Practical Python with Applications in Finance

Overview

What this Course Is

This is a beginner's practical Python course for students from a financial background. It does not require any previous programming knowledge. The goal of this course is to instill a practical understanding of Python in the context of how it is used in the financial industry; students can expect to come out with the ability to read, understand, and write code in Python for their first day on the job. The examples and case study are geared towards finance; specifically, loan and asset-backed security modeling.

This is a practically-oriented course. It teaches Python programming at a high level, without going into any deep Computer Science theory. Therefore, the lectures will often gloss over the deeper language details.

The primary focus of the course is practical Python. The necessary finance/mathematics information will be introduced as necessary to complete the exercises. Additionally, the finance examples may often be over-simplified, beyond real-life situations, to suit the purpose of the course.

What this Course Is Not

This is not a theoretical or academically-oriented Python course. It does not contain any Computer Science theory and will not give a deep understanding of the mechanics of Python.

This is not a finance or structured-finance course. As the primary focus is on practical Python programming, the finance applications will be introduced as they become necessary. In some cases, you may need to 'accept' certain financial concepts and just focus on implementing the concept into your code.

Prerequisites

Required

- Basic finance
- College mathematics
- Know your way around the computer

Recommended

- Bond pricing knowledge
- Excel

Syllabus

LEVEL 0: INTRODUCTION

0.1: IDE
0.2: Syllabus/Background

LEVEL 1: PYTHON SYNTAX 101

1.1: Variables/Conditionals
1.2: Lists/Loops
1.3: Functions
1.4: Built-in Functions
1.5: Sets/Dictionaries
1.6: Packages

LEVEL 2: OOP

2.1: Classes 101
2.2: Intermediate Classes

LEVEL 3: INTERMEDIATE PYTHON SYNTAX

3.1: Advanced Functions
3.2: Generators 101
3.3: Exception Handling
3.4: Context Managers

LEVEL 4: I/O AND STRING MANIPULATION

4.1: Python Strings
4.2: Logging
4.3: File I/O

LEVEL 5: ADVANCED PYTHON SYNTAX

5.1: Date/Time
5.2: Decorators

LEVEL 6: MONTE CARLO IN PYTHON

6.1: Random Number Generation
6.2: Concurrency

CASE STUDY: ASSET BACKED SECURITIES

Final Project

How to Progress

- Watch **all** videos in a single level. You should proceed in the provided level order.
- Read the *Tips, Tricks, and Pitfalls* document for the level. Note that these documents will be consistently updated with additional tips based on students' experience (feel free to suggest additions!).
- Read and analyze all supplementary material, if applicable.
- Try out the examples demonstrated in the videos:
    - Try out the exact examples demonstrated.
    - Play around with the examples, make changes, etc., to get an intuitive feel for things.
    - The lecture example code is deliberately *not* provided (except in rare cases), as it is important to code things up manually to gain a better understanding.
- Ask questions on the forum until everything is clear.
- Do the HW for the level. Feel free to ask questions on the video or to the instructor (highly encouraged).
- Repeat, for the next level.

How to Ask Questions

- Questions may be on the lectures, material, or exercises.
- Search the lecture comments to see if your question has already been asked/answered.
- If it has already been answered, feel free to ask any follow-up questions to clarify if still unclear.
- If it has not already been answered, email a new question to the instructor.
- Emails should be within the following guidelines:
    - Email subjects should be fully descriptive (don't just post the exercise number, but the exercise number and a brief title on topic of your question).
    - Emails related to exercises should **not** contain any code; they should be conceptual in nature and/or descriptions (in English) of issues encountered. Instructors' responses will also be conceptual in nature – they will guide you to finding the solution on your own, without providing the solution.
- Students are encouraged to respond to other students' queries in the comments, within these guidelines.

Recommended Study Planner

Total time for the course is 8 weeks. After 8 weeks, you will lose instructor support, ability to earn the certificate, and access to the course material.

You may work at your own pace within the allotted 8 weeks, but the below is the recommended pace per level:

Level	# of exercises	Sugg. TTC (weeks)
1	42	1.5
2	13	1
3	20	1
4	17	1.5
5	9	0.5
6	4	0.5
7	Case Study	2
		8

Submission Instructions

- Submissions should be for an entire Level at a time (no individual sections or exercises).
- Every exercise should be submitted as its own script(s). You should have a parent directory for the level, a subdirectory for each section, and a subdirectory for each exercise.
- For more complex programs (which contain multiple scripts and/or its own subfolders), you should put the entire program into the exercise subfolder.
- Each exercise's folder should be names to reflect its number. i.e., **Exercise 1.2.1**
- You should create a single **zip** file with the parent folder for the level.
- The **zip** file should be named with the following format: **student_name Level X submission**
- Levels must be submitted in order.
- You should wait for feedback on your submissions prior to submitting the next level, as this will enable you to incorporate feedback into your future code.

Grading Policy

- Once you submit your exercises for a given level, your instructor will provide detailed feedback on your code. This will usually occur within 24-48 hours of submission; often quicker, rarely slower.
- Each exercise is assigned a grade out of 100. The score for a level is the straight average of all the exercises in that level.
- Each level (and the final project) is assigned an overall course weight. Your final score is the weighted average of all the levels and final project. The level weights are as follows:

Level	Grade Weight
1	10%
2	20%
3	15%
4	15%
5	10%
6	5%
Final Project	25%
Total	100%

- You will need a minimum of a **70**, weighted average score to earn the certificate. A minimum score of **90** will earn you a certificate with distinction.

Grading Criteria

- Each exercise is graded out of 100. An exercise that is **code correct** *and* meets the specification of the instructions gets an 80. The extra 20 points are devoted to code commenting, code format, code conciseness/clarity, efficiency of code, and taking the optimal coding approach (within what was already taught).
- You are expected to build upon your knowledge as you progress through the course. Therefore, once a concept is taught, you are expected to incorporate it into your future code when applicable. Taking an earlier (non-optimal) approach will cost you points.