Python

Blaise Thompson

April 6, 2020

Contents

Part of the training materials prepared by the Chemistry Shops at UW–Madison.
This document was prepared using LaTeX.
Source code and all associated files can be found at git.chem/shop/training/python.
If you find any mistakes or feel that any information is missing, please open an issue.

1 introduction

Python is a wonderfully expressive and elegant language which is becoming more popular each year. Python is multi-paradigm: the language supports functional, object oriented, event-based, and procedural programming. Python is used in a wide variety of applications: data processing, web development, hardware interface, and many other applications. This wide amount of flexibility makes Python one of the most popular programming languages today.

The core Python language is very minimal. On top of this core, Python offers a package system that allows developers to install additional features that can be "imported" into their scripts. The Python Package Index contains over 200,000 such packages that can be installed for free.

Python is not just Python! In many cases, Python is used as a "glue language". Consider NumPy: the fundamental package for scientific computing with Python. The NumPy source code is more than 50% C. For packages like NumPy, the Python code is simply an "interface" to the more peformant C implementation. Python for graphics typically work this way as well. It is also common to provide a Python "scripting interface" to complex applications written in other languages: consider for example the Python plugin interface to KiCad.

All this flexibility means that it isn't practical to just "learn Python". Even developers that have been using Python for many years are probably only familiar with a small piece of the ecosystem. It's best to approach Python with a "project oriented" mindset because otherwise you are likely to get lost in the multitude of options.

2 installation

Please use Python 3. Python 2 was sunset on 2020-01-01, and is no-longer receiving security patches. If you are using Python 2 due to some legacy code, upgrading is typically not challenging. The syntax differences between Python 2 and Python 3 are minimal.

Because Python often serves as a "glue language", it is sometimes difficult to install packages. To install NumPy from source, you need a C compiler, which typically isn't available on a Windows machine. Anaconda is a company that attempts to solve this package distribution problem by distributing pre-compiled packages for each operating system (for Python and other languages too). Their package manager is called conda. We recommend installing miniconda, which minimally installs conda on your machine. You can also install the "full" anaconda distribution, which is the conda package manager plus a bunch of pre-installed packages. Once conda is installed there are a few commands to know:

- → `conda config --add channels conda-forge` add conda-forge channel
- → `conda --help`
- → `conda install <NAME>`
- → `conda list` list installed packages
- → `conda upgrade --all` upgrade packages

3 built-in behavior

The repository at https://git.chem.wisc.edu/shop/training/python contains scripts that we will use for this lesson. Refer to the "basics" folder. This lesson covers built-in Python behavior. These are features of the language that require no additional packages.

The simplest way to interact with Python is via the REPL (Read Eval Print Loop). Simply type `python` into your terminal to enter the REPL. Also `python -i <SCRIPT>.py`.

`hello_world.py`

`math.py`

`bool.py`

`strings.py`

`files.py`

`iterables.py`

`functions.py` Note the use of docstrings. We recommend NumPy style docstrings.

4 scientific python

The repository at https://git.chem.wisc.edu/shop/training/python contains scripts that we will use for this lesson. Refer to the "scipy" folder. This lesson introduces the core packages of the scientific python ecosystem. These packages extend Python with powerful data processing and simulation functionality. We will be using NumPy, Matplotlib, and SciPy. You can install these with conda if you don't already have them:

```
$ conda install numpy matplotlib scipy
```

Scripts to go through:

`sine.py`

`genfromtxt.py`

`broadcasting.py`

`leastsq.py`

There are several other "favorite" Python packages that deserve mention.
For working with data:

→ h5py - store and process huge numerical arrays
→ pandas - interact with tabular data
→ xarray - metadata for multidimensional arrays

For interacting with instrumental hardware:

→ pySerial - send and receive serial commands
→ PyDAQmx - interact with DAQs from National Instruments
→ yaq - a modular and extensible instrument control framework

Other:

→ scikit-learn - machine learning
→ nmrglue - work with NMR data
→ WrightTools - work with multidimensional spectroscopy data

5 packaging

"Packaging" is the process of taking Python code and making it into a Python Package. This will allow other people to "install" your code and access it through importing. Moving from the scripting stage to developing a package can be a bit of a leap, since packaging requires learning some slightly more advanced concepts like modules and setuptools.

At some point Blaise would like to explain packaging... for now you can copy the structure of WrightTools if you want to see a completed package.

6 resources

development environments

Python files are plain text, so you may write and refine them using any text editor. However there are many integrated development environments (IDEs) that you may find helpful.

→ PyCharm
→ VS Code
→ Spyder
→ Notepad++ (Windows only)
→ Atom

Jupyter Notebooks also deserve special recognition as an excellent interface to Python in certain circumstances. In general, notebooks are good environments for scripting and processing data. Notebooks do not scale well to application or package development.

learning more

Learning Scientific Programming with Python

Matplotlib tutorials