

Getting familiar with Python (and a few other data science tools) as a bench scientist

Key things to familiarize yourself with

Updated 06/04/25

In addition to Python itself, here are key Python modules, programs, etc. to install, learn, etc.

- **Numpy** – for dealing with numerical data
- **Pandas** – for working with data tables
- **Matplotlib** – for plotting
- **Seaborn** – for plotting (based on matplotlib but sleeker)
- **Plotly** – for interactive plotting
- **Jupyter notebook/Jupyter lab** – for organizing & working interactively with code
- **Google colab** – for organizing & working interactively with code
- **Anaconda** – for installing packages and managing virtual environments
- **Github** – for finding and saving code
- **Google** – for when you get stuck!
- **Basic command line** – bash etc. for navigating directories (folders), running scripts, etc.

Python basics

- **data types**: strings, floats, integers, lists, dictionaries, tuples, etc.
 - converting between types
- **indexing, slicing, splitting**
 - beware that indexing for most things in Python starts with 0 and doesn't include the end
 - e.g. [0:2] will get you the first and second values
- **functions**
 - a lot of the time there will be optional arguments that make the docstrings look scary but really you don't need to worry about because they have default values set
 - ***args** and ****kwargs** lets a function take an unknown number of input values or key-value pairs
 - can use ***** or ****** when giving input to give a function a list (*****) or dictionary(******) to unpack
- **scope** (global, local, etc.)
 - when you're in a function you are "isolated" from the rest of the file so you need to pass it what you want it to have in the arguments & then have the function return back to you what you want
- **for loops & while loops**
 - check out **enumerate** function for iterating
- **Boolean statements** (e.g. True/False)
 - beware that **=** is to assign values, but **==** is to check whether things are equal
- **comments** (start with **#**) and **docstrings** (surrounded by **' '** or **''' '''**)
 - docstrings are basically like the "manuals" for using a function
- use **help(thing_you_want_help_with)** to get docstrings

Python basics +

- **lambda functions**
 - another one of those things that looks super scary but is really super helpful, especially when working with pandas
- **list comprehension**
 - lets you make lists from lists (filtering out and/or modifying the values)
- **string formatting** (check out f strings)
- **regular expressions**
 - search for specific strings

Key modules

Modules contain extra functions, etc. that aren't included in core Python & are tailored for various specialized purposes (graphing, calculating, etc.)

- **pandas** (for working with data tables - think Excel on steroids)
 - import pandas as pd
- **numpy** (for working with arrays of numbers, plus a bunch of helpful science/math functions)
 - import numpy as np
- **matplotlib** (for plotting things)
 - import matplotlib.pyplot as plt
 - %matplotlib inline
- **seaborn** (for plotting things more prettily & easily)
 - import seaborn as sns
- **plotly** (for interactive plotting)
- **Biopython** (for working with DNA and protein sequences, etc.)

importing modules

- You can import a module from a .py file containing functions you want to use (e.g. module_name.py).
 - o You can make this module with your own custom functions or import a module from the standard library or a non-standard module that you have already installed.
- If a module is not in your current directory, make sure Python knows where to look for it.
- **basic import** (import module_name)
 - o if you just use import "module_name" you will have access to the functions in the module - but each time you want to use one, you will need to direct Python to the module (using dot notation) by its full name
 - i.e. module_name.cool_function()
- **import as** (import module_name as nickname)
 - o Module names can be long and you probably don't want to have to type out long names each time you want to use a function, so you can use "import" module "as" nickname to simplify the typing.
 - i.e. import module_name as mn
 - For example, we commonly "import pandas as pd" and "import numpy as np"
- **importing individual functions** (from module_name import cool_function)
 - o if there are only a couple of functions you need, or if there are some you want to use a lot, you can import the function directly and then you don't need to specify the module
 - o you can combine with "import as" if you want to give it a nickname
 - For example, we commonly "import matplotlib.pyplot as plt"
- **Importing everything** (from module_name import *)

- o Using the asterisk imports everything from a module with its names so you never have to specify the module name for them
- o This is typically **not recommended** because
 - Different modules may use the same names for different functions & then Python wouldn't know which you wanted
 - It can make it harder to de-bug things if there's a problem with a module (you can't easily see where it comes from)
- **autoreload**
 - o you can use the autoreload magic command to reload the module, incorporating any changes you made to it you will need to install the autoreload extension if you haven't yet more on its usage
here: <https://ipython.org/ipython-doc/3/config/extensions/autoreload.html>
 - o This is helpful when you're making changes to a module and want to use the updated module without having to restart the kernel and rerun everything, etc.

