01.1_Introduction

January 18, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang
- Class Location and Time: ENV 336, Mon & Wed 12:30 pm - 1:50 pm

2 Why, What, Who, When, How

2.1 Why learning coding?

[1]:
```
from IPython.display import YouTubeVideo
display(YouTubeVideo("YPE2dO5sII0", width="900",height="450"))
```

2.2 Why Learning Coding? Your story

Class activities: discuss with your neighbor for 3 mins about why you want to learn programming and share your story with the class.

2.3 Why learning coding? (1)

Perhaps…

- you are performing the same functions every day, month, or year, but new data keep coming (automating some tasks)
- your data are in textual form that can be better understood by using (cartographic) maps, but the data is simply messy and the data volume is huge (data wrangling)
- you want to write a simple game
- you simply feel coding is elegant
- job ads often specify coding experience is preferred

2.4 Why learning coding? (2)

- Many disciplines (including geography) are becoming increasingly quantitative and basic programming skills are one of the fundamental skills that will help you be a better scientis

- You can extend existing software by developing your own solutions when solutions do not exist or are inefficient
- Programming is fun and rewarding!

2.4.1 The scientific method...

...and how programming can make you a better scientist

1. Define a question
2. Gather information and resources
3. Form an explanatory hypothesis
4. Test the hypothesis by performing an experiment and collecting data in a reproducible manner
5. Analyze the data
6. Interpret the data and draw conclusions that serve as a starting point for new hypothesis
7. Publish results
8. Retest (frequently done by other scientists)

Learning to program can help us...

1. Define a question
2. **Gather information and resources**
3. Form an explanatory hypothesis
4. Test the hypothesis by **performing an experiment** and **collecting data** in a reproducible manner
5. **Analyze the data**
6. **Interpret the data and draw conclusions that serve as a starting point for new hypothesis**
7. Publish results
8. Retest (frequently done by other scientists)

Good programming practices can help us...

1. Define a question
2. Gather information and resources
3. Form an explanatory hypothesis
4. Test the hypothesis by performing an experiment and collecting data **in a reproducible manner**
5. Analyze the data
6. Interpret the data and draw conclusions that serve as a starting point for new hypothesis
7. Publish results
8. **Retest (frequently done by other scientists)**

2.4.2 Course goals

- Provide an introduction to Python
- Upon successful completion of this course, students will be able to:
  - Master the fundamentals of writing Python scripts.
  - Write Python functions to facilitate code reuse.
  - Make their code robust by handling errors and exceptions properly.
  - Develop python programs for data manipulation.
  - Create python programs for solving problems.

2.5 Why open source?

Guido van Rossum (Creator of Python):

> I see this as the essence of open source projects: The energy and creativity of many people with diverse goals together can work miracles!

Encourages innovation through collaboration!

Enhancing the Scientific Process!!!

2.6 What is a programming language?

- A computer language is what we use to 'talk' to a computer
  - Unfortunately, computers don't yet understand our native languages
- A programming language is like a code of instructions for the computer to follow
  - It is exact and unambiguous
  - Every structure has a precise form (syntax) and a precise meaning (semantics)
- Python is just one of many programming languages

2.7 What is Python?

- High-level
- Interpreted
- Object-oriented
- All-purpose
- Scalable
- Extensible
- Easy to learn, read and maintain
- Robust
- Rapid prototyping tool
- It's Fun!

2.7.1 Python VS Other languages

- C, Fortran, C++
- Java, Perl, Ruby, Scheme, VB
- Matlab, Gauss, R, Mathmatica, Maple

Advantages of Python

- Free
- *super glue*
  - no need to replace legacy code
  - useful for heterogenous projects/data/languages
- Shorter, easier to develope
- High readability

Read More on the comparisons

2.7.2 Java VS Python

```java
public class PythonandJava {
    public static void main(String[] args)
    {
        System.out.println("Python and Java!");
    }
}
```

Output:

```
Python and Java!
```

[2]:
```
print("our first class!")
```

```
our first class!
```

[3]:
```
print("Python and Java!")
```

```
Python and Java!
```

2.7.3 Testimonials

Guido van Rossum http://www.artima.com/intv/speed.html

> A 20,000-line Python program would probably be a 100,000-line Java or C++ program. It might be a 200,000-line C program, because C offers you even less structure. Looking for a bug or making a systematic change is much more work in a 100,000-line program than in a 20,000-line program. For smaller scales, it works in the same way. A 500-line program feels much different than a 10,000-line program.

Bruce Eckel http://www.artima.com/intv/tippingP.html

> One of my first real productive experiences with Python, beyond just playing around with the language, involved image processing. I wanted to resize some GIF files. Given my experience with other languages, I figured this task might take me half a day if I were lucky. Even if there were an existing image processing library in Python, I figured the library would be complicated and take significant time to understand. I discovered a Python library that did graphics manipulation, and to my surprise, resizing GIFs was as simple as you can imagine it could be. You create an object, call reformat, pass in some arguments, and you're done. In C++, and even in Java, the ease of understanding a library is not really part of the culture. In Python it really is. Instead of taking a half a day, which was my best hope, after a half an hour, I couldn't think of any more features to add to my program. And I was just stunned. I thought, oh, that's what people mean when they talk about Python's incredible productivity.

