# FRE6831

# COMPUTATIONAL FINANCE LABORATORY (PYTHON)

**Edward D. Weinberger, Ph.D., F.R.M**
**Adjunct Professor**
**Dept. of Finance and Risk Engineering**
**edw2026@nyu.edu**
**Office Hours by appointment**

This half-semester course introduces the Python programming language. Given that generative AI's such as ChatGPT can code in Python or just about any other language, one can easily ask, "Why bother learning *any* computer language?". The short answer is that, while the various generative AI's are capable of writing computer code, they won't necessarily write *correct* code, *i.e.* code that correctly implements requirements, and they certainly aren't guaranteed to write code that runs efficiently, etc. So code reviews by human experts will be needed for the foreseeable future. Courses like this one can help you become one of those experts.

Python expertise, in particular, is in demand, partly due to

- Python's many extensions (NumPy, Pandas, SymPy, etc.),
- Python's dominance in AI/machine learning, and
- Readable syntax.

It is particularly of interest in finance because a version of it is being used as the "glue" that holds together the computing infrastructure of several major financial institutions (Bank of America, JP Morgan Chase, and Goldman Sachs have working systems, and I'm told that Barclays is working on it.).

Python is sufficiently quirky that a half-semester course must necessarily focus on the language itself, as opposed to specific financial applications. Python basics, just like the basics of English, are reasonably easy to learn. However, just as there is a world of difference between a native English speaker and one that is merely fluent, there is a world of difference between mastering the basics and becoming a true Pythonista. Hopefully, there will be time to present some of this advanced material towards the end of the course.

## **Prerequisites**:

Students will be expected to have fluency in an object oriented language, such as C++, Java, or C#, as this course is intended to introduce students to Python, not programming in general, or to the object oriented paradigm in particular.

## **Required text**:

*The Quick Python Book* by Naomi Cedar, 4th Ed., Manning, 2025, ISBN-13: 978-1633436336.

## **Recommended Reading**:

Just about everything there is to know about Python can be found somewhere on the web, most directly by asking your favorite AI. Non-AI methods of accessing Python related information include

- Googling "Python <name of feature>",
- Posts on `stackoverflow.com`,
- The standard documentation maintained by the Python Software Foundation, https://docs.python.org/3/, which is both definitive and surprisingly readable.

## **Grading**:

There will be two components to your grade:

1. Correctness of questions that I ask in class (50%). From time to time, I will ask about such things as the likely output of a program, etc. The questions will be sufficiently simple that any student that is following the lecture could reasonably be expected to answer them without too much thought. My TA will record whether a question is answered correctly. I expect to ask each of you to provide correct verbal answers to 5 such questions, so each correct answer earns 10 points towards your final grade. Since I hope to ask lots of questions, you can keep trying for 5 correct answers until you have reached that goal.

   *In order to facilitate the tracking of who answered which questions correctly, I ask that you sit in alphabetical order by family (last) name, as per instructions given in the first class.*

2. Results of attempts to use 2 generative AI's to prepare a Python class that implements bond analytics, as per the document "AI aware Project Description" and PowerPoint slides referenced therein (50%). Please submit a WORD or PDF document (Name and NYU ID # please!) describing
   - The prompt that you submitted to each AI (which could be the same prompt for both),
   - Which version of the code that you decided to correct,
   - What you did (running test cases, code inspection, etc.) to determine to how to correct that code,
   - What those corrections were.

   In addition, please submit 3 text files, each with ".py" suffix (*not* a Jupyter notebook!). Two of these files contain the code that each AI generated without any corrections and the third contains the version of the code that you believe to be error free.

   I will run the same tests that I used above to test your code, hence the ".py" requirement for your finished code.

## Detailed Course Outline

Notes:

- While I am willing to commit to the following choice of topics and the order of presentation, I don't know how long it will take to make these presentations. I therefore cannot be sure of the lecture in which I will present each topic.
- Lectures will include some of the experiments with generative AI proposed by our textbook.