At the top of any notebook I typically include:

```
Import numpy as np
Import pandas as pd
```

And if I want to do any graphing I add

```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
%matplotlib inline
```

Jupyter notebook - makes working with code much easier and more interactive, modular

a few helpful things

- **shift-tab** will give you lots of info, including docstring, in a pop-up window
 - for objects, it will provide data type, length, etc.
 - for functions it will pull up doc-string
 - you can have that contextual info always available on the side by pressing command+I
- **%who** will list all your variable names
 - you can also add a data type to search for after who if you just want to look for strings or something (e.g. %who str)
- **%whos** will list them and tell you about them (datatype, etc.)
- **tab** will auto-fill (or show you auto-fill options)
- **X** will delete a cell
- **B** will insert a cell below
- **c/m/r** to change to code, markdown, raw
- can run shell script in a notebook if you start the line with !
- can comment/uncomment code with **command + /**
- **File** → **new console** for notebook will let you run a console interacting with your notebook

- Under the **Help tab** there are manuals for numpy, pandas, etc.
- if you use **ipykernels** you can switch between different conda environments, choosing different kernels

Learn some basic command line

A couple really key bash commands are

- **pwd** tells you your Present Working Directory - where is the terminal running from - this will dictate what files it has access to
- **cd** lets you Change Directory if you want to move to a different one
- **cd ..** takes you up a level in directories
- **ls** lists what's in a directory
- **rm** deletes something
- **mkdir** makes a new directory (folder)
- use **up arrow** to go up to previous commands you've typed in
- use **tab** to autofill file names, etc.

Where applicable, talk to your IT department or lab mates or whoever can help to learn how you can connect to your institution's remote server so you can run process-heavy stuff. You typically will need to use something like "ssh myname@servername."

- learn how to move things to and from the server (e.g. with "scp" - and technical note that if you want to move folders (aka directories), include -r for recursive).

Virtual environments - for isolating packages to avoid conflicts

- use **Anaconda** to install packages in virtual "conda" environments

Git/GitHub - allows for version control

- Here's GitHub's Quickstart guide: <https://docs.github.com/en/get-started/quickstart>
- And here's a link to download GitHub Desktop: <https://desktop.github.com/>
- I'm in NO WAY an expert on this. In fact, I only started using Git/GitHub for my own work this morning. But am really excited about keeping track of all the fiddling around with code I'm doing - and doing that fiddling around without worrying I'm messing up things that worked and/or I liked <https://youtu.be/o1cNRKF4nes>

Learn basic R

This is another programming language that you might need or want

- **RStudio** is a great free user interface for working with it
- check out **ggplot2** for making fancy plots - there's a great gallery of examples (with source code) here: <https://r-graph-gallery.com/index.html>
- Also check out (or at least be aware of) **tidyverse**. It's a sort of "alternative" to "base R" which is the "normal" R language, and it lets you use more human-like language (things like "select" and "filter" and "group_by" rather than a bunch of brackets and dollar signs and numbers).
 - There are some great cheat sheets you can find here: <https://www.rstudio.com/resources/cheatsheets/>

more about why it's useful for scientists to learn some basic computer stuff (and some key things to focus on) in this previous post of mine: blog: https://bit.ly/why_program ;
YouTube: <https://youtu.be/hoi7g5tqj44>

picking up pandas

I highly recommend learning pandas (the Python data analysis library) if you want to do any sort of data analysis or think you might want to in the future. Yes, wet work scientists this means you too! Those results you get at the bench will need some work to analyze and Excel won't cut it!

Start with getting Python fundamentals down:

- Python Programming Fundamentals playlist, Joe James:
<https://youtube.com/playlist?list=PLj8W7XlvO93paRBPCuGFGEc94xJofOgO5>
- 5 Common Python Mistakes and How to Fix Them, Corey Schafer:
<https://youtu.be/zdJEYhA2AZQ>
- 10 Python Tips and Tricks For Writing Better Code, Corey Schafer:
<https://youtu.be/C-gEQdGVXbk>

Then get into pandas:

- Official docs
 - pandas getting started guide:
https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html#getting-started
 - pandas User Guide:
https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html#user-guide
 - 10 minutes to pandas:
https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html#min
 - pandas cheat sheet:
https://github.com/pandas-dev/pandas/blob/main/doc/cheatsheet/Pandas_Cheat_Sheet.pdf
- My favorite resources is this Pandas Tutorials playlist from Corey Schafer:
<https://youtube.com/playlist?list=PL-osiE80TeTsWmV9i9c58mdDCSskIFdDS>
 - It has all the fundamentals & it's broken up by topics (indexing, filtering, grouping, sorting, etc.) so you can easily find what you need
- Project Data Science YouTube channel: <https://www.youtube.com/@ProjectDataScience>
 - 80/20 NumPy and 80/20 Pandas, Pandas Mega-Tutorial
- Python pandas - An Opinionated Guide by Data Talks:
https://youtube.com/playlist?list=PLgJhDSE2ZLxaENZWWF_VOua5886KiUd15
 - I especially recommend the “Stack, Unstack, Melt, Pivot - Pandas” video if you're finding yourself running into weird multi-index problems after grouping
<https://youtu.be/kJsiiPK5sxs>
- I also did a video on pandas tips for getting a quick overview of a dataframe (head, head.T(), info, describe, etc.) <https://youtu.be/-yqUrYAXXi4>

You can plot data from pandas dataframes and I like using **Seaborn** to do this. It's built on matplotlib (which you can use without Seaborn as well but Seaborn makes things easier and prettier). Here's a good Seaborn tutorial:

- Seaborn Tutorial: Seaborn Full Course by Derek Banas: <https://youtu.be/6GUZXDef2U0>

I like working with **Jupyter Notebooks** in Jupyter Lab. More tips for that here:

- Jupyter Notebook Tutorial, Project Data Science <https://youtu.be/DKil6NfSle8>
- 5 Jupyter Notebook Tips & Tricks to Improve your Data Science Workflow!, Keith Galli <https://youtu.be/YuWZNV4BkkY>
- I also did a video on this, Jupyter Lab/Notebook tips - things I find helpful as a beginner learning (not an expert tutorial) <https://youtu.be/vsXktUm2hOM>

If you are using Jupiter Lab, you can access the pandas documentation under the Help tab, and you can access docstrings (usage guides) for various functions by pressing shift+tab at the end of the term.

corresponding YouTube video: <https://youtu.be/-UPjEOG7WYM>

helpful resources

Google & Youtube - so much good stuff out there! Below are some of my favorite

note: If you're super new to coding, if possible, try to take an actual course (maybe through your local community college or there are lots of free courses online). I took an intro comp sci course in undergrad then had a short basic course in the beginning of grad school that helped me familiarize me with concepts, which is making things way easier now. Once you have the basics down, I find the best way to learn is really just to find projects that are relevant to your work and try to figure out how to answer questions or perform tasks you'd be doing anyway! Also, check out GitHub for examples and real code you can work with and modify.

- this is a really great intro to Python series from Corey Schafer - watch in order if you're just getting started or skip to specific lessons: <https://youtube.com/playlist?list=PL-osiE80TeTskrapNbzxhwoFUiLCjGgY7>
- Jupyter Notebook Tutorial from Project Data Science: <https://youtu.be/DKil6NfSle8>
- Also check out 80/20 NumPy from Project Data Science: https://youtu.be/EmA_TuC2Vdk
- and 80/20 Pandas - Pandas for Data Science from them: <https://youtu.be/Yp3fccNNfjQ>
- Another really great one is "Pandas Tutorial 2021" from Derek Banas: <https://youtu.be/PcvsOaixUh8>
- I also like the videos by Keith Galli, such as this this pandas one: <https://youtu.be/vmEHCJofslg>

- Corey Schafer PANDAS tutorials: <https://youtube.com/playlist?list=PL-osiE80TeTsWmV9i9c58mdDCSskIFdDS>
- here's Seaborn's official guide: <https://seaborn.pydata.org/tutorial.html>
- here's a great YouTube tutorial by Derek Banas I thankfully stumbled upon: Seaborn Tutorial 2021, <https://youtu.be/6GUZXDef2U0>
- here's the guide to matplotlib: <https://matplotlib.org/stable/api/index.html>
- and matplotlib cheatsheets: <https://matplotlib.org/cheatsheets/>
- A great tutorial on Python, Jupyter Notebooks and more from my friend and grad school colleague Dr. Shaina Lu. URP 2021 Programming Course: https://github.com/shainalu/URP_2021_Programming_Course
- basic shell commands: <https://wethegeek.com/mac-terminal-commands-cheat-sheet-everyone-should-have/>

Here's the documentation for Jupyter Lab: <https://jupyterlab.readthedocs.io/en/stable/>

You can install it from Anaconda: <https://anaconda.org/conda-forge/jupyterlab>

Here's a nice guide to getting Jupyter Lab set up with different conda environments by Justin Güse: <https://towardsdatascience.com/how-to-set-up-anaconda-and-jupyter-notebook-the-right-way-d3b7623ea4a>