Intro to Python for Global Affairs
Jackson Institute for Global Affairs, Spring 2020
Syllabus[1]

Instructor: William Casey King, PhD
Email: Casey.king@yale.edu
Class Time: Thursday, 7:00-8:50 pm
Location: Virtual

Teaching Assistant: Flynn Chen
Email: Flynn.chen@yale.edu

Office Hours: TA: TBD

Prerequisites: There are no prior courses required.

Class Size: 15 person limit

Course Description:

In the second decade of the twenty-first century, 'big data' analytics and techniques have fundamentally transformed policy decisions both in the United States and throughout the globe. NGO's, NPO's, political campaigns, think tanks and government agencies more and more recruit policy analysts with the necessary skills to embrace novel, data-driven approaches to policy creation and evaluation. This course is designed to help students meet this growing demand. It is an introductory course in python programming and data analysis for policy students with **no prior coding experience**. Unlike massive introductory classes, this course is deliberately small; designed to provide the necessary support for humanists to make a smooth and nurturing transition to "tech humanists." Ultimately, students should be comfortable using what they've learned in further Yale courses in programming and statistics, or in research and policy after leaving Yale. They should know enough to productively collaborate on projects with engineers, to understand the potential of such work, to have sufficient background to expand on their skills with more advanced classes, and to perform rudimentary data analyses and make policy recommendations based on these analyses.

Learning Objectives:
After completing this course, students will ...

[1] This syllabus is based, in part, on the required intro course taught in the Cal Berkeley Data Science Master's Program, developed by Paul Laskowsi et al. The instructor reserves the right to modify the course as necessary.

- Be able to navigate a file system, manipulate files, and execute programs using a command line interface
- Understand how to manage different versions of a project using Git and how to collaborate with others using Github
- Be fluent in Python syntax and familiar with foundational Python object types
- Be able to design, reason about, and implement algorithms for solving computational problems
- Be able to test and effectively debug programs
- Know how to use Python to extract data from different types of files and other sources
- Know how to read, manipulate, describe, and visualize data using the NumPy and Pandas packages
- Be able to generate an exploratory analysis of a data set using Python
- Be prepared for further programming challenges in more advanced data science courses

Instruction:

Student Commitment:

Learning to code well is easy, but requires a consistent application of effort, rather than tremendous effort at the last minute. Like fluency in any spoken language, which is best learned with daily practice, Python or any coding language is very much the same. In order for you to become Python competent, it will require considerable effort, a minimum of 8 hours per week outside of class and as much as 20 hours a week when projects are due. If you cannot allocate this amount of time, you should postpone your Python learning until you can make the necessary commitment. Obviously, the more time you are able to commit to the practice, the better your skills will be at the finish line.

Class meetings:

We meet once a week. In our meetings, I will present material via lecture. Some lectures will include in class exercises.

Homework:

There are a number of weekly activities, projects, and in-class breakouts that require a computer, Python3, and Jupyter. Homework (weekly problem set) for that unit is assigned after class and due the following week.

Grading:

- 10 Weekly Assignments - 50% (5% each)
- 1 Final Project - 20%
- Midterm Exam (take home, open stack) - 20%
- Participation. and Attendance- 10%

Required Textbook and other Readings:

- Lubanovic, B. (2014). *Introducing Python: Modern computing in simple packages*. Sebastopol, CA: O'Reilly Media.
- McKinney, W (2012). *Python for Data Analysis*. Sebastopol, CA: O'Reilly Media

Online Tools:

- Class Github: https://github.com/Jacksonpython/Spring_2021
- *Install and configure Git & Bash.*
- Anaconda with Jupyter Notebooks & Python3: https://anaconda.org/anaconda/python
- Slack:
  Use Slack to informally communicate with your classmates, ask questions, share problems and your successes (but don't share code).
- Stack Overflow: Open source coders cannot memorize everything. When we are stuck, the first thing we do is look to **stack overflow**: https://stackoverflow.com/

Course Schedule:

Week1: Introduction to Programming, Command Line, Source Code ○ Readings:
Lubanovic, B. (2014). *Introducing Python: Modern computing in simple packages.* Sebastopol, CA: O'Reilly Media. Chapter 1

Content

1. Introduction
2. Introducing Python
3. Programming Language Characteristics
4. The World of Programming Languages 5. Installations:
   a. Bash and Git and a Text Editor
   b. Anaconda
6. Anaconda
7. The Command Line
8. Exercise: Introduction to the Command Line
9. Navigating a File System
10. Source Control
11. Git and Distributed Source Control
12. Git Vocabulary
13. Starting Out with Git
14. The Course Workflow
15. Resolving Merge Conflicts
16. Your First Git Assignment

Week 2: Starting Out with Python ○ Readings: Lubanovic - chapter 2 and beginning of chapter 4, up to but not including the section labeled "cancel with break" [pp. 69-75].
○ **Homework: Unit 1: Due 11:59pm Wednesday**

Content:

1. Introduction
2. Installations: The Anaconda Distribution
3. Starting Python
4. Objects
5. Objects Drill
6. Objects and Types
7. Objects and Types Drill
8. Representing Numbers
9. Numbers
10. Numbers Drill
11. Variables, Part 1
12. Variables, Part 2
13. Variables Drill
14. Strings
15. First Module
16. Control (if, elif, else statements)
17. While Loops
18. Control Drills:

Week 3: Sequence Types and Dictionaries ○ **Reading**:
Lubanovic - Chapter 3 ○ **Homework: Unit 2: Due 11:59 Wednesday**

Content:

1. Introduction
2. The Jupyter Notebook
3. Sequences
4. Sequences Drill
5. List
6. Lists Drill
7. Lists and Mutability
8. Mutability, Part 1
9. Mutability Example
10. Mutability Example Drill
11. Mutability, Part 2
12. Mutability, Part 2 Drill
13. Tuples
14. Tuples Drill
15. Ranges
16. Ranges Drill
17. Dictionaries
18. Dictionaries Drill

Week 4: More About Control ○ **Reading:** Lubanovic - chapter 4, start with "cancel with break" (p. 75) through "comprehensions" (p. 85) ○ **Homework:**
Unit 3: Due 11:59 Wednesday

Content:

1 Introduction
2. For Loops
3. For Loops Drill
4. Fancy Loops Exists
5. Fancy Loops Exits Drill
6. Comprehensions
7. Comprehensions Drill

Week 5: Functions ○ **Reading: Lubanovic - chapter 4, start with Functions, p. 85.**
○ **Homework: Unit 4: Due 11:59 Wednesday**

Content:

1. Introduction
2. Functions
3. Calling Functions
4. Nesting Functions
5. Nesting Functions Drill
6. Functions and the Call Stack
7. The Stack Trace
8. Advantages of Functions
9. Namespaces
10. Namespaces in Detail
11. Accessing Global Variables
12. Recursion Basics
13. Traversing Nested Dictionaries with Recursion
14. Using Parameters
15. Functions are Objects
16. Exceptions
17. Exceptions Drill

Week 6: Classes ○ **Reading:** Lubanovic - chapter 6 through "Define a Class with class" and next "In self

Defense" through "Method Types" ○

Homework: Unit 5: Due 11:59 Wednesday

Content:

1. Introduction
2. Classes and Attributes
3. Using the Class Definition
4. Binding Methods
5. Binding Methods Drill
6. Initializing a Class
7. Initializing a Class Drill

Week 7: Files and importing packages ○ **Reading:** Lubanovic -chapter 14 (stop at "write a binary file with write()" skip to "pathnames". Stop at "pathnames") ○

Homework: Unit 6: Due 11:59 Wednesday

Content:

1. Reading a file
2. Writing to a file
3. Modules and Packages
4. Modules and Packages Drill
5. Modules and Import Statement
6. Modules and Import Statement Drill
7. Packages Drill
8. Python Standard Library

Week 8: All the President's Speeches: Text As Data Scraping and NLTK

Midterm Exam (Take home, Due Wed eve before class,11:59pm)

1. Introduction to "Scraping"
2. NLTK Package in Python
3. Scraping Data from the "Wild"
4. Introduction to BeautifulSoup
5. Introduction to Requests
6. Introduction to Selenium

Week 09:: Data analysis with Pandas ○ **Reading: No reading for this week**

○ **Homework Unit 7**

Content:

1. Introduction
2. Introducing Pandas
3. Pandas. Series
4. Pandas.Series Drill
5. Pandas.DataFrame
6. Pandas.DataFrame Drill
7. Bracket Indexing in Pandas
8. Loc and iloc
9. Applying Functions to DataFrames
10. Applying Functions to DataFrames Drill 11. Sorting
12. Example: Marathon Training

Week 10: April 9, 2020: Plotting and Visualization ○
Reading: No Reading for this week
○ **Homework: Homework Unit 8**

Content:

1. Introduction
2. Pythons's Plotting Libraries
3. Figures and Axes
4. Colors, Lines, Legends
5. Titles, Ticks and Text
6. Seaborn
7. Seaborn Drill
8. Guidelines for Visualization
9. How to Lie with Plots
10. Information Density and Data-Ink

Week 11: Pandas Aggregation and Group Operation ○ **Reading:** McKinney, W. (2012). Python for data analysis: Data wrangling with Pandas, NumPy, and iPython. O'Reilly Media, Inc. Chapter 10 ○ **Homework: Homework units 9, 10** Due 11:59 PM (this is campaign contr)

Content:

1. Introduction
2. Series.groupby
3. Series.grouby Drill
4. DataFrame.groupby
5. DataFrame.grouby Drill
6. Data Aggregation

7. Data Aggregation Drill
8. General Split-Apply-Combine Drill
9. Pivot Tables and Crosstabs
10. Example: Airline Flight Data

Week 12: A: Final Class Select presentations of final projects

Weekly Assignments:
The weekly assignments are designed to reinforce and extend the programming concepts covered in the lecture. A typical assignment consists of several programming exercises of varying difficulty. While some exercises can be completed in a single line of code, others may require students to combine commands in innovative ways, to design their own algorithms, and to navigate common sources of documentation. Students may consult with each other about the assignment but must write their own code and list their collaborators in their submissions. The expectation is that these assignments will take around 10-20 hours to complete each week. Depending on your experience with coding, this time estimate might be more or less. The weekly assignments are due at 11:59 PM the Wednesday before class.

Projects:
There is one large coding project. It is a group project that comes at the end of the course and involves the analysis of an actual data set using Python's system of data analysis packages. Further details about the projects will be given during the school term.

Exams:
The midterm and final exams are cumulative. Unlike the weekly assignments, students must do all of their work independently. Both exams include multiple-choice and short-answer questions, as well as short programming tasks, designed to test each student's grasp of important programming concepts. Further details about the exams will be given during the school term.

Participation:
Students are expected to participate in class activities, to contribute to discussions held in live section and on other platforms, to behave professionally towards classmates, and to help maintain a supportive atmosphere for education. Participation scores will be assigned based on these criteria.

Academic Integrity:

Avoiding Plagiarism:
Plagiarism is a serious academic offense, and students must take care not to copy code written by others. Beginning students sometimes have trouble identifying exactly when plagiarism takes place. Remember that it is generally fine to search for examples of code (e.g., on forums such as stackexchange). This is a normal part of programming and can help you learn. However, it is

important that you understand the code you find and use what you learn to write your own statements. It is okay if a single line of code happens to match an example found on the internet, but you should not copy multiple lines at once. If in doubt, simply document the place you found your example code and ask your instructor for further guidance.

Late/Extension Policy: We recognize that sometimes things happen in life outside the course, To help with these situations, we will be given 8 "late days" to use throughout the term as you see fit. Each late day gives you a 24 hour (or any part thereof) extension to any assignment in the course except the project presentations or final exam. Once you run out of late days, each 24 hour period (or any part thereof) results in a 10 percentage point deduction on the grade for that assignment. You can use a maximum of 2 late days on any single assignment. We will not be accepting any submissions more than 48 hours past the original due-date, even if you have late days. If you have **very** extenuating circumstances (for example, an emergency medical situation), please reach out to your section instructor. If you know an event is coming up that will make doing the assignment difficult, talk to your section instructor to maybe get the assignment slightly early. Plan your time accordingly!

Tutoring: If you are finding it difficult to get the programming concepts in the course please reach out to one of the TAs to setup a 1 on 1 (or 1 on small group) tutoring session.

Assignment Help:

If you are stuck on a problem:

1. Search for the error message or problem - stack overflow is a good coding help resource.
   2. Keep trying to solve it on your own before you give up
2. Use Slack to chat among yourselves.
3. Come to TA office hours.
4. Email the TA or instructor
5. Request a 1 on 1 meeting with the TA