Course 11-603: Practical Programming with Python

Course Description & Syllabus

Carnegie Mellon University
Summer 2022

1. Overview

Title: Practical Programming With Python
Units: 9 units
Pre-requisites: Introduction to Computing or equivalent experience

Carnegie Mellon's LMS access: https://canvas.cmu.edu

Lecture: 40-60 minute video released at the start of each module
TA Office Hours: Remote, see below
Recitation: In-person/Remote, Thu 9 am - 10 am EST
Weekly check-in with instructors: Remote, Thu 3 pm - 4 pm

Instructors:

Jaromir Savelka
jsavelka@cs.cmu.edu
Office Number: GHC 7225

Annette Han
ahan2@cs.cmu.edu
Office Number: WEH 3126

Marshall An
haokanga@andrew.cmu.edu
Office Number: GHC 6025

Teaching Assistants:

Aadarsh Pratik
apratik@andrew.cmu.edu
Office hours: Saturday 1 - 2 pm EST (Zoom)

Zhihao Wang
zhihaowa@andrew.cmu.edu
Office hours: Wednesday 3 - 4 pm (Zoom)

Hao Wu
haowu3@andrew.cmu.edu
Office hours: Friday 4 - 5 pm (Zoom)

2. Course Description

Students learn the concepts, techniques, skills, and tools needed for developing programs in Python. Core topics include types, variables, functions, iteration, conditionals, data structures, classes, objects, modules, and I/O operations. Students get an introductory experience with several development environments, including Jupyter Notebook, as well as selected software development practices, such as test-driven development, debugging, and style checking. Course projects include real-life applications on enterprise data and document manipulation, web scraping, and data analysis.

3. Course Goals

In this course, students gain hands-on experience solving real-world problems by completing programming projects in Python focused on the language fundamentals, software development practices, data manipulation and analysis, web scraping, and automatic document processing. Specifically, students are exposed to realistic software development projects, real-world data, and scenarios in order to learn how to:

1. Explain and use fundamental constituents of the Python programming language to solve a variety of computational problems.
2. Read data stored in different data formats, such as CSV, JSON, and XML, or database systems, such as MySQL or MongoDB, transform it, and persist the transformed data in a different file format or a database system using Python.
3. Recognize and utilize well-established software development practices.
4. Process common office document formats and obtain information from the web or publicly available APIs using Python.
5. Analyze and visualize real-world data using Python data science libraries.

Through this process, we aspire for our students to become sophisticated, independent, and resilient problem solvers who are able to overcome challenges and learn.

4. Learning Outcomes

In this project-based course, we have conceptual and project learning objectives.

The conceptual learning objectives (LOs) are the following. Students will be able to:

1. Explain the core building blocks of a programming language, such as variables, user-defined, and built-in functions.
2. Explain the flow control techniques (iteration, conditionals).
3. Describe the fundamental programming data structures and their implementation in Python (lists, sets, tuples, dictionaries).
4. Discuss the object-oriented programming paradigm and its purpose.
5. Discuss fundamental software development lifecycle and practices (e.g., top-down design, test-driven development, object-oriented analysis, and design).
6. Describe common document formats and considerations related to web scraping and office document processing and manipulation.
7. Explain the fundamental principles of database design, common data formats, and programming interfaces.

8. Discuss and compare different data storage, access solutions (flat files, SQL, NoSQL, key-value stores), and application programming interfaces (APIs).

9. Discuss data loading, cleaning, manipulation, analysis, and visualization.

4.1. Project Learning Objectives

The project learning objectives (LOs) are the following. Students will be able to:

1. Write small to medium Python scripts using variables, built-in and user-defined functions.

2. Write more advanced scripts using conditionals and iteration (control flow).

3. Employ Python data structures, such as lists, dictionaries, sets, and tuples.

4. Use libraries implemented in an object-oriented fashion and interact with classes and objects imported from those libraries.

