Ways of installing Python for beginners

I find myself having to go over methods of installing Python over and over, so I'm going to document my usual recommendations here. Each installation method should be in a separate section so instructors can link to whichever one you are supposed to you. However, if you are on your own, I'll also try to provide some guidance.

The hedges

This document is not an attempt to be a corpus of every possible method, tool, or program related to Python. There will be omissions both purposeful and accidental. I'm focused here on the tools that I usually mention to my students first, and may add some in later. I teach beginners to programming, so I'm sticking with general discussion of tools and avoiding discussion of advanced features. If you want advanced discussion of which editor handles VCS the best, then you don't need this guide or you aren't one of my students.

Any commentary I provide is also biased toward the academic/research environment and not meant to speak toward full development shops. They have their own needs and can speak for themselves.

There is no One True answer

Unless you're being told to use something specific by an instructor or boss, then there is no one right answer for which tool to use. Try a bunch out and go with the one that you like the most. It is normal for experienced coders to have multiple tools that are utilized to match the task at hand.

Having a single tool when first starting out is absolutely fine! Getting used to one tool at a time is important. Eventually I hope you have Opinions about that tool, which may sway you to change tools or add another into your tool chest. As a starting framework, I suggest that your preferred tools should fall along a use spectrum from experimental, to light weight, to heavy development.

IDNGAF and don't want to read all this crap

Don't worry, I feel you.

1. Be sure to install anaconda (see Step 1 and Appendix 1 for testing).
2. Just download PyCharm Education edition (see Appendix 2 for connecting it to anaconda).
3. You might want to read my notes on Spyder and Jupyter Notebooks if you're in the data game.

4. Happy coding 😎

I don't want/can't install things on this computer

This is the Not Fun pathway, but sometimes you have no choice.

There are several ways to write/run Python online if you are in a position where installation just isn't working for you, or you don't have admin rights and can't install things. The grand caveat here: all these tools are either very new or very free, which yields some buggy, fussy, or broken behavior. You may end up having to pay for a cloud service if this limitation will really be long term.

Getting access to a command line/interpreter based Python is pretty easily. However, finding a tool that allows you to edit a script file and have a pleasant experience with running it is harder to find.

Here are some tools that I've used and like:

1. Python Anywhere
    - Pros:
        - lots of console options, including full bash and variety of SQL console support
        - upload/download of files straight forward
        - a totally-good-enough-to-start free account
        - cool extras if you get a paid account
    - Cons:
        - has Python script editing interface, but running the script in the same window is clunky
        - interface can be confusing for beginners, so requires very specific directions
2. Repl.it
    - Pros:
        - writing/running script interface is pretty clean/straight forward
        - easy to share repls.
        - can make classrooms and assignments
    - Cons:
        - uploading data files feels like a hack (this is a new feature when I tested it, so it may be improved later on)
3. Trinket
    - Pros:

        - Good for encapsulating small programs into executable/editable scripts
        - Nice sharing options
        - These embed nicely into webpages or HTML presentations

    - Cons:
        - Not a replacement for an actual development environment, need to pay for Python 3

4. My Binder

    - This one is a little different. You can create a GitHub repo with Jupyter Notebooks and set it up in My Binder to have a live version of the notebooks launched in the cloud.
    - Pros:
        - Free
        - Links to your data and packages well

    - Cons:
        - Requires set up on the instructor side
        - I've found it too unstable to depend on in the classroom, but their support model is very grassroots.

5. Microsoft Azure Notebooks

    - Another cloud based Jupyter Notebook system
    - Pros:
        - works well if you literally only have just a self contained notebook to run and play with
        - More stable than My Binder

    - Cons:
        - You can upload data files that will only persist a long as the container is active. So you'll likely have to upload every time, which isn't great for a classroom situation.
        - Looks like it's free so long as you have a Microsoft or 360 account

There are, of course, options for self hosting cloud things, and there may be campus services if you have an affiliation. But as I mentioned, the options here are passable but not glorious.

Basic installation framework

At a very high level, you're going to have to install two things to work with Python: Python itself and a tool for interacting with Python. There are some built in methods of interacting with Python, and they'll work in a pinch, but most people will choose to use additional tools as their primary form of writing in Python.

Step 1: Install Python itself

I'm going to keep this simple: unless you're being told otherwise, just use Anaconda with the most recent version of Python (e.g. avoid the 2.x versions).

1. Go here: https://www.continuum.io/downloads
2. Download the visual installer for Python 3.6 (or above).
3. Wait, because this is a giant download
4. Install like a normal application thing.
5. Restart your computer
6. Test your install with Appendix 1.
7. Once your installation is done and tested, install your IDEs of choice.

Anaconda provides an installation of Python in your system, but you don't usually directly interact with Anaconda to write Python. You will use specific editor tools to write Python, and those tools are connected (in a variety of ways) to your installation of Python. Thus, those tools will launch Python and execute the code. So the installation of Anaconda is necessary, even if it doesn't look like you'll be using it.

You'll start interacting with Anaconda more directly once you need to start doing more advanced environment management, and is the subject for many tutorials and documentation that is not necessary to repeat here.

Step 2: Pick an IDE and install it

Refer to the Grand Trio of Tools section to decide on which might work the best.

Step 3: Test and connect things

This step will depend on what you've installed and how, so follow any directions you've been given to do so.

Step 4: Go code stuff and feel good

Why do you need a specific tool?

We're very used to conducting our writing and document creation in things like Word or Google Docs, but that isn't the right tool for the job here. Those programs are designed to create highly formatted text content for humans. While you still have to interact with code as a human and that reading experience is important, you're

creating code documents to send off the computer. So you need to write your scripts in plain text.

What is plain text?

This is the difference between a .txt file a a .docx file. Plain text documents are just the characters literally written into the file. No formatting or other fanciness. Whereas Microsoft Word and Google Docs are optimized for formatting and writing formatted documents, plain text documents are the content stripped of all that. You can open a plain text file in pretty much any document reader, but you can only open a Word document in specific programs.

Plain text is important when writing code because there's a lot of secret sauce behind how content is displayed in a Word document versus plain text. That extra content introduces a lot of invisible garbage that will break any tool trying to do stuff with that program. You need that plain literal text for your scripts. Some of those script files are transformed in further ways, but they also need to start out as plain text. This is why you should never attempt to write Python code or any other programming code in Word or another formatted document editor. It is technically possible, but is a deeply unfortunate experience.

Plain text doesn't mean plain tools

The text documents you're writing with your code may be "plain" text, but the tools good at doing so are anything but! There's a lot of value add in tools that are aware of the language that you are writing. They can help watch out for errors, and perform a lot of common but fussy tasks. Think about making webpages or writing blog posts for something like Wordpress, you certainly could bust out the HTML and write it all by hand, but having a tool that can automatically generate error free templates for you saves you time and cognitive power you need for the task at hand.

The selection of writing tools that you prefer to work with is completely up to you! What's important when you're first starting out is playing around with them and seeing how they fit into your workflow. Many developers have very strong opinions about which tools and services they prefer in the writing experience, which you are welcome to accept or reject as you like.

The Grand Trio of Tools

This is at least how I like to think about it, and you're welcome to disagree. I usually recommend having three levels tools in mind for use:

1. The "heavy" tool, an IDE that you like and can help you keep track of things in a large project. This may start as your sole tool as you're learning Python and can use the help of an IDE and then you start incorporating other tools as you get more advanced skills.

2. The "light" tool, which is likely just an advanced plain text editor rather than a full IDE. This tool is good for short/simple programs, or for code you need to have running for days on end (e.g. web scraping or large processing tools).
3. An interactive tool for experimentation, exploration, testing proof of concepts, etc.

I encourage you to have a combination of 1 & 3 or 2 & 3 depending on your preferences. Some full IDEs also have interactive Python tools built in, so a single program might satisfy 1 & 3. I will provide recommendations for each category.

For example, my personal trio is:

1. I actually have two:
    - PyCharm education edition for large processing scripts and instruction
    - Jupyter Notebooks for data exploration/munging

2. Sublime Text Editor
    - For large web scraping projects to run for ages or when I'm writing a one-off tool.

3. IPython
    - Interactive console that I will use in conjunction with 1 or 2 to experiment or check how data types interact.

Heavy editors

Don't forget that you'll also want to install Python via anaconda before installing an IDE!

There are many full IDEs for Python, so if you care about a specific feature to be present in an IDE, I suggest you search for that feature among lists of IDEs. I aim to keep this list short because when you're a beginner long lists can be difficult to navigate.

Summary:

1. Are you doing scientific python?
    - Yes: Spyder or Jupyter Notebooks (try both see which works the best for your projects)

2. Are you doing data analysis/science and want the pandas things to be pretty?
    - Yes: Jupyter Notebooks

3. Are you just getting started with programming?
    - Yes: likely PyCharm Edu or Wing 101 will suit you the best, presuming that you're using traditional educational materials. You might want to do Jupyter Notebooks if you're going down the data analysis track.

PyCharm

Links: https://www.jetbrains.com/pycharm/

I usually have my students use PyCharm Education edition because it:

- has a cleaner interface
- less noisy with warning, style prompts, etc.
- as a nice big green button for running scripts
- still capable of connecting to anaconda environments

- Pros
    - Several free versions
    - Can be connected to anaconda environments
    - Good hooks to version control
    - Has a built in IPython interactive console
    - Project oriented enforced file structure
    - has a lot of pro tools as you move up to more complex tasks
    - Looks the same for mac/windows (as an instructor, this pleases me)
    - Looks like a normal application that you double click to open

- Cons
    - Sometimes too many style warnings, etc.
    - Complex setup windows
    - Project oriented enforced file structure (yup, repeating this as it might annoy some people, or be too much for quick/small scripts)

Spyder

Links: http://pythonhosted.org/spyder/

This is not an environment I've used much, but some people really like it. This is an IDE designed for scientific computing, and will feel very similar to RStudio or Matlab.

- Pros
    - Free and comes as part of the anaconda installation
    - Variable view can be great for beginners
    - Has console and scripting view in a single display
    - Works very nicely with plots/data viz
    - Will feel very comfortable to an R or Matlab switcher

- Cons
  - Can be resource intensive, so not great for smaller computers
  - Requires command line to launch, which not all users will have permissions to do
  - Interface might be cluttered to some newcomers
  - Not as many syntax error warnings as PyCharm

Jupyter Notebooks

Some will argue that this isn't a 'real' IDE, and my response to that is "I don't care." I don't need to repeat any of the glory of Jupyter Notebooks for data analysis, and there are other great tutorials worth spending time with if you like this.

- Pros:
  - Interactive exploration
  - Many tools automatically display nicer in the notebook environment
  - Heavily used and the standard tool in some scientific/data communities.
  - Easily sharable analytics
  - Inline data viz
  - Has nice presentation tools
  - You can store markdown content to capture narrative, documentation, etc. all in one document

- Cons:
  - Best for solo work, shared editing not awesome yet
  - Not quite plain text (it's giant json), so you have to interact with it rendered in a web browser
    - there are ways to export it out to HTML for easy sharing, but that's static and not interactive
  - The cell interaction can be confusing to some beginners and allows for accidental non-linear traps if you aren't prepared

Others I haven't used

Not to just dump a bunch of links in here, but that's exactly what I'm about to do.

- Eclipse has Python plugins
- Wingware (http://www.wingware.com/)
  - They also have Wing 101 for beginners, which is a minimal editor for beginners: http://www.wingware.com/downloads/wingide-101

Lighter weight editors

These editors are usually just fancy plain text editors that have specialized features for interacting with code and may even be able to run the code for you. Most of these perform just as well as any other, so just pick one and roll with it. I'm not going to get deep into these discussions of each because of that, but these are ordered in my own personal preference order. There is an order here, but all these tools are commonly used and very good. You're very welcome to disagree with my preferences 😀

- Sublime Text Editor
- Atom Editor
- Text Wrangler
    - Mac only
- Notepad++
    - Windows only

Interactive tool

This one is pretty easy, because you don't need a very feature rich environment, and there are some stock options built into your Python installation. The interactive tools are great for when you're checking out data types interact, just doing quick tests if packages will import correctly, etc. Some programs, like PyCharm, have a Python console built in, so that's an easy choice. Others, like Jupyter Notebooks, are natively interactive, so that may be sufficient (but caution, it can add clutter to your notebooks or mess up your local variables).

There are three choices here:

1. Use the stock command line Python instance, which is also IDLE but not an application. You can follow the directions in Appendix 1 to access this program, and it does require a bit of command line work. This will be your better choice if you're working with specific environments and have conda or PyPI packages installed.
2. As an improvement to stock IDLE, use IPython. This should be automatically included with your Anaconda installation, and is accessed the same way as regular IDLE (in option 1). Instead of typing `python` you type in `ipython`. This environment has lots of extra tools, and copy/pasting multiple lines of code plays nicer in this environment. This is the one that I recommend.
3. Use the IDLE application. When you download the Python application from Python.org it'll effectively be this. This is a completely sufficient, if limited, environment to work in. This option that the bonus of being a regular program you can double click to open, and avoid working in the command line. Caution: this option will not be connected to your anaconda installation or any conda environments you make, so your PyPI or conda installed packages will not be available (this is technically possible, but so fussy that it isn't worth the headache).

Appendix 1: Testing your Python installation

1. Be sure to have restarted your computer.
2. Open your command line/prompt.
    - Windows users: open up your Command Prompt application (in your system search bar, type in 'cmd' and an application will open up. YouTube has videos explaining this more if you need it.
    - Mac users: open your Terminal application. Either search via spotlight for 'terminal' or open it from Applications -> Utilities -> Terminal.

3. Within your command line application, type in ‘python’ (without quotes) and press return.

    - Windows users: you should not get an error that it doesn’t know what Python is.
    - Mac users: it should state it is Python 3.6 (or whatever anaconda said it was) and not Python 2.7
    - Should have text roughly like the text below.
    - `Python 3.6.1 |Anaconda 4.4.0 (x86_64)| (default, May 11 2017, 13:04:09)`
    - `[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin`
    - `Type "help", "copyright", "credits" or "license" for more information.`
    - `>>>`
    - (ed note: doing block quote code in lists sucks in markdown)

4. Presuming everything described here as worked, you can close the window. This was just a test.

Appendix 2: Hooking PyCharm to your anaconda environment

These directions presume that you have successfully installed Anaconda (or some other Python environment) into your computer. Review Appendix 1 for directions.

1. Download (Available here: https://www.jetbrains.com/pycharm-edu/download/) and or other PyCharm edition that you'd like.
2. Open the application.
3. Select "Create New Project" when prompted in the opening screen.
4. On the next screen you're going to tell PyCharm where your anaconda installation is and more information about your project.
5. In **Location**: this is where you're going to save your project files. You're going to be presented with a file path that ends with Untitled. You can click the ... button next to it to navigate to another folder. Then change Untitled to something reasonable, but leave the rest of the path in place.
6. Under that is **Interpreter**: Click the drop down and let it populate with things. It might take a few seconds

to find your anaconda. You should select the path that has 'anaconda' in it.

    - Windows: you might only see one option, which will be the anaconda bath.
    - Mac: you will see paths that will say 2.6 or 2.7 and one will likely be selected by default. It'll have System/Library/Frameworks at the beginning of the path. You need to find the Anaconda path, which should be something like `~/anaconda/bin/python`. This is the one that you want (or the relevant anaconda environment that you created elsewhere).

7. **Be sure that you are selecting a 3.x version of Python!** Yes, it matters.
8. Once you have set the correct path and Python, click Create.
9. Now you're in the project view for PyCharm. On the left is your project file directory, and you have a gray screen which is where your scripts will appear when you have some.
10. Right click on the project folder name on the left (this will have the project name that you gave it in step 5).
11. In the right click menu, select New -> Python file
12. Give it a name like 'testing'. You don't need to provide a file extension. Click OK.
13. You should now have a blank white text file screen where the gray screen was.
14. If this is your first time through with PyCharm, it may need several minutes to index your Python folder. Eventually you should see a big green button near the top left of that script.
15. In the first line of the script, write `print("hello world")`.
16. Click that big green button!
17. The bottom of the screen should change to have an output screen appear, and it should read "hello world" followed by some blank lines and "Process finished with exit code 0". This means that everything worked just fine and you're ready to go!

Bonus: check that you have Python 3 hooked up

1. Go into your Python project you just created, or really any PyCharm project.
2. Make a new Python file.
3. On the first line of the script file, write `print "hello"`.
4. Click the big green button.
5. You should get an error in the output area that says `SyntaxError: Missing parentheses in call to 'print'`. You're good to go if this happens! This should fail.
6. If this actually printed `hello` to the output area, then you have Python 2.x hooked to this project.
7. Go into the Preferences or Settings pane (depending on if you're windows or mac), and you should see the project name as one of the options in there. Click on that and redo the Interpreter settings found in the previous set of directions (starting at item 6).