Basics of Writing and Running Python Code

Jason D. Josephson | Last update: January 16, 2024

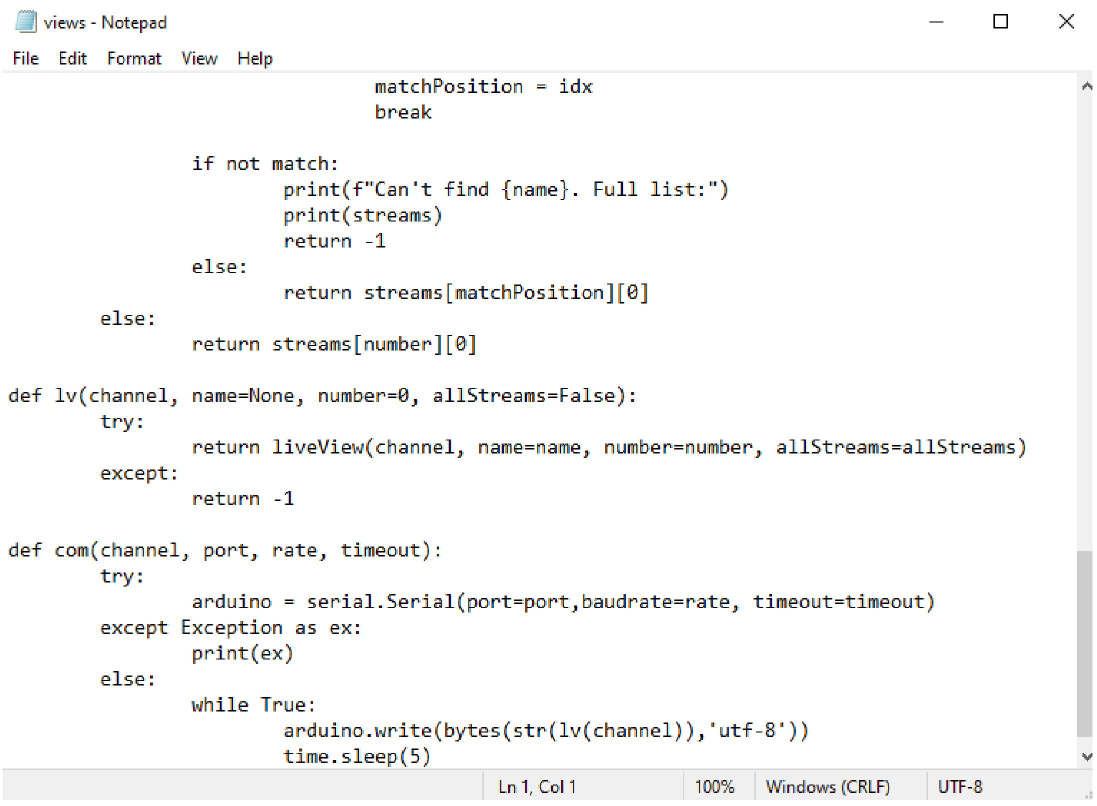
This short document will just cover the very basics of what you need to get started writing and running Python code. No programming experience is assumed. The upshot is you'll need some program to write the code and some program to execute it. I recommend reading the whole document first before installing anything.

Writing code: text editors

Python code is saved in files with the `.py` file extension. No special software is needed to write these files. You can write them in basic text editors, e.g., Notepad on Windows or TextEdit on Mac, and simply save the file as a `.py` file instead of a `.txt` file. This means that the program used to write `.py` files can have nothing to do with executing the code (or even with the Python language as such).

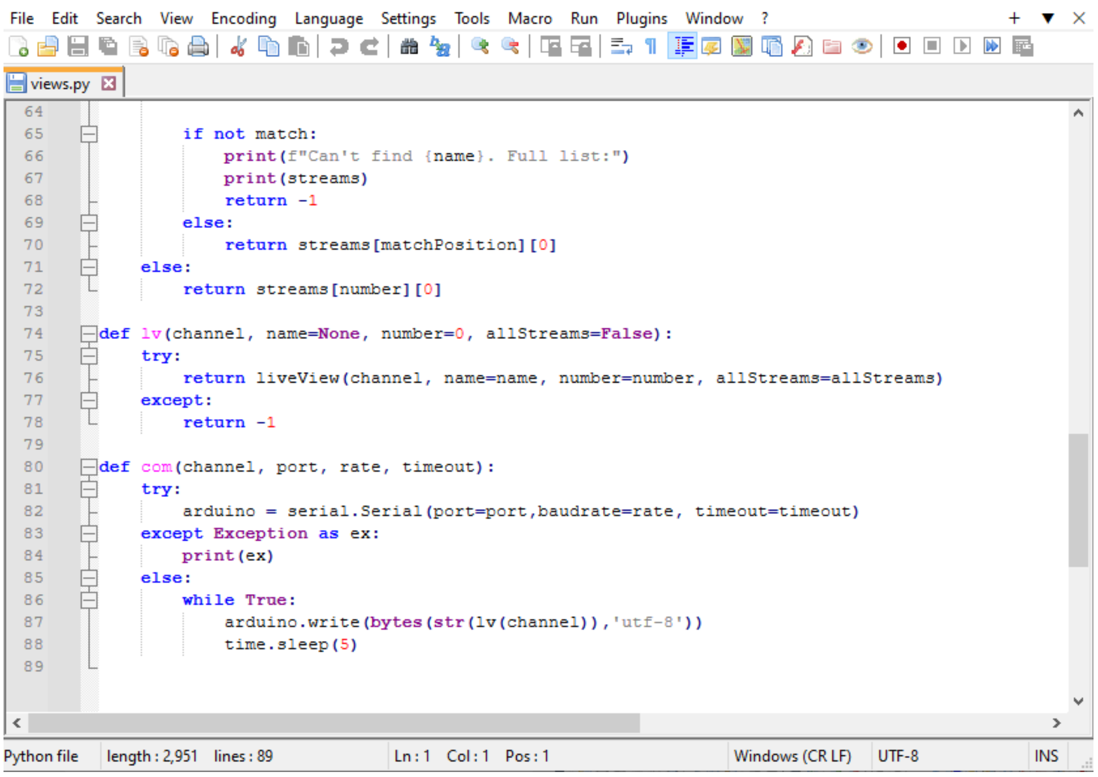
Now, although you *can* use such simple text editors, it will probably be more convenient to use a more advanced writing environment. I will use the **Notepad++** text editor to illustrate a few of these, but we will then list a few other text editors. Notepad++ is a free text editor aimed at maintaining the simplicity of Notepad while still including simple features that are very useful for programming. Unfortunately, it only works on Windows.

Syntax highlighting: terms in the code are displayed in different colours based on what they do. Compare Figures 1 and 2, which show the same code.



```
                                        matchPosition = idx
                                        break

                if not match:
                        print(f"Can't find {name}. Full list:")
                        print(streams)
                        return -1
                else:
                        return streams[matchPosition][0]
        else:
                return streams[number][0]

def lv(channel, name=None, number=0, allStreams=False):
        try:
                return liveView(channel, name=name, number=number, allStreams=allStreams)
        except:
                return -1

def com(channel, port, rate, timeout):
        try:
                arduino = serial.Serial(port=port,baudrate=rate, timeout=timeout)
        except Exception as ex:
                print(ex)
        else:
                while True:
                        arduino.write(bytes(str(lv(channel)),'utf-8'))
                        time.sleep(5)
```

Figure 1: Python file opened in Notepad.

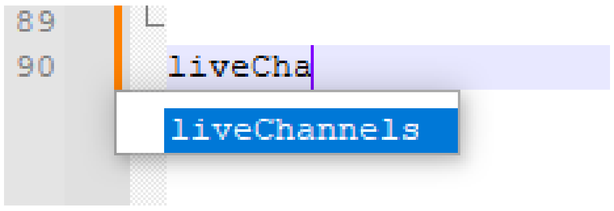

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
views.py

```
        if not match:
            print(f"Can't find {name}. Full list:")
            print(streams)
            return -1
        else:
            return streams[matchPosition][0]
    else:
        return streams[number][0]

def lv(channel, name=None, number=0, allStreams=False):
    try:
        return liveView(channel, name=name, number=number, allStreams=allStreams)
    except:
        return -1

def com(channel, port, rate, timeout):
    try:
        arduino = serial.Serial(port=port,baudrate=rate, timeout=timeout)
    except Exception as ex:
        print(ex)
    else:
        while True:
            arduino.write(bytes(str(lv(channel)),'utf-8'))
            time.sleep(5)
```

Python file length : 2,951 lines : 89 Ln : 1 Col : 1 Pos : 1 Windows (CR LF) UTF-8 INS

Figure 2: Syntax highlighting in Notepad++.

Tab completion: when typing a term already used, pressing the Tab key will automatically complete the rest of the term (Figure 3).

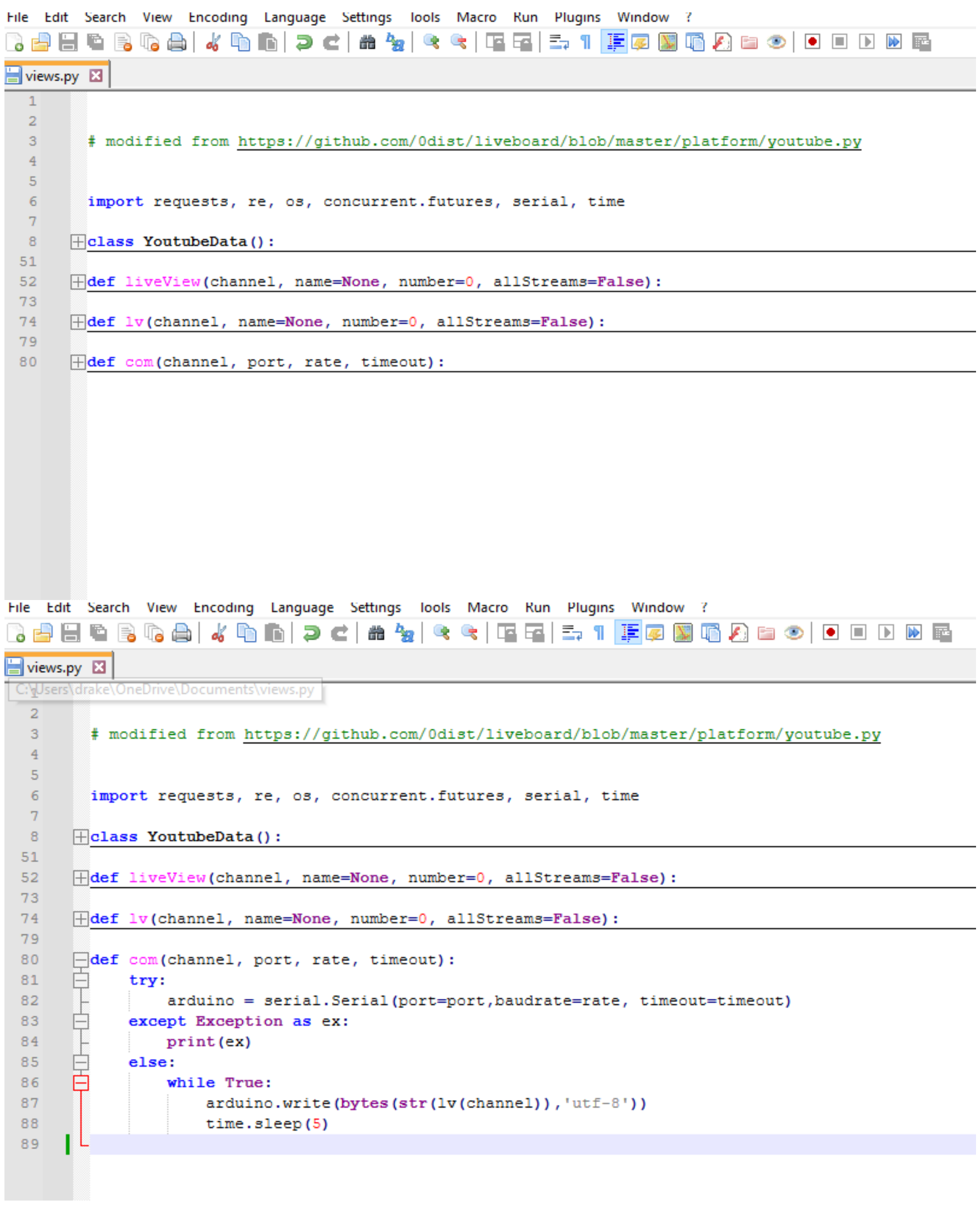


```
liveCha
```

```
liveChannels
```

Figure 3: Tab completion in Notepad++.

Code folding: clicking an icon hides a section of code (without altering the code itself), making scrolling through large files quicker (Figure 4).


File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
views.py
C:\Users\drake\OneDrive\Documents\views.py

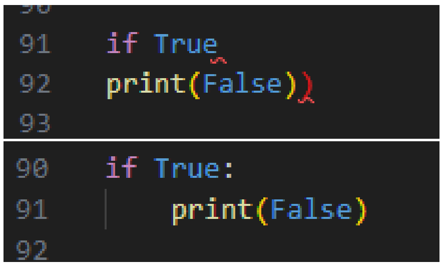
```


# modified from https://github.com/0dist/liveboard/blob/master/platform/youtube.py


import requests, re, os, concurrent.futures, serial, time

class YoutubeData():
def liveView(channel, name=None, number=0, allStreams=False):
def lv(channel, name=None, number=0, allStreams=False):
def com(channel, port, rate, timeout):
```

```

# modified from https://github.com/0dist/liveboard/blob/master/platform/youtube.py


import requests, re, os, concurrent.futures, serial, time

class YoutubeData():
def liveView(channel, name=None, number=0, allStreams=False):
def lv(channel, name=None, number=0, allStreams=False):
def com(channel, port, rate, timeout):
    try:
        arduino = serial.Serial(port=port,baudrate=rate, timeout=timeout)
    except Exception as ex:
        print(ex)
    else:
        while True:
            arduino.write(bytes(str(lv(channel)),'utf-8'))
            time.sleep(5)

```

Figure 4: Code folding in Notepad++. **TOP**: All classes and functions (blocks of code under lines starting with `class` and `def`, respectively) have been "folded" and are not visible. **BOTTOM**: Last function has been unfolded and is now visible in its entirety.

If you just want to keep things simple, Notepad++ is a good choice. If you're interested in more features, there are a number of other editors you may like.

Visual Studio Code (VS Code) is a famous, free code editor which is made by Microsoft which can be customized by installing extensions of which there are a wide variety. For instance, there is a feature which is not in Notepad++ called code "linting" in which the editor identifies some types of errors in your code (Figure 5). (Obviously this depends on the language you're using.) This is obviously very useful, and it can be easily added by installing the "Python" extension. We will see how VS Code can be used to also execute code soon.



```
if True
print(False))
```

```
if True:
    print(False)
```

Figure 5: Code linting in VS Code with the Python extension. **TOP**: Errors in the code are underlined with a red squiggle, and the extra right parenthesis ")" is coloured red. **BOTTOM**: Upon fixing the errors, the red squiggles disappear and all parentheses are in yellow.

Pulsar is a new, free text editor which has a selection of community-made packages; simply go to Settings>Install and search for the package you want. Linting doesn't come with Pulsar upon installation, but linting packages specific to Python can be installed. Pulsar is basically a successor to a major text editor called **Atom**. In 2022 Atom was "sunset" (blame Microsoft); it and its packages have been archived online but are not actively updated, so it's more convenient to get Pulsar.

Sublime is an editor with a reputation for being high-quality, as far as I'm aware, though I've never used it myself since it is proprietary; at the time of writing it was 99 USD (= 132 CAD) for an individual licence. I don't recommend paying until you're confident that you've gotten your feet wet and know this is something you want to invest in.

These are just a few editors. A more comprehensive list is here on the official Python Software Foundation website.

Executing code: interpreters

The code you've written gives the computer instructions to do something. How will you make the computer do it? Python is a "high-level" programming language: its concepts are abstract and don't make reference to the operations directly performed by the computer hardware. The computer itself understands "low-level" 1s and 0s; it doesn't know how to read Python. Thus you need a program to actually make the computer do what you've written, i.e., to execute/run the code.

Again, a programming language is something theoretical, a set of rules. The actual programs which "speak" the language, that is, programs which can turn written code into a real series of actions taken by the computer, are called **implementations**. Different implementations will work in different ways; ideally, all will adhere to the abstract rules of the language, but some may have faster speed or be designed for specific types of computer, etc. Another important term here is code compilation. A **compiler** translates code from one language to another. This could be a translation from a human-readable language into "machine code" consisting of 1s and 0s. A final term is the **interpreter**, a program which executes code that is not in machine language.

