Python Fundamentals for Data Science
Syllabus

Developers:	Paul Laskowski	William Chambers	Kay Ashaolu
Email:	paul@ischool.berkeley.edu	wchambers@ischool.berkeley.edu	kay@ischool.berkeley.edu
Place:	Virtual/Online		

Course Description:
The Python programming language is an increasingly popular tool for the manipulation and analysis of data. This fast-paced course aims to give students the fundamental Python knowledge necessary for more advanced work in data science. The course structure provides students with frequent opportunities to practice writing code, gradually building to an advanced set of skills focused on data science applications. We begin by introducing a range of Python objects and control structures, then build on these with classes and object-oriented programming. A major programming project will reinforce these concepts; give students insight into how a large piece of software is built, and give students experience in managing a full-cycle development project. The last section of the course is devoted to two popular Python packages for data analysis, Numpy and Pandas. The course ends with an exploratory data analysis, in which students apply a script-style of programming to describe and understand a dataset of their own choosing. Aside from Python, the course also spends time on several other technologies that are fundamental to the modern practice of data science, including use of the command line, Jupyter notebooks, and source control with Git and GitHub.

Prerequisites: Due to the fast pace of the course material, previous experience in a general-purpose programming language is required.

Learning Objectives:
After completing this course, students will:

- 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.
- Understand the principles of object-oriented design and the process by which large pieces of software are developed.
- Be able to test and effectively debug programs.
- Know how to use Python to extract data from different type of files and other sources.
- Understand the principles of functional programming.
- 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.

Required Textbook:
Lubanovic, B. (2014). *Introducing Python: Modern computing in simple packages.* Sebastopol, CA: O'Reilly Media.
Course Outline:

1. Programming Languages, the Command Line, and Version Control (1 lecture) The course begins with an overview of programming languages and an introduction to some of the lower-level tools that support the work of a data scientist.
   - Programming Language Characteristics: Interpreted Versus Compiled, Low Level Versus High Level, General Purpose Versus Specialized
   - Using the Command Line
   - Version Control with Git
   - Collaboration with GitHub
2. Python Objects and Basic Control Structures (5 lectures) We continue with a vocabulary-building survey of basic Python syntax, object types, and control structures. These elements are common to virtually all programming applications. Students will gain experience designing algorithms and organizing code logically into functions and modules.
   - Important Python Object Types
   - Iteration and Conditionals
   - Functions
   - The Design of Algorithms
   - Writing and Presenting Code in Jupyter Notebooks
   - Python Modules and Packages
   - Big-O Notation
3. Classes and Object-Oriented Programming (3 lectures) We will spend 3 weeks discussing classes, as well as the larger practice of object-oriented programming. This section of the course will give students a view into how large production systems are organized and developed. At the end, students will complete a significant coding project that will reinforce their understanding of object-oriented development.
   - Classes and Attributes
   - Class Inheritance
   - Object-Oriented Programming
   - Project 1
4. Using Python's Packages for Data Analysis (6 lectures) We introduce the basics of data analysis using Python's system of scientific programming packages. Students learn the common tools that form the basis of the PyData ecosystem and gain experience with programming in a functional style. The final two weeks of the course give students time to work on a final data analysis project, while emphasizing good practices for developers.
   - File Input-Output
   - Text Encoding
   - Common Structure File Formats
   - Functional Programming
   - NumPy Arrays

- Pandas Series and DataFrames
- Basic Data Set Manipulation With pandas
- Plotting With Matplotlib
- Descriptive Statistics
- Test-Driven Development
- Resources for Further Development
- Project 2

Grading:

1. Weekly Assignments - 30%
2. 2 Projects - 40% (20% each)
3. Midterm Exam - 10%
4. Comprehensive Final Exam - 10%
5. Participation - 10%

The weekly assignments are due six days after the corresponding live session. Students will have at least two weeks to work on each group project.

Withdrawing from the Course:
Students who wish to withdraw from the course must do so before the beginning of week 3.

Weekly Assignments:
The weekly assignments are designed to reinforce and extend the programming concepts in each live session. 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.

Group Projects:
There are two large coding projects. The first is an individual project that comes at the end of the discussion of object-oriented programming and allows students to practice designing a multiclass program using best coding practices. The second 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.

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 projects and exams will be given during the school term.

Participation:
Students are expected to participate in class activities, to contribute to discussions held in live session 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:
Please read UC Berkeley's policies around academic integrity: http://sa.berkeley.edu/conduct/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 (for example, on forums like 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 ok 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 instructors for further guidance.