2 / 2.0)
print(7.8 // 1.25)
print(6.8 * 9.2)
print(9.99 ** 10.4)
print(8.2 % 4.1)
```

Q2: Do they all work the same if operating on 2 floats?

Part 4: Science and Variables

Consider throwing a ball up in the air. The vertical position of the ball y changes with time t according to the following equation (where v_0 is the initial velocity and g is the acceleration due to gravity):

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

Q3: Given v_0 = 5m/s, g = 9.81 m/s^2, using python, what is the position of the ball at t = 0.6s?

To make this calculation more convenient to edit you can assign variables (*note: we are explicitly casting the floats t and y to str using* `str(t)` *to display them, there are several semi-automatic ways to do this in python which we'll explore in later classes*):

```
# Create a variable for storing initial velocity in m/s
v_0 = 5

# A variable for storing the acceleration due to gravity in m/s^2
g = 9.81

# Create a variable for storing the desired time in seconds
t = 0.6

# Calculate the ball position in metres
y = v_0 * t - 0.5 * g * t**2

# Inform the user of the current ball position
# we can do this in multiple ways: explicitly casting the floats y and t
to strings:
print("At t = " + str(t) + "the position of the ball is" + str(y))
# we can also do this using commas to automatically do the casting
print("At t = ", t, "the position of the ball is", y)
# the "best" practice current is to use f-strings as these are easy to
read and allow specific formatting of the numbers such as a fixed number
of decimal places
print(f"At t = {t} the position of the ball is {y:.2f}")
```

Q4: Edit the above code and re-run the cell to calculate the position at t=0.9s

Q5: What does the "+" operator do when used on strings**?**

Q6: Create and submit code for that converts temperatures from Celsius (C) degrees into corresponding Fahrenheit (F) degrees with the formula: $F = \frac{9}{5}C + 32$

The code should print output in the format:

16 degrees Celsius is 60.8 degrees Fahrenheit

Q7: Use this code to provide the temperature in Fahrenheit for 2 degrees Celsius, -32 degrees Celsius, and 1000 degrees Celsius.