The most popular implementation of Python can be found at the official (i.e., owned by the Python Software Foundation) website, python.org. This implementation is called CPython. In it, written Python code (the **source code**) is first compiled into **bytecode**. Bytecode is lower-level than Python but higher level than machine code. The bytecode is then executed by an interpreter. (Figure 6, Top)

Luckily for us, all of this compilation and interpretation goes on behind the scenes. Once you've downloaded the implementation, you'll be able to run your code simply by pressing Enter, and the interpreter will do its magic. (Figure 6, Bottom)

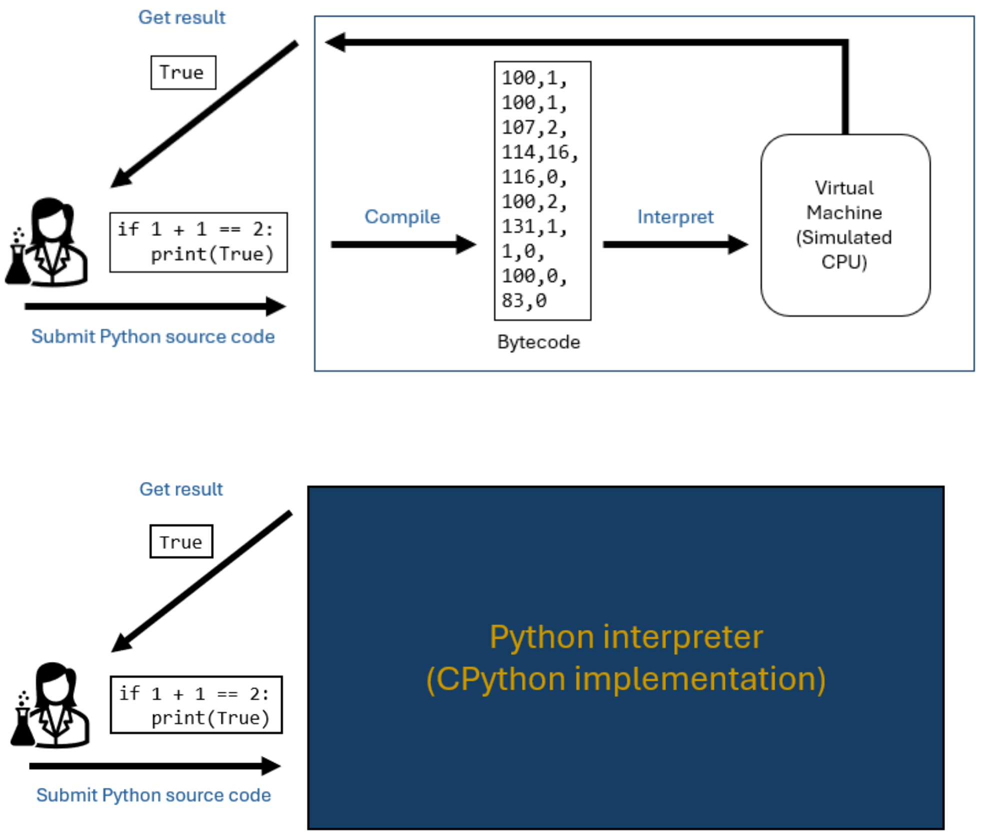
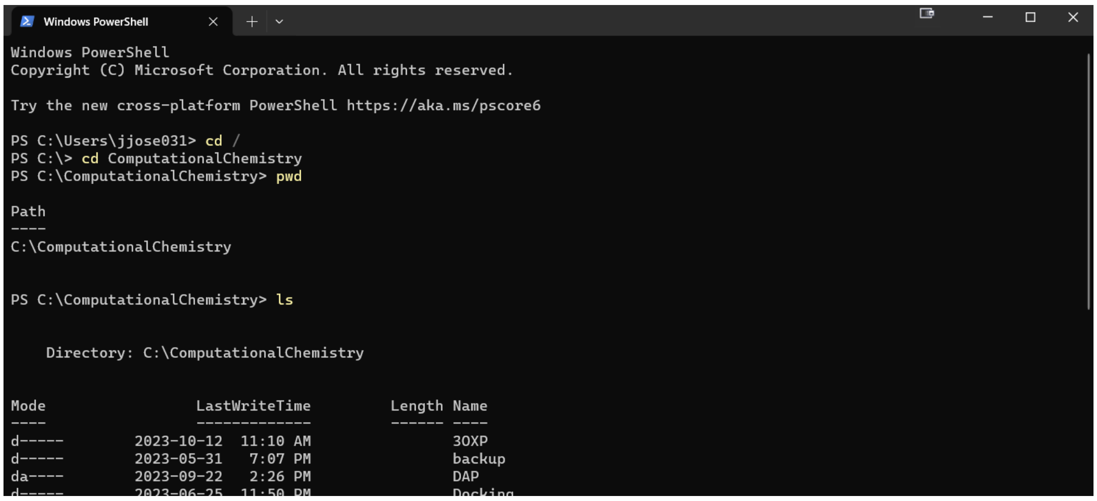


Figure 6: Execution of Python code with the CPython implementation. **TOP**: Simplified scheme of source code compilation and interpretation. If this interests you, read these deeper explanations by Ayan Das and Obi Ike-Nwosu—although you may want to bookmark them and return when you've learned more about Python and computer science. **BOTTOM**: Normally, the user simply experiences the Python interpreter as a black box. Diagrams made in MS PowerPoint.

NOTE: Python version number is important, since different versions may not be compatible. Code written for one version may not work with another, and this is true not only of Python itself but also any code modules/libraries used in a script. This goes especially for the transition from Python 2 to Python 3, which are so different that you should *not* be using Python 2 at all, unless for some reason you have to work with some old code written for Python 2. Some Windows and Mac systems come with Python 2.7 pre-installed, so be sure to install a new version.

Once you've installed your Python implementation, you'll want to use a terminal, some variant of which will be available on Windows, Mac, or Linux systems. On Windows, you can search for Windows PowerShell or install the Terminal app from the Microsoft Store. On Mac, you can use the Terminal app. For Linux, Ctrl+Alt+T should open the Terminal app. This webpage at realpython.com goes over basic terminal commands. The name of the directory (the folder) you're currently in will be displayed in the terminal; by typing `cd` followed by the name of a folder, you can change your current directory to that folder (Figure 7). When specifying folders, one period (`.`) refers to the current directory, while two (`..`) refers to the parent directory of the current directory. So, e.g., suppose you have a folder `parentFolder` which has two sub-folders: `childFolder1` and `childFolder2`. If your current directory is `C:\parentFolder\childFolder1`, to change directories to the other sub-folder, use `cd ..\childFolder2`; or, to change directories to the parent folder, just use `cd ..`. You can also use tab completion, i.e., you can hit the Tab key to autocomplete partially-typed file/folder names. Pressing the up/down arrow keys also allows you to go through commands you've used before.

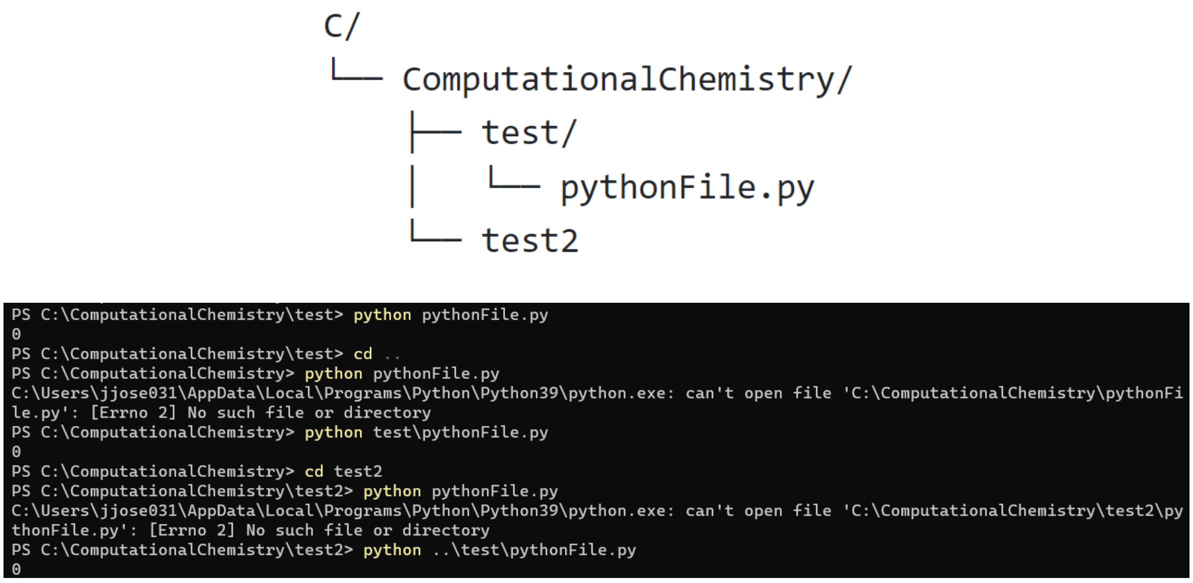


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\jjose031> cd /
PS C:\> cd ComputationalChemistry
PS C:\ComputationalChemistry> pwd

Path
----
C:\ComputationalChemistry


PS C:\ComputationalChemistry> ls


    Directory: C:\ComputationalChemistry


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2023-10-12  11:10 AM                3OXP
d-----        2023-05-31   7:07 PM                backup
da----        2023-09-22   2:26 PM                DAP
d-----        2023-06-25  11:50 PM                Docking
```

Figure 7: Windows PowerShell used in the Terminal app. After the introductory message, "`PS`" (PowerShell) is displayed, followed by the path of the current directory. In the example shown, the commands used are: `cd /` brings us to the root directory (the `C:\` drive). From there, `cd ComputationalChemistry` goes to the `ComputationalChemistry` folder in its parent `C:\` directory. `pwd` (print working directory) prints the current working directory, which is indeed `C:\ComputationalChemistry`. Finally, `ls` is used to list the files and folders within the current directory.

Once you're in the directory containing your `python.exe` application, you can simply type `python` to run the program. To avoid the need to be in this directory, you can add this directory to your system's PATH environment variable. When you run a program, the PATH variable contains a list of directories that will be searched for the program in addition to your current directory. During installation of Python, there should be a check box prompting you to add Python to the PATH. But you can add/remove programs from the PATH at any time; see, for instance, realpython.com's tutorial. Once this is done, you should be able to run `python.exe` by typing `python` in the terminal, regardless of what directory you're in.

If you type `python` followed by a `.py` filename, the code in the entire `.py` file will be executed. Note that if the `.py` file is in your current working directory, you only need to type the filename; but if the file is elsewhere, you'll need to type the filepath relative to your current directory. Think of the relative filepath like a map from your current directory to the file (see Figure 8). Normally, it's convenient to set your working directory to the folder containing the `.py` file(s) you're working with.



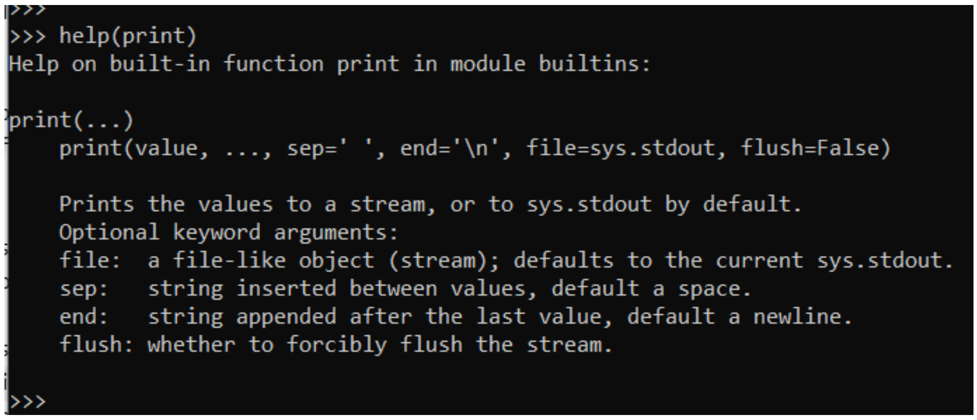
```
C/
└── ComputationalChemistry/
    ├── test/
    │   └── pythonFile.py
    └── test2
```

```
PS C:\ComputationalChemistry\test> python pythonFile.py
0
PS C:\ComputationalChemistry\test> cd ..
PS C:\ComputationalChemistry> python pythonFile.py
C:\Users\jjose031\AppData\Local\Programs\Python\Python39\python.exe: can't open file 'C:\ComputationalChemistry\pythonFi
le.py': [Errno 2] No such file or directory
PS C:\ComputationalChemistry> python test\pythonFile.py
0
PS C:\ComputationalChemistry> cd test2
PS C:\ComputationalChemistry\test2> python pythonFile.py
C:\Users\jjose031\AppData\Local\Programs\Python\Python39\python.exe: can't open file 'C:\ComputationalChemistry\test2\py
thonFile.py': [Errno 2] No such file or directory
PS C:\ComputationalChemistry\test2> python ..\test\pythonFile.py
0
```

Figure 8: Use of terminal (PowerShell in Terminal app) to run a sample Python script, `pythonFile.py`, which simply prints the number 0 when run. **TOP**: Tree diagram of directories used in this example, including `test` folder as the location of `pythonFile.py`. Diagram generated using `https://tree.nathanfriend.io/`. **BOTTOM**: When the current working directory is `test`, the filename `pythonFile.py` can simply be given. After using `cd ..` to change directory to the parent directory, `ComputationalChemistry`, using `pythonFile.py` no longer works, giving an error. Instead, the filepath relative to the current directory must be given, in this case, the `test` folder followed by the filename. Again, after changing to the directory `test2`, which doesn't contain the file, simply using the filename will not work. Instead, the relative filepath now becomes `..` (the parent directory, i.e., `ComputationalChemistry`) followed by `test` and the filename.

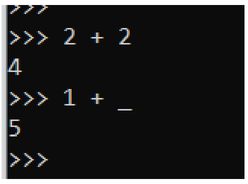
If you simply type `python` without giving a filename, this will allow you to use the Python interpreter interactively. You should also be able to access this mode by clicking on the `python.exe` application directly from your file manager/explorer. This interactive mode is called read-evaluate-print-loop: REPL. Instead of running a whole file of code at once, short bits of code are typed, the Enter key is pressed to execute the code, the output is returned, repeat. This can be much more convenient then having to create and run a file every time one wants to run any snippet of code.

There are a few tricks to using the REPL that come in handy. (1) Type `help(x)` to get information about some object `x` (Figure 9). (2) Like in the terminal, you can use the up/down arrow keys to access previously used lines of code. (3) You can use tab completion. (4) An underscore "`_`" acts as a variable for the most recent output of the REPL. (Figure 10). (5) Pressing Ctrl+C (Cmd+C for Mac) while the interpreter is executing code will halt the execution; this can be important if you accidentally make code that loops forever or takes unreasonably long.



```
>>>
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

Figure 9: Use of the `help()` function in the REPL to provide information about an object, in this case the `print()` function.



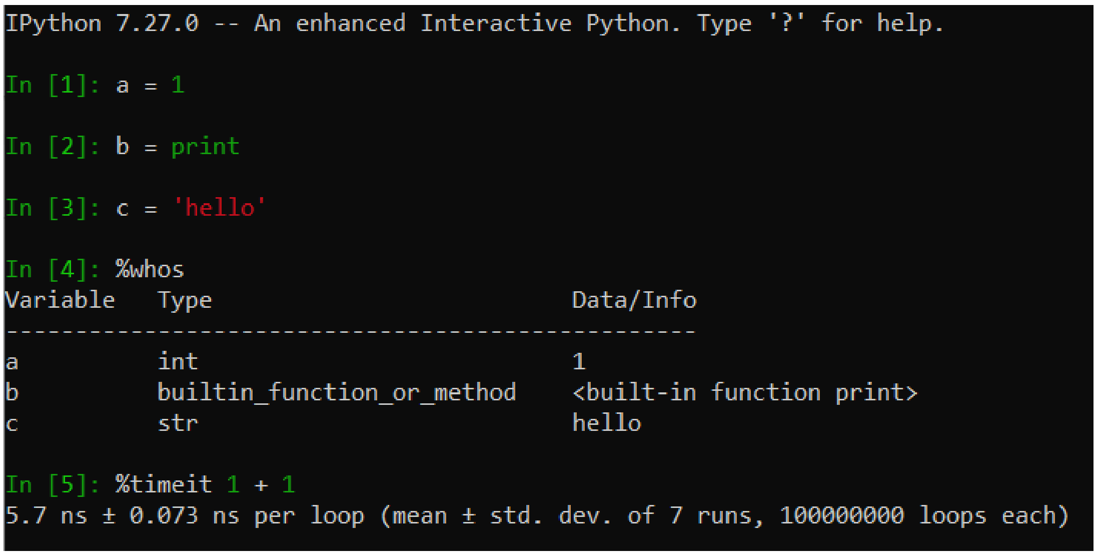
```
>>>
>>> 2 + 2
4
>>> 1 + _
5
>>>
```

Figure 10: Use of an underscore in the REPL. After evaluating the expression `2+2` as `4`, `4` is assigned to the underscore automatically, and the underscore can be used in place of `4` in the next expression.

Another Python interpreter, which I definitely recommend for interactive/REPL use over the one just shown, is IPython (**I**nteractive **Python**). Using IPython works essentially the same way but with some useful added features that don't make things too complex. You should be able to install IPython using the "pip" program. pip should have come with your Python installation; go to the folder containing `python.exe` and look for the `Scripts` folder; pip should be in here. To use pip from the terminal, either navigate into this folder or add this folder to the PATH environment variable (*vide supra*). Then use the command `pip install ipython`, and after installation IPython should be accessible from the terminal.

We will go over a number of distinguishing features of IPython. (1) It uses syntax highlighting. (2) Input code and output results are labelled with numbers in green and red on the left side of the line. The underscore functionality of the prior interpreter is also present, and two (`__`) or three (`___`) underscores act as variables for the second and third most recent outputs. Beyond this, any output of the current session can be accessed with "`_n`", where `n` is the output number. (3) In addition to the `help()` feature, using a question mark after an object gives information about it. For functions, in particular, placing one question mark prints a short description, while two question marks prints the code of the entire function. (4) Ctrl+S allows you to search for a former input containing whatever you type; use the up/down keys to scroll through, and press Escape to exit the mode. (5) Ctrl+A brings your cursor to the far left of the line. (6) If you use Ctrl+C while writing (as opposed to using it to interrupt execution), your current writing will be erased.

In addition to these, there are IPython “magic” commands. These start with a percentage sign. When the `%run` command is followed by the name of a Python file, that file is executed within the interactive session. Using `%timeit` followed by a code expression makes IPython run the code multiple times, returning the mean and standard deviation of how long the code took to execute. `%who and %whos` followed by one or more types of object gives a list of all such objects that have been defined by the user manually; the latter gives more info than the former. Using `%cpaste` begins a mode in which you can use Ctrl+V or a right mouse click to paste code.

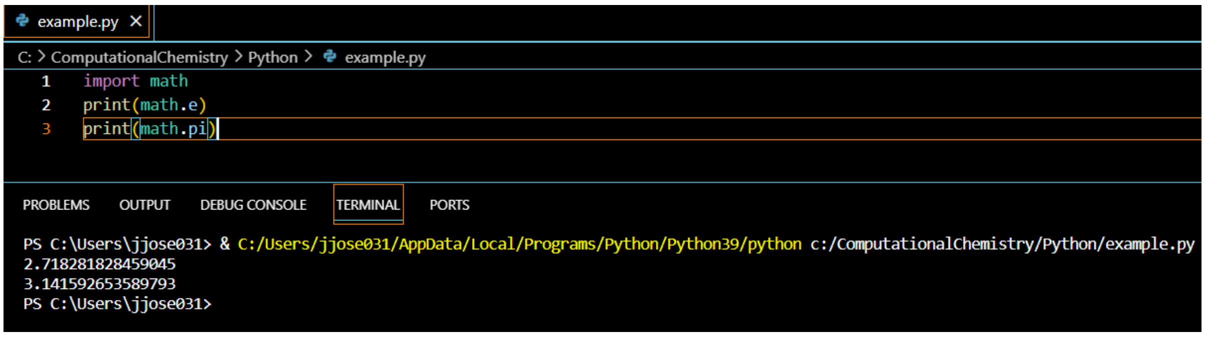


```
IPython 7.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a = 1

In [2]: b = print

In [3]: c = 'hello'

In [4]: %whos
Variable   Type                          Data/Info
--------------------------------------------------
a          int                           1
b          builtin_function_or_method    <built-in function print>
c          str                           hello

In [5]: %timeit 1 + 1
5.7 ns ± 0.073 ns per loop (mean ± std. dev. of 7 runs, 100000000 loops each)
```

Figure 11: IPython interpreter displaying syntax highlighting and magic commands. Note that when `%whos` is not given any object type, it lists all manually defined variables.

A useful feature is the ability to log sessions. To begin logging, use `%logstart` followed by a filename in which your commands will be recorded. Use `%logstart -o` and then the filename to include the outputs in the log. See the link in the former paragraph for more options. Use `%logstop` to stop logging.

Now that we’ve covered text editors and Python interpreters, it’s good to note that some programs, like VS Code, incorporate the use of a terminal directly into the editor. VS Code is elaborate enough that it’s not merely a text editor but rather a “code editor” or “integrated development environment” (IDE). Both the editor and the interpreter can be integrated into a single window (Figure 12), along with the other features and extensions VS Code offers.

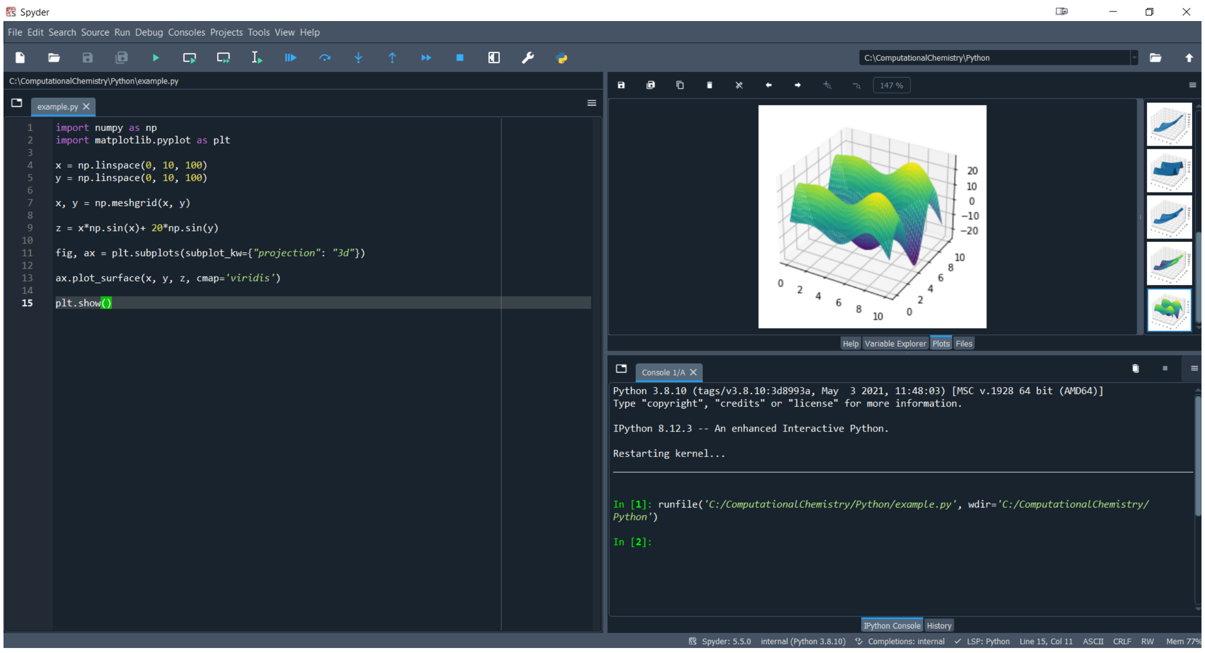


```python
import math
print(math.e)
print(math.pi)
```

```
PS C:\Users\jjose031> & C:/Users/jjose031/AppData/Local/Programs/Python/Python39/python c:/ComputationalChemistry/Python/example.py
2.718281828459045
3.141592653589793
PS C:\Users\jjose031>
```

Figure 12: PowerShell terminal integrated into the VS Code editor. The file `example.py`, visible above, is run, printing out the numbers seen in the terminal below.

On this note, another IDE is **Spyder**. Spyder has multiple panels in a single window for editing Python files, using an IPython terminal, and displaying information, graphs, etc. As in VS Code, code linting is built in and tools are available for debugging the code.



```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)

x, y = np.meshgrid(x, y)

z = x*np.sin(x)+ 20*np.sin(y)

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

ax.plot_surface(x, y, z, cmap='viridis')

plt.show()
```

```
Python 3.8.10 (tags/v3.8.10:3d8993a, May  3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.12.3 -- An enhanced Interactive Python.

Restarting kernel...


In [1]: runfile('C:/ComputationalChemistry/Python/example.py', wdir='C:/ComputationalChemistry/
Python')

In [2]:
```

Figure 13: Spyder IDE. The graph produced by the file in the left panel and run in the IPython terminal (bottom right) can be viewed in the top right panel.

Virtual environments

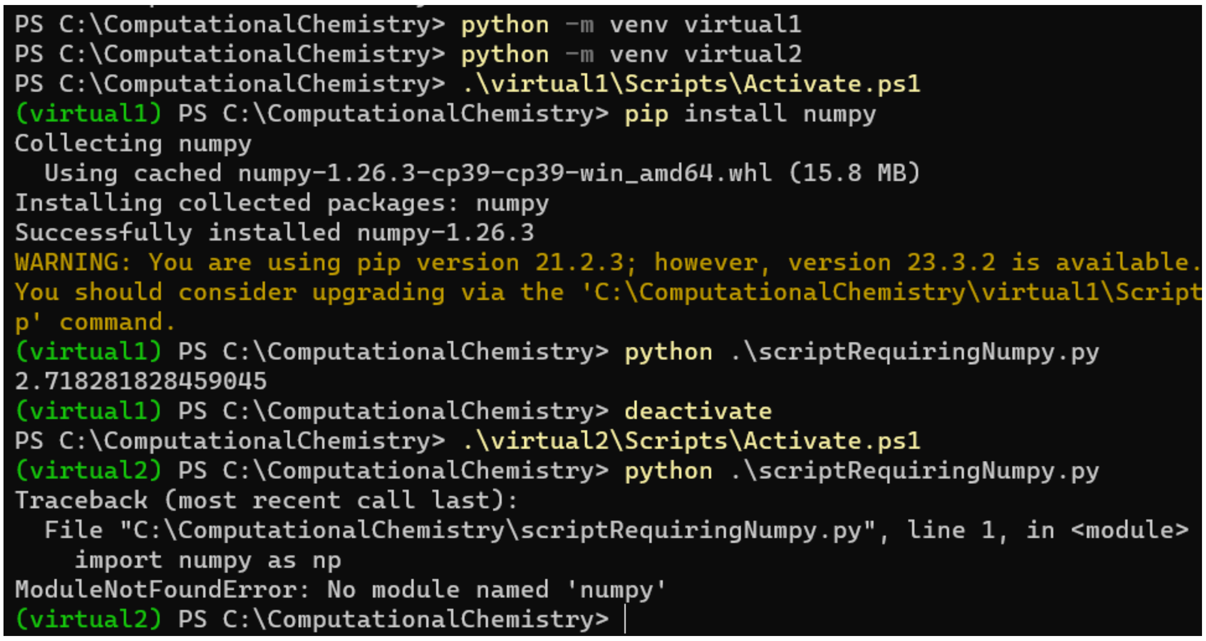
It is often helpful to make use of **virtual environments**. Python has a large number of packages and libraries; these are so numerous and so useful that you will use some in almost every project. But for some scripts/projects, you may wish to use a particular set of packages. Perhaps for one old script you need to use an old version of a package, while for another you need a newer version (which would, if installed, overwrite the old version). Two different packages might also conflict with each other, unbeknownst to you, since packages often depend on other packages (their “dependencies”). Also, you may want to keep everything neat and tidy, with a project having a discrete compartment containing only the packages it needs. This allows you to easily keep track of what packages are needed for what project, especially important when sharing the project.

A virtual environment contains a Python interpreter along with various packages and libraries. When operating inside the virtual environment, this specific interpreter is used, and only the packages inside the environment are available. Any packages installed with pip while in the virtual environment are installed within the environment only and aren't accessible while in other environments.

Creating an environment can be done on Windows or Linux with the command `python -m venv folder` where `folder` is the folder (or path to the folder) of the new virtual environment. A new interpreter (`python.exe`), pip, and various other files will be created in this folder. In order to work within the virtual environment, the environment's folder doesn't have to be your working directory in the terminal; in fact, the official venv documentation says that you "don't place any project code in the environment." Rather, you will "activate" the virtual environment, and as long as you don't deactivate it, the `python` command will run the environment's interpreter and only use the environment's packages. To activate an environment, you need to run a script which is automatically created in the `Scripts` (or `bin` on Linux) folder of that virtual environment's folder (see Figure 14). For Windows or Linux, see the commands listed here. Note that if using PowerShell, you may need to enable scripts to be run. Try using the command `Set-ExecutionPolicy -ExecutionPolicy Unrestricted` to do this, which will require running in administrator mode. To deactivate a virtual environment, simply use the command `deactivate`.

For Mac, you will need to use pip: `pip install virtualenv`. To create a virtual environment, use `virtualenv folder`, where "folder" is the environment name. To activate the environment, run the script in the virtual environment's `bin` folder using `source folder/bin/activate`. To deactivate, use `deactivate`.

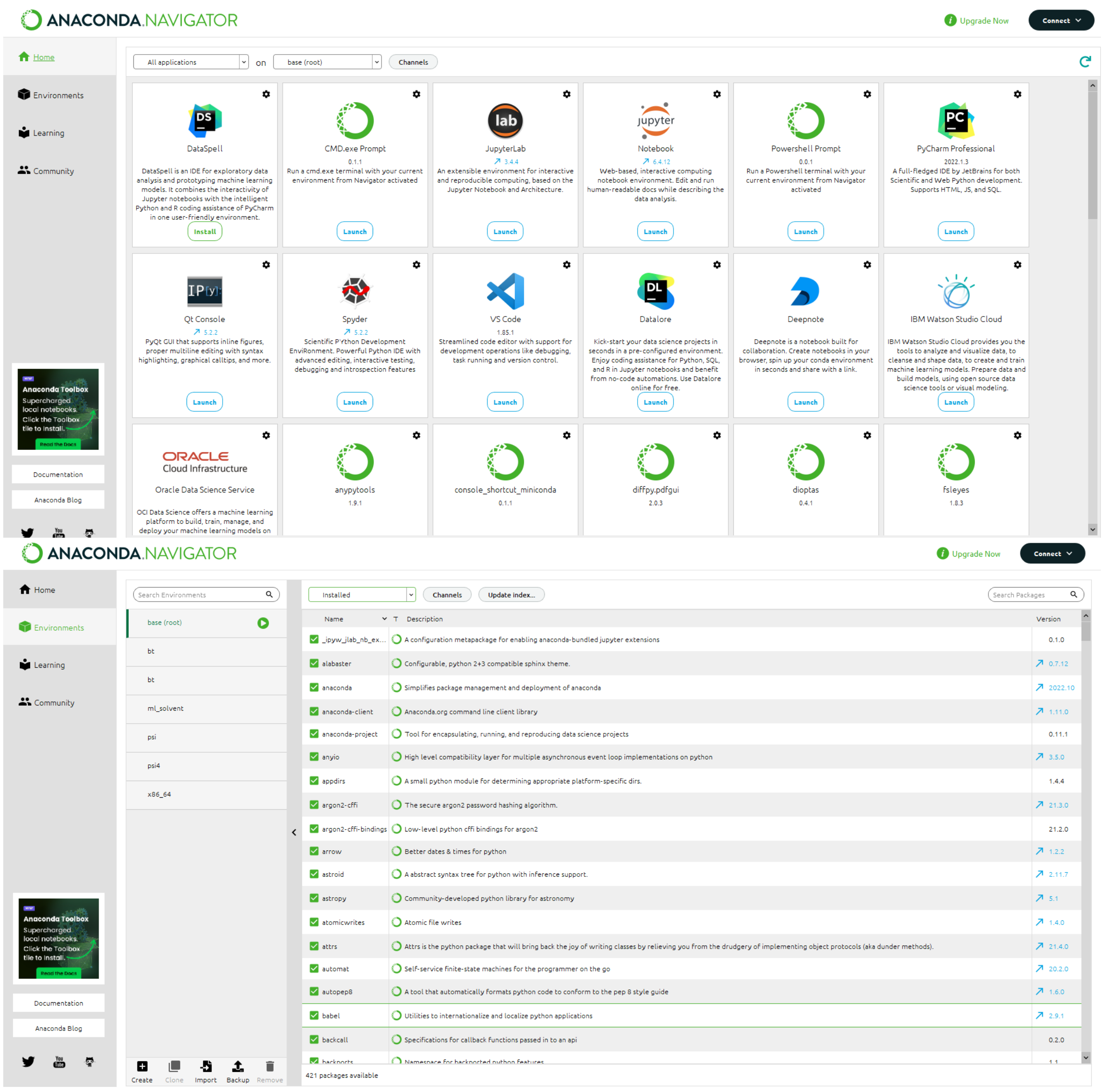
There's nothing special about deleting a virtual environment; just delete the folder like any other. For technical reasons which are explained in the link given earlier, if you want to move an environment folder from one folder to another, you should delete the folder and make a new one at the desired location instead of copying and pasting the folder. To get a list of all the packages installed in a virtual environment (which you'll need when you make the new one), use `pip freeze`. By using `pip freeze > requirements.txt`, this list will be saved to the file `requirements.txt`. You can use this file to reinstall all the packages in the new environment with `pip install -r requirements.txt` when you've activated the new environment.



```
PS C:\ComputationalChemistry> python -m venv virtual1
PS C:\ComputationalChemistry> python -m venv virtual2
PS C:\ComputationalChemistry> .\virtual1\Scripts\Activate.ps1
(virtual1) PS C:\ComputationalChemistry> pip install numpy
Collecting numpy
  Using cached numpy-1.26.3-cp39-cp39-win_amd64.whl (15.8 MB)
Installing collected packages: numpy
Successfully installed numpy-1.26.3
WARNING: You are using pip version 21.2.3; however, version 23.3.2 is available.
You should consider upgrading via the 'C:\ComputationalChemistry\virtual1\Script
p' command.
(virtual1) PS C:\ComputationalChemistry> python .\scriptRequiringNumpy.py
2.718281828459045
(virtual1) PS C:\ComputationalChemistry> deactivate
PS C:\ComputationalChemistry> .\virtual2\Scripts\Activate.ps1
(virtual2) PS C:\ComputationalChemistry> python .\scriptRequiringNumpy.py
Traceback (most recent call last):
  File "C:\ComputationalChemistry\scriptRequiringNumpy.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
(virtual2) PS C:\ComputationalChemistry> |
```

Figure 14: Using virtual environments in the PowerShell terminal. First, environments `virtual1` and `virtual2` are created. `virtual1` is then activated by running the appropriate script. The `(virtual1)` at the start of a line indicates that this environment is active. Next, pip is used to install the Numpy library in the `virtual1` environment. A Python script which uses Numpy is then run (note that the script itself is not in the `virtual1` environment), and the script runs successfully, printing the number *e*. The `deactivate` command is then used to deactivate `virtual1`, and `virtual2` is activated. Since `virtual2`, doesn't have Numpy installed, running the same script produces an error in this environment.

There is another optional program which we will now briefly mention: Anaconda. Anaconda is a Python distribution which provides some convenience over using Python without it. It is designed to manage your various packages and environments. Be sure to add it to the PATH environment variable, which should come up as an option during installation. In each environment, you can have a set of apps, including but not limited to a Python interpreter, Spyder, VS Code, as well as an interpreter for the "R" programming language, commonly used for statistics. And each environment will of course have its own packages. Note that to install packages with Anaconda, you will need to select the correct "channel" to install the package; usually you can look this up on the package's website. Anaconda provides a graphical interface (Anaconda Navigator) and a terminal (Anaconda Prompt), so you can use whichever you prefer to run programs within environments, install packages, create/delete environments, etc. See this link for some terminal commands. Anaconda is free, incidentally, so ensure that you don't purchase one of the business versions by mistake.



DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Install

CMD.exe Prompt

0.1.1

Run a cmd.exe terminal with your current environment from Navigator activated

Launch

JupyterLab

↗ 3.4.4

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch

Notebook

↗ 6.4.12

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch

Powershell Prompt

0.0.1

Run a Powershell terminal with your current environment from Navigator activated

Launch

PyCharm Professional

2022.1.3

A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

Launch

Qt Console

↗ 5.2.2

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launch

Spyder

↗ 5.2.2

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch

VS Code

1.85.1

Streamlined code editor with support for development operations like debugging, task running and version control.

Launch

Datalore

Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

Launch

Deepnote

Deepnote is a notebook built for collaboration. Create notebooks in your browser, spin up your conda environment in seconds and share with a link.

Launch

IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Launch

ORACLE Cloud Infrastructure

Oracle Data Science Service

OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on

anypytools

1.9.1

console_shortcut_miniconda

0.1.1

diffpy.pdfgui

2.0.3

dioptas

0.4.1

fsleyes

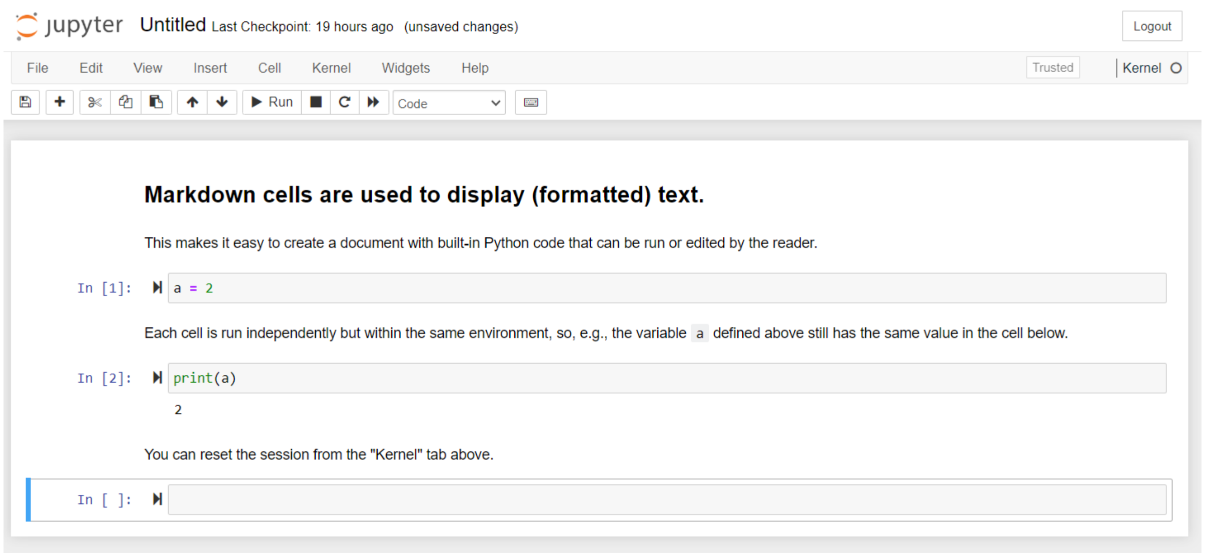
1.8.3

	Name	T	Description	Version
✓	_ipyw_jlab_nb_ex...		A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓	alabaster		Configurable, python 2+3 compatible sphinx theme.	↗ 0.7.12
✓	anaconda		Simplifies package management and deployment of anaconda	↗ 2022.10
✓	anaconda-client		Anaconda.org command line client library	↗ 1.11.0
✓	anaconda-project		Tool for encapsulating, running, and reproducing data science projects	0.11.1
✓	anyio		High level compatibility layer for multiple asynchronous event loop implementations on python	↗ 3.5.0
✓	appdirs		A small python module for determining appropriate platform-specific dirs.	1.4.4
✓	argon2-cffi		The secure argon2 password hashing algorithm.	↗ 21.3.0
✓	argon2-cffi-bindings		Low-level python cffi bindings for argon2	21.2.0
✓	arrow		Better dates & times for python	↗ 1.2.2
✓	astroid		A abstract syntax tree for python with inference support.	↗ 2.11.7
✓	astropy		Community-developed python library for astronomy	↗ 5.1
✓	atomicwrites		Atomic file writes	↗ 1.4.0
✓	attrs		Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	↗ 21.4.0
✓	automat		Self-service finite-state machines for the programmer on the go	↗ 20.2.0
✓	autopep8		A tool that automatically formats python code to conform to the pep 8 style guide	↗ 1.6.0
✓	babel		Utilities to internationalize and localize python applications	↗ 2.9.1
✓	backcall		Specifications for callback functions passed in to an api	0.2.0
✓	backports		Namespace for backported python features	1.1

Figure 15: Anaconda Navigator. **TOP**: Viewing applications installed in an environment. **BOTTOM**: Viewing packages installed in an environment. From this screen, packages can be installed and removed, and environments can be created and deleted.

Jupyter Notebook

Along with `.py` files, there's another important type of file used for Python code: `.ipynb` files. These files are used by the Jupyter Notebook program, among others, and are useful for educational purposes. These files are divided into "cells," each of which can either contain Python code or regular text. Each cell containing code is run one at a time when the user decides. The cells containing text, "Markdown cells," are simply there for the user to read; different font sizes can be used, along with bold, italics, URL links, etc., so reading ordinary (non-code) text is much easier than if it were written in a `.py` file. Together, code cells and Markdown cells can be used for teaching something in Python or for telling some kind of narrative interspersed with Python code. Of course, you don't *have* to use it like this; there's nothing stopping you from using it to write and experiment with code, although if you've written a program for others to use, it wouldn't typically be given in this format.



Markdown cells are used to display (formatted) text.

This makes it easy to create a document with built-in Python code that can be run or edited by the reader.

```
In [1]: a = 2
```

Each cell is run independently but within the same environment, so, e.g., the variable `a` defined above still has the same value in the cell below.

```
In [2]: print(a)

        2
```

You can reset the session from the "Kernel" tab above.

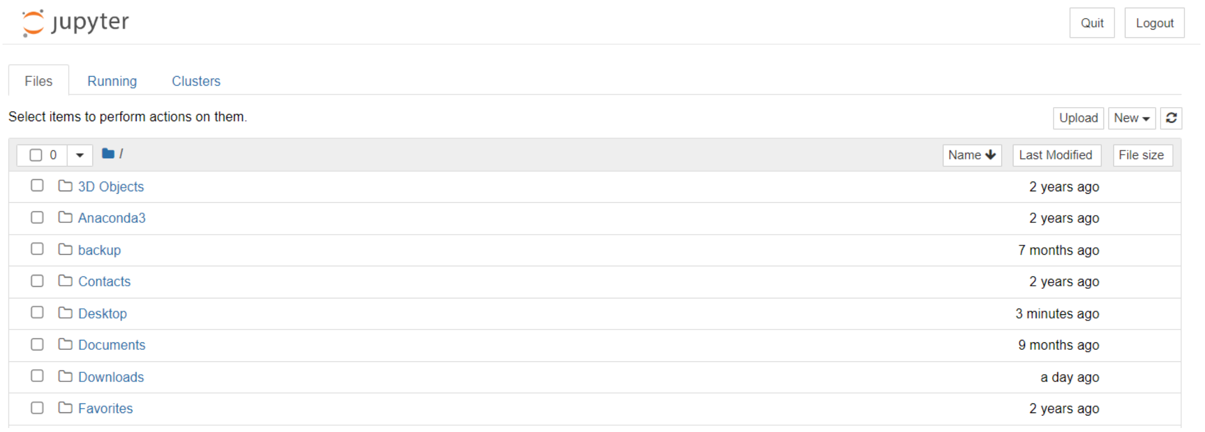
```
In [ ]:
```

Figure 16: A `.ipynb` file run in Jupyter Notebook.

To select/highlight a code cell, click on the cell outside of the space for writing code. Clicking inside this space will let you write/edit code in the cell. Ctrl+Enter while writing inside or highlighting a code cell will run that cell's code, as will clicking on the button towards the left of the cell. Clicking anywhere on a Markdown cell will highlight the cell. Double-clicking on it will allow you to edit the text. Ctrl+Enter will then render the text, i.e. convert the text you've written in the cell to proper-looking text stylized (e.g., bold, italics) as you've indicated.

You can use the up/down arrow keys to change which cell is highlighted. When any type of cell is highlighted, but not when writing inside a cell, type 'a' to insert a new cell above the currently highlighted cell; type 'b' to insert a new cell below. New cells will be code cells by default. Press 'd' *twice* to delete a cell. To convert a code cell to a Markdown cell, type 'm' while the cell is highlighted. To convert a Markdown cell to code, use 'y' instead. The text written in Markdown cells uses its own sort of code ("markup") to add bold, italics, bullet points, links, etc. You can view a cheatsheet for them here.

To install Jupyter Notebook, use `pip install notebook`. You can of course also install using Anaconda instead of pip. To launch, run the command `jupyter notebook`. The program itself should launch in your Internet browser.



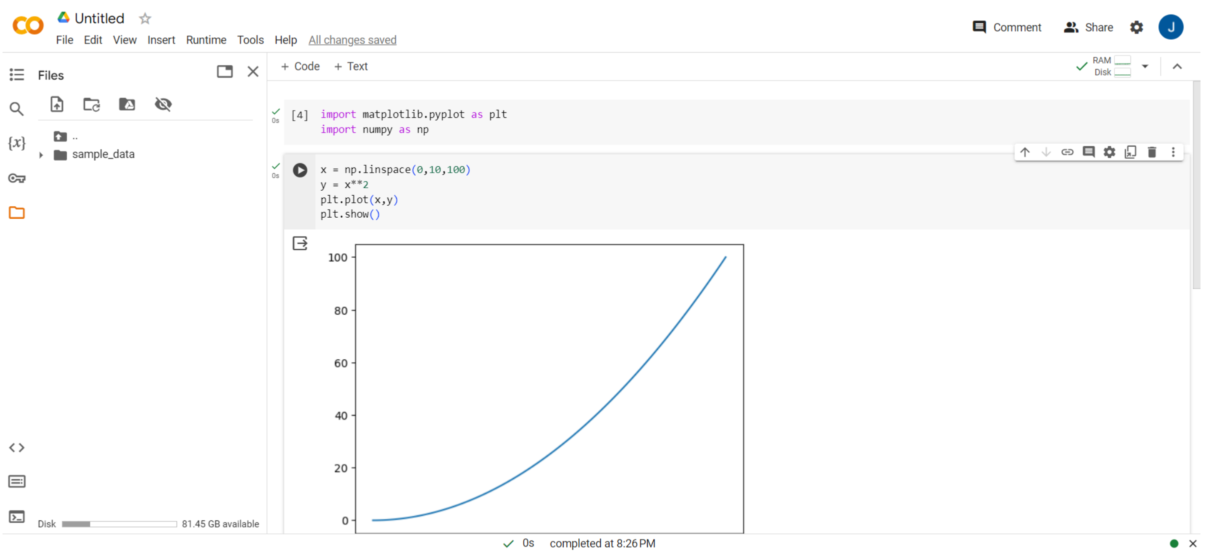
Select items to perform actions on them.

Name	Last Modified
3D Objects	2 years ago
Anaconda3	2 years ago
backup	7 months ago
Contacts	2 years ago
Desktop	3 minutes ago
Documents	9 months ago
Downloads	a day ago
Favorites	2 years ago

Figure 17: Jupyter Notebook running in a Chrome browser. From here, navigate through folders to find the desired `.ipynb` file or create a new one in the present folder by clicking the "New" drop-down menu near the top right.

Online editors and interpreters

In addition to all of this, there are some Python editors and interpreters which can be used online. A common resource is Google Colab, which functions similarly to Jupyter Notebook. Again, one can use `.ipynb` files to write and experiment with code, so Colab can still be used even if you aren't making a pedagogical document. At the time of writing, I unfortunately have a tiny little Chromebook as my laptop, so I frequently use Colab, which I can open as a tab in my browser rather than having to open other programs. Other files (containing data, etc.) can be uploaded and downloaded to/from the Colab environment for you to read from/write to with your Python code. Colab also has code linting and other features similar to an IDE like Spyder or VSCode.



```
[4] import matplotlib.pyplot as plt
    import numpy as np
```

```
x = np.linspace(0,10,100)
y = x**2
plt.plot(x,y)
plt.show()
```

Figure 18: Google Colab running in a Chrome browser.

You need a Google account to use Colab, which you might already have for Gmail, YouTube, Google Drive, etc. Colab is free, but of course the free plan is limited, since all the computational work is done on a machine somewhere else in the world. (If you're curious about where this machine is running, type `!curl ipinfo.io` into a cell and run it.) The free plan should be enough for most purposes. Note that if you go 90 minutes without interacting with the site, your session will end and executing code will be halted. This can be a problem if you're doing machine learning—machine learning models can take hours to train—and you want to run the training overnight. And, of course, it's Google, so you certainly shouldn't expect your privacy to be respected.

There are other options, as well, which you can feel free to search online. PythonAnywhere and Replit are two options, both of which you can access with free accounts. Usually, of course, these will be limited; you really ought to know how to install and work with a code editor and interpreter on your own machine.