Topic 1: Introduction to the Course and to Python

I. Course "Mechanics"
II. Observations on the FinTech eco-system
    a. 50 years of programming: What has changed and what hasn't
        i. Turing's theorem on the existence of a universal computer
        ii. Basics of Python via Pictorial Programming
        iii. Data, data, everywhere! Hence objects
            1. Input sources
            2. Databases
            3. Need to process disparate data elements, hence the need for objects
            4. In Python, *everything* is an object
    b. $O(N)$ vs $O(\log N)$ vs $O(1)$ implementations
    c. Industrial strength programming
III. How Python fits in
    a. Why Python?
        i. Elementary Python easy to learn, but also "expert friendly"
            1. Addresses some annoying things in other languages
            2. Can do a lot in a few lines
                a. "batteries included" libraries
                b. Expressive syntax, most notably lists and dictionaries
        ii. Intended to be readable
        iii. Free, but very well supported
        iv. Many, many extensions (SciPy, NumPy, SymPy, AI libraries, etc.)
        v. Multi-platform (no platform dependencies)
        vi. Full support for object-oriented programming, including operator overloading, but without
            1. explicit garbage collection
            2. explicit pointers
    b. Problems with Python (primarily because Python is interpreted)
        i. Slower than C/C++
IV. Characteristics and Quirks of Python
    a. Readability is key; hence indents used as block identifiers
    b. Python is interpreted, but …

c. Python uses “duck typing” and automated garbage collection
d. Lots of introspection

V. Installing Python
a. “Hello, world!”
b. Libraries

VI. The very beginnings
a. Numbers: `int float complex`
b. Strings
c. Booleans
d. None
e. Built-in functions: https://docs.python.org/3/library/functions.html
f. Dates via `datetime`

Reading: Cedar, Introduction and Chapters 1 - 4

Topic 2: Lists, Tuples, and Sets

I. Lists
a. “Declaring” a list
b. Arrays, but with a twist!
i. Length unspecified beforehand; entries added at end
ii. List operators (append, indicies/slices, etc.)
iii. List operations
iv. Lists as queues and stacks
v. Nested lists and deep copies

II. Tuples
a. Mutability vs Immutability
b. Declaration
c. List-tuple conversion
d. Packing/unpacking tuples

III. Sets
a. Uniqueness of elements
b. Set

operations Reading: Cedar,

Chapter 5 Topic 3: Strings

I. Strings as immutable sequences of characters, including special characters
II. `str` vs `repr`
III. String methods
a. `split` and `join`
b. Conversions
c. Other string methods
IV. The many ways of formatting and printing strings
V. The `bytes` data type

VI. Unicode basics

Reading: Cedar, Chapter 6

Topic 4: Dictionaries

I. Review: Hashing
II. Definition as an associative array with immutable
III. Dictionary operations
IV. Some applications

Reading: Cedar, Chapter 7

Topic 5: Control Flow

I. Statements, blocks, and indentation
II. Boolean values and expressions
III. Standard stuff: `if` and `while`
IV. Loops over sets
    a. The `range` function
    b. `break` and `continue`
    c. tuple unpacking
    d. `enumerate` and `zip`
    e. list comprehensions
    f. generat

ors Reading: Cedar,

Chapter 8 Topic 6: Python

Functions

I. Definition and scoping
II. Function parameter options
III. Lambda expressions
IV. Functions assignment to “pointer” variables
V. Decorators
VI. Generator functions
VII. The `dir` function
VIII. Comments and doc strings

Reading: Cedar, Chapter 9

Topic 7: Objects in Python

I. Basics of object definitions
II. Member vs class variables
III. Static and class methods

IV. Inheritance
V. Private variables and methods
VI. Operator overloading Reading:

Cedar, Chapter 15 and 17 Topic 8:

Modules and Programs

I. Setting up a module
II. Local and global variables
III. The `import` statement
IV. The `main` statement and Python programs
V. Scoping rules

Reading: Cedar, Chapters 10 and 11

Topic 9: Input and output

I. File objects
II. Reading command line parameters

Reading: Cedar, Chapters 12 and 13

Topic 10: Exceptions

Reading: Cedar, Chapter 14

Topic 11: Regular Expressions

Reading: Cedar, Chapter 14

Various advanced topics, chosen from the following:

I. Multi-threading
II. Unit testing and the mock library
III. Newer features
    a. Sorted dictionaries
    b. Ways of declaring variables with a given type
IV. Functional programming
V. Python and SQL databases
VI. Other topics, to be determined

Reading: Cedar, chapters to be determined; other sources to be determined

## Diversity Statement

*The NYU Tandon School values an inclusive and equitable environment for all our students. I hope to foster a sense of community in this class and consider it a place where individuals of all backgrounds, beliefs, ethnicities, national origins, gender identities, sexual orientations, religious and political affiliations, and abilities will be treated with respect. It is my intent that all students' learning needs be addressed both in and out of class and that the diversity that students bring to this class be viewed as a resource, strength, and benefit. If this standard is not being upheld, please feel free to speak with me.*