Python for Digital Humanities

Theena Kumaragurunathan

Fri Nov 18 12:16:00 AM +0530 2022

Contents

1 Lesson 1 INTRODUCTION TO PYTHON FOR DIGITAL HUMANITIES

1.1 Resources

This notebook is based on lessons found here: Python For Humanities

1.2 Audience

Though the course is geared specifically towards the needs of the Humanities students and researchers, the techniques and concepts discussed here have direct application and applicability to the work of writers, fiction and non-fiction. Like the social sciences, writers too grapple with large chunks of

text they'd encounter during research phases, writers too would be distilling the research into chunks that need to, thereafter, be distilled once more into their writing. The difference is academic rigor; the toolkit can be similar.

Python is that: a vast array of toolkits that can be used to automate many of the most mundane computational tasks that both serious academics and writers encounter on a daily basis.

1.3 Requirements

1. Python 3.8 or >
2. A text editor of your choice, but preferably PyCharm

1.4 Acknowledgements

This notebook owes a debt of gratitude to Python tutors all over the web, who have spent the last few decades evangelizing its usage outside the traditional domains of computer programming, arguing cogently that here was a language for everybody who uses computers to do any manner of computational work, whether that is writing a novel, a thesis, analyzing excel sheets, or reading PDFs while researching. In all these use cases, and more, Python finds itself to be a useful utility.

This notebook follows the coursework made available by Dr William Mattingly] on PythonHumanities.Com

1.5 Copyright

concepts. If you wish to fork this repo, please ensure that you make clear that the course material itself belongs to Dr William Mattingly.

I welcome any pull requests! The notebook was written in a hurry, to meet a specific work-related need on my end - all errors - whether in text or code, are mine. Please feel free to send a pull request if corrections are needed.

1.6 What is Python

Python is a general purpose **interpreted** programming lanuage.

It is general purpose because its usage is wide, finding utility across a diverse range of disciplines. Finance professionals use it to analyze Excel data, data scientists use it find insights in vast troves of statistical data, programmers themselves use it to create simple applications to solve specific every-day problems they come across in their day-to-day work. Those of us outside the world of code and financial data, will find Python just as useful: automating the downloading of public domain images, summarizing an academic paper, finding keywords in a research report using textual analysis. The writer/programmer/educator Al Sweigart's Automate the Boring Stuff With Python highlights the many ways the menial tasks that we perform on computers can be relegated to Python. In this way, I liken it to a Swiss Army Knife of programming languages.

Programming languages are of two main types:

- Compiled
- Interpretted

The distinction between is useful to know.

Since we are speaking of *languages*, the distinction between the two can be likened to reading a set of instructions - let's assume a recipe for a curry; the instructions are written in Sinhala. There are two ways for you to follow the instructions: *either look for a translation of the recipe* or *have a friend who reads the language to read the instructions line by line.*

In this analogy, the translated version of the recipe is a *Compiled Language*, while the friend offering you line-by-line translations, is the *Interpretted Language*

The advantages of one over the other is beyond the scope of this course.

1.7 Why Python

Why Python specifically then? Why not the C programming language? Why not JavaScript?

If we go back to the language analogy, we know from real life experience that some languages are easier to pick up than others. Python excels because its grammar and lexicon are easier to grasp for the non-computer programmer. Python's syntax, once internalized, has a logic that isn't unlike learning a human language - in fact, Python's code is especially known for being readable in a way, say, code written in *C* or *C++* or *Java* aren't.

As a result, Python users are diverse, encompassing people from the technical to the non-technical, from the sciences to the arts, and everyone in between.

There is of course nothing wrong with learning C or any other programming language if you are feeling motivated, but to employ C to do many of the tasks that non-programmers typically perform on computers, will be the equivalent of using *Excaliber* to slice a loaf of bread; it will get the job done, but you might damage your kitchen in the process. JavaScript, on the other hand, works in the intersection between HTML and CSS, and is best suited for extending the fucntionality of websites and web apps.

1.8 Installing Python

Python can be installed on Windows, MacOS and Linux. The steps vary. Please visit Python.org for detailed instructions for your operating system.

1.8.1 Text Editors

This is personal choice. For beginners, PyCharm Edu, in my view, offers the best experience. VSCode can be overwhelming to newcomers and tries to be the *code-anything-in-any-language* text editor for programmers. For our purposes, PyCharm would be the most focused for a beginner. My own choice is a combination of PyCharm and NeoVim, the latter because it is the text editor of choice for any text editing task.

1.9 Pythonic

Pythonic is the orthodox way to design and write code in Python. Think of this as a unviversal style guide on commonly accepted conventions and principles on Python code. This convention is followed by the Python community, both programmers and non-programmers, making the sharing of

Python code as frictionless as possible. This also enables quick diagnosing of the problem should you run into errors with your code.

For example when calling upon the **Python Library** *NumPy*, the code would be written as follows: `import NumPy as np`