2.8 Personal Experiences with Python

- Started my Python Journey in 2014
- GeoComputation
  - Monte Carlo papers
    - Spatial econometrics
    - Local spatial modeling
  - Exploration of new ideas -> papers
- Full-blown applications
  - PySAL: Python Spatial Analysis Library
    - PySAL Entry in UCGIS: PySAL and Spatial Statistics Libraries
  - geosnap: geospatial neighborhood analysis package

2.9 Who and When

2.9.1 Origins

- Guido van Rossum 1989
- Origins in ABC
- Public distribution 1991
- Named after the BBC show "Monty Python's Flying Circus" and has nothing to do with reptiles
- "Computer Programming for Everybody" proposal 1999
  - An easy and intuitive language just as powerful as major competitors
  - Open source, so anyone can contribute to its development
  - Code that is as understandable as plain English
  - Suitability for everyday tasks, allowing for short development times

2.9.2 How to Work with Python

- interactively
- running scripts

[4]:
```
13+2
```

[4]:
```
15
```

2.9.3 Using the interpreter: interactivity

- Start from a shell
  - Mac: terminal
  - Windows: powershell
- Integrated Development Environment (IDEs)
  - Pycharm: Python Console
  - Visual Studio Code: Python Console
  - ArcGIS Pro: Python Console
- Jupyter Notebook
  - Code cell

[5]:
```
14+2
```

[5]:
```
16
```

[6]:
```
10 * 100 / 20
```

[6]:
```
50.0
```

2.9.4 Running Python Scripts (1)

An alternative to using the interpreter is to put a collection of Python statements inside a text file, and then run this file. For example, create a new file from the Jupyter directory called `hello.py` and enter the following into this file

```
print('Hello World!')
```

2.9.5 Running Python Scripts (2)

- Shell:

```
python hello.py
```

- Python interpreter:

```
>>> exec(open("hello.py").read())
```

2.9.6 Running Python Scripts (3)

- Jupyter Notebook:

```
exec(open("hello.py").read())
```

[8]:
```
exec(open("hello.py").read())
```

```
Hello World!
```

2.9.7 Comparison between the two modes

- Scripts are useful when you want to permanently save some code with an eye for reuse later.
- You can use the interpreter to build up and test pieces of code until you get them working to your liking, at which point you can save them to the text file/script.
- Complementary: using both an interpreter and scripts together is a common use pattern for scientific programming in Python.

2.10 Feelings about python programming: classroom Activities

- Each student uses **two minutes** to write down three words describing his/her feelings about python programming in a shared doc
  - each student occupy one line
  - words are seperated by comma
  - an example: motivated, excited, worried

- We will use python programming to quickly generate a wordcloud about these feelings.

[12]:
```
!pip install wordcloud
import os

from os import path
from wordcloud import WordCloud

# get data directory (using getcwd() is needed to support running example in␣
 ↪generated IPython notebook)
d = path.dirname(__file__) if "__file__" in locals() else os.getcwd()

# Read the whole text.
text = open(path.join(d, 'data/feelings.txt')).read()

# Generate a word cloud image
wordcloud = WordCloud().generate(text)

# Display the generated image:
# the matplotlib way:
import matplotlib.pyplot as plt
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")

# lower max_font_size
# wordcloud = WordCloud(max_font_size=40).generate(text)
# plt.figure()
# plt.imshow(wordcloud, interpolation="bilinear")
# plt.axis("off")
# plt.show()
```

[12]:
```
(-0.5, 399.5, 199.5, -0.5)
```

3 About the course

- Course content
  - Tools
  - Python Basics
  - External Libraries
- Objectives
  - Master the fundamentals of writing Python scripts.
  - Write Python functions to facilitate code reuse.
  - Make their code robust by handling errors and exceptions properly.
  - Develop python programs for data manipulation.
  - Create python programs for solving research problems.

4 Evaluations

- Classroom participation
  - "Random" in-class quizzes (6)
  - a lab on 02/13 (4)
- Programming homework (10 * 5 = 50)
  - one week after the homework is released
  - submit on Jupyter Hub (introduce in a few weeks)
  - due by 10pm on the specified day
  - late work not accepted
- Exams: multiple-choice and open-ended questions
  - Mid-term Exam (15)
    - scheduled on March 8
  - Final exam (25)
    - scheduled on May 8

4.1 Class policies

- Academic dishonesty not accepted
- Accomodations (case by case)
- Use office hours
  - Dr. Wei Kang (Mon & Wed 2:00 – 3:00 pm)
  - TA: Prashant Thapaliya (Tue & Thur 11:00 AM – 12:30 PM)

4.2 Questions about syllabus?

Take several minutes to read the syllabus

5 Next Class (01/23)

Topics: Installation, Jupyter Notebook, Markdown

Reading: Jupyter Notebook Documentation

You are **strongly encouraged** to bring your own laptop to the class and install the software we will be using on your own laptop or desktop.

- You can more fully explore the capabilities of these packages as you will no longer be tethered to the lab.
- Because many open source projects have update schedules that are more frequent than proprietary packages, we may want to install a new version of a package in the middle of the course. As I do not have install rights to the lab computers, this means we would be prohibited from doing so.

[ ]: