Python

Guide for students of Medieval Words, Modern Methods

P. S. Langeslag

revision of August 28, 2025

Requirements

This document assumes you have installed Anaconda (or else another Python distribution plus Jupyter), as well as VS Code with the extensions entitled Python, Python Environments, and Jupyter (all three by Microsoft), as described in the Software guide.

Overview

Python is sufficiently documented elsewhere; the present document serves only to get you started, and to suggest further reading on your journey to Python mastery.

Principles

Python is an **interpreted programming language**, also known as a **scripting language**: we don't compile our code into binaries, but run it straight from a specialized interpreter. This makes the code inefficient to run, but it has the advantage that we can continually tweak the code as we experiment with it.

Many of the more accessible Python textbooks and tutorials still expect you to enter your code directly into an interpreter, which is a little arcane and rather frustrating whenever you make an error, or else to save your code as `.py` script files and run those from the command line. In education, these options have long been overtaken by the Jupyter notebook, which allows you to break your scripts up into individual code cells that you can run and bugfix on the fly, independently of the surrounding code.

Where most programming languages rely heavily on various kinds of bracketing to indicate hierarchy, Python replaces some of this with tab indentations. Thus whereas PHP might have

```
foreach($x as $y)
    {
        $z = explode(", ", $y);
        $index[$y] = $z;
    }
```

Python omits the curly braces:

```
for y in x:
    z = y.split(', ')
    index[y] = z
```

This means that whereas tabbing is just a matter of convention in PHP, it is of crucial importance in Python that you engage in exact tabbing, in multiples of four spaces. Also note that Python lacks the semicolons signalling the conclusion of a statement in PHP, but it has colons to announce that a subordinate hierarchy follows.

Libraries

Python relies heavily on user-contributed libraries. A library, or package, is just a collection of Python scripts meeting certain standards of quality and utility to have been admitted into the pool of packages made available by the package managers (`pip` and `conda`). Some packages are considered so crucial that they ship with Python itself — they are part of the Python Standard Library; for instance, `math` for mathematical operations, `os` for disk access, and `json` for loading and storing data. But most libraries have to be installed first, and all have to be imported before they can be used. The Software guide describes the process of installing libraries; here we may just illustrate how to import libraries before using them. It is conventional to import all libraries at the top of your script or notebook. Take special note of the three different ways of importing libraries, and how to access them thereafter.

```
import os,glob,json
from collections import Counter
import pandas as pd

my_counter = Counter(some_dictionary)
df = pd.DataFrame(some_data)
with open('data.json') as json_file:
    json_data = json.load(json_file)
```

Functions

If this is your first programming experience, you'll probably find that instead of writing functions, you're just processing data as you need it. That's fine for now; just keep in mind that as you start to apply the same code repeatedly, it will become more efficient to write a function and apply it repeatedly rather than repeat longer sequences of code. Here's an example you'll want to use early on. The actual function definition is just the block starting with `def`:

```
string1 = "æghwilc þæra manna se þe ðam rihtlice onfehð æghwilc þæra onfehð \
    eces lifes wedd"
```

```
string2 = "he cwęð ne eom ic na to þam on middanearde cumen þæt ic wolde þæt \
    me menn þenodon ac þæt ic wolde mannum þenian"

substitutions = {
    'ð': 'þ',
    'ę': 'æ'
    }

def normalize(text):
    for k,v in substitutions.items():
        text = text.replace(k,v)
    return text

for input in [string1, string2]:
    print(normalize(input))
```

Learning Python

The present guide serves to give you some pointers, not to teach Python as such. Because you will be engaging in NLP (natural language processing), a good introduction to Python is the online textbook *Natural Language Processing with Python*, built atop the NLTK library. Beyond that, Real Python has an amazing collection of guides on various tasks and aspects of Python, many of which are freely available.

Jupyter Notebook Modes and Commands

Like the Vim editor popular among programmers, the Jupyter Notebook interface has a command mode and an edit mode. When you open a notebook, it is in command mode and will respond to keyboard commands like the following:

`Enter`	Enter edit mode
`Shift+Enter`	Run cell code
`k` or arrow	Navigate up
`j` or arrow	Navigate down
`a`	Add a cell above
`b`	Add a cell below
`y`	Change cell mode to code
`m`	Change cell mode to Markdown
`dd`	Delete current cell

Once in edit mode, you have access to commands like the following:

`Esc`	Enter command mode

`Shift+Enter`	Run cell code and command new cell below

For more keyboard shortcuts, see e.g. here.

Markdown

Markdown is an efficient structured text format which it is well worth mastering, as it is the most convenient and powerful format in which to write nearly all the documents you need in your professional and personal life (except spreadsheets or complex tables). Fuller guides are all over the web, but the short version is that it relies on formatting cues like the following:

~~~
# Top-level headings receive one hash/pound sign

## Lower-level headings receive two

Paragraphs are ended by double line breaks.

Like so.

[Links](https://are.encoded.thus) or they may be encoded <https://thus.ly>

```python
Code demonstrations ("listings") are set off within triple backtick delimiters.
```
~~~

You can find more examples in the folder containing this guide, as I have shared the (Pandoc-flavoured) markdown files underlying these guides by way of example.

The relevance of Markdown to Jupyter Notebooks is that it is the format for note-taking cells. You can toggle the format of a selected cell from code to Markdown with the `y` and `m` keybindings (press `Esc` first if in edit mode), or using the dropdown menu. A Markdown cell serves to document or annotate your code, or to create logical sections within a Python script. Jupyter's markdown, which is the same as GitHub's markdown, is not as rich as Pandoc's markdown, and e.g. tables are subject to a more restricted syntax.