5. Employ fundamental software development practices (e.g., top-down design, test-driven development, style, linting, pep8, documentation).

6. Scrape information from the web or query publicly available APIs and extract data from common office document formats, transform it and present it using a different appropriate format.

7. Contrast among and experiment with different data storage and access solutions (CSV, JSON, XML, SQL, NoSQL, key-value stores).

8. Analyze and visualize real-world data using Python data science libraries.

4.2. Course organization

Your participation in the course will involve several forms of activity:

1. Reading the conceptual content for each unit.
2. Completing the graded weekly assessments after each unit.
3. Complete projects which are hands-on training and automated feedback.
4. Complete the final exam which is hands-on training with instructor provided feedback.

5. Getting help

Students are encouraged to ask questions about content and projects through the Q & A forum. The course link for the forum is:

https://piazza.com/cmu/summer2022/11603/home

6. Policies

Working Alone on Projects

Projects that are assigned to single students should be performed individually.

Handing in Projects

All assessments are due at 11:59 PM ET (one minute before midnight) on the due dates specified on the Sail() Platform. All hand-ins are electronic.

Appealing Grades

After each project module is graded, you have seven calendar days to appeal your grade. All your appeals should be provided by email to the instructor.

7. Assessment

Inline activities (assessme) which are available throughout conceptual content, are simple, non-graded activities to assess your comprehension of the material as you read through the material. You are advised to put honest effort into and complete all of the inline activities before proceeding through to the primers or projects. If you struggle with many of the activities, it is recommended that you review the material again.

The conceptual units consist of modules of content marked as "[Concepts]" on Sail(), each week has a Checkpoint Quiz that you must complete before the deadline posted on Sail(). Checkpoint Quizzes will be worth 20% of your total grade. It is your responsibility to ensure that the quiz is submitted prior to the deadline. You will have only a single attempt to complete each Checkpoint Quiz.

This course includes several individual projects. Each project module has to be completed based on the deadlines posted on Sail(). The write-up required to complete each project module is available on Sail(). Each module has a submission process that is specific to the project module that is due. It is the students' responsibility to make sure that all project work is completed and that the project module is submitted prior to the deadline. Students typically have multiple attempts to submit the project module on Sail().

This course also includes a final exam that will be completed towards the end of the course. The final exam is similar to the individual project but it is not auto-graded. Instead, the instructors and teaching staff will provide students with feedback and grading.

Projects and the final exam project will be worth the remaining 80% of your total grade.

Type	Number	Weight
Conceptual Content Quizzes	8	20%
Projects and Final Exam	8 + 1	80%
Total Grade		100%

8. Cheating

We urge each student to carefully abide by the course policy on academic integrity, which outlines the policy on cheating, plagiarism, or unauthorized assistance. It is the responsibility of each student to produce her/his own original academic work. Collaboration or assistance on academic work to be graded is not permitted unless explicitly authorized by the course instructor. Each unit checkpoint quiz or project module submitted must be the sole work of the student turning it in. Student work on the cloud is logged, submitted work will be closely monitored by automatic cheat checkers, and students may be asked to explain any suspicious similarities with any piece of code available. The following are guidelines on what collaboration is authorized and what is not:

What is cheating?

1. Sharing a solution, code, or other electronic files by either copying, retyping, looking at, or supplying a copy of any file. Copying any solution, code from the internet (stackoverflow.com or GitHub or others). Do not use other students' solutions or code to "test" the auto-grader. Anything you submit to the auto-grader must be your work.
2. Copying answers to any checkpoint quiz from another individual, published or unpublished written sources, and electronic sources.
3. Collaborating with another student or another individual on checkpoint quizzes or project modules.
4. Sharing written work, looking at, copying, or supplying work from another individual, published or unpublished written sources, and electronic sources.

What is **not** cheating?

1. Clarifying ambiguities or vague points in-class handouts.
2. Helping others use computer systems, networks, compilers, debuggers, profilers, or system facilities.
3. Helping others with high-level design issues.
4. Guiding others through code debugging but not debugging for them.

Cheating in projects will also be strictly monitored and penalized. Be aware of what constitutes cheating (and what does not) while interacting with students. You cannot share or use solutions, written code, and other electronic files from students. If you are unsure, ask the teaching staff.

Be sure to store your work in protected directories. The penalty for cheating is severe, and might jeopardize your career – cheating is simply not worth the trouble. By cheating in the course, you are cheating yourself; the worst outcome of cheating is missing an opportunity to learn. In addition, you will be removed from the course with a failing grade. We also place a record of the incident in the student's permanent record.

9. Conceptual Topics

The course content will be structured into the following modules:

Module	Title	Learning Objectives
0	Programming and Python	• Explain what a computer program is. • Explain what programming languages are. • Explain the difference between high-level and low-level programming languages. • Define a programming language interpreter and a compiler.
1	Python Basics and Introduction to Functions	• Explain what Python is and how to use it to run single-line expressions as well as small multi-line programs. • Define the fundamental arithmetic and relational operators in Python and describe how to use them in connection with brackets to construct single-line expressions in Python. • Define the integer, float, string, and boolean Python data types. • Explain what variables are and the effects of variable assignment and reassignment. • List and describe the behavior of the print, input, len, int, str, float, and bool built-in functions. • Explain the importance of comments and docstrings in coding. • Trace the execution of a simple Python program using basic data types, operators, built-in functions, and variable assignments. • Explain syntax, runtime, and semantic errors.

		• Explain what a function is. • List several Python built-in modules and explain what it means for a Python function or a module to be built-in. • State the Python syntax for importing modules and sub-modules or specific functions and constants. • Explain how to define custom (i.e., user-defined) functions and explain the role of parameters and return values in a user-defined function. • Differentiate between a function definition and a function call as well as between function parameters and arguments. • Differentiate between a function returning a value and a function having a side-effect. • Explain the concept of scope and differentiate between a global and a local scope. • Discuss how to use properly named functions and variables to improve code quality.
2	Flow Control, Strings, Input and Output	• Explain how conditional statements are used to solve computational problems involving decisions. • Determine the value of a given Boolean-valued expression in Python. • Apply relational and logical operators and functions to form Boolean-valued expressions in Python. • Describe the Python syntax for conditional expressions. • Trace Python code containing conditional statements. • Explain how iteration is used to solve computational problems involving repetition. • Identify an iterative algorithm and express it using a loop structure. • Describe the Python syntax for while loops and for loops. • Trace the execution of a Python program involving iteration. • Describe the fundamental properties of strings in Python. • Describe common string operations in Python. • Explain how to format strings in Python. • Explain how to read and write files in Python. • Describe the Python syntax for reading and writing files using the with statement.
3	Python Data Structures	• Describe what data structures are and recall basic sequence mechanisms and operations. • Explain the purpose of the Python list data container and describe its fundamental properties. • Describe how to modify a list and how to access and iterate over its elements. • List and define several fundamental list operations. • Explain the purpose of the Python dict data container and describe its fundamental properties. • Describe how to modify a dict and how to access and iterate over its elements. • List and define several fundamental dict operations. • Explain the purpose of the Python set data container and describe its fundamental properties. • Describe how to modify a set and how to access and iterate over its elements. • List and define several fundamental set operations. • Explain the purpose of the Python tuple data container and describe its fundamental properties. • Describe how to modify a tuple and how to access and iterate over its elements. • List and define several fundamental tuple operations.

		• Contrast the use cases where a list, a dict, a set, or a tuple is the appropriate solution in Python. • Contrast the behavior of a list to the behavior of a set when it comes to the ordering of elements and their inclusion. • Contrast the behavior of a list to the behavior of a tuple (immutability).
4	Object-oriented Programming with Python	• Explain what object-oriented programming is, describe its core principles, and contrast the approach to procedural programming. • Discuss the benefits and shortcomings of object-oriented programming. • Observe how object-oriented programming concepts are used in modules from the standard Python library. • Explain the concepts of a class and an object and relate the two in a way that a class is considered a blueprint for an object. • Discuss the concept of a class and its objects as a logical grouping of data and methods that operate on them. • Describe the implementation of classes and objects in Python with attributes and methods. • Explain the role of the most prominent special methods in Python, such as __init__, __str__, __len__, and __eq__. • Describe encapsulation, inheritance, composition, and polymorphism in object-oriented programming and explain their benefits? • Recognize the utility of object-oriented approach for improving code organization, readability, and maintainability.
5	Software Development	• Discuss the importance of keeping the code clean. • Explain the concept and the role of code style standards. • Describe the fundamental code style requirements for Python programs defined in Python Enhancement Proposal (PEP) 8. • Describe several Python style checking and formatting tools. • Describe common code design principles, including DRY, KISS, and separation of concerns. • Discuss the importance of comprehensive automated testing during software development. • Explain the concept of test-driven development. • Explain the concept of test coverage. • Describe several popular tools supporting test-driven development with Python. • Explain what are some common approaches to distribute Python code to the intended users and what challenges do these techniques overcome. • Describe the basic functionality of Package installer for Python (pip).
6	Files and Datastores	• Describe fundamental properties of comma-separated values (CSV), JavaScript object notation (JSON), and extensible markup language (XML) file formats. • Explain the utility and typical use cases of CSV, JSON, and XML file formats. • Discuss the typical concerns related to processing data stored in CSV, JSON, and XML files. • List several mainstream Python libraries focused on processing CSV, JSON, and XML files. • Describe how to load, manipulate, and export data stored in CSV, JSON, and XML files to/from a Python program.

		• Describe the fundamental properties of Relational Database Management Systems (RDBMS) and Structured Query Language (SQL). • Identify several mainstream RDBMSs and their key differences. • Discuss the advantages and limitations of mainstream RDBMSs. • Explain how to query a MySQL instance using Python libraries. • Discuss the main motivations behind Not Only SQL (NoSQL) systems. • Describe the fundamental properties of NoSQL systems. • Describe the four main NoSQL systems types and name several mainstream systems from each category. • Explain how to query a MongoDB instance using Python libraries.
7	Web Scraping	• Explain what web scraping and web crawling are as well as their ethical and legal implications. • Describe the HTML and CSS languages, their utility, and typical use cases. • Explain how to use common Python libraries to download HTML files automatically from the Web. • Explain how to use common Python libraries to extract information from HTML files. • Discuss some concerns related to HTML file parsing.
8	Data Analysis	• Define the basic types of data and describe their properties. • Define the ETL process and describe its overall purpose as well as the purpose of its stages (extract, transform, load). • Describe the basic features of Jupyter Notebook and explain why it is often the IDE of choice for various data engineering and data science tasks. • Explain the role and basic functionality of the pandas Python library. • Describe the fundamental properties of pandas Series and DataFrame data containers. • Discuss the importance of properly visualizing data as well as some common pitfalls. • Describe several common data visualizations and associate them with the appropriate type of data. • Give examples of incorrect/manipulative data visualizations and explain in what way they are incorrect/manipulative. • Explain the basic concept of regular expressions and discuss various use cases.

10. Projects

The projects are geared towards providing hands-on experience. Students will learn to develop all projects using various public cloud services. For each project, students are expected to work within a specified budget otherwise they risk being penalized, and fulfill the following learning objectives.

10.1. Project 0: First Interaction with Python Code

- Set up a Python project development environment on a local machine to meet the project requirements.
- Submit a solution of a task for grading on Sail() from a local machine, given properly configured project environment, to obtain assessment and feedback from Sail().

- Inspect the outputs generated from a submission to Sail() to discover the outcome and contextualized feedback from the auto-grader to iterate on the solution if needed.
- Modify files with Python code from a local machine, given simple and precise step-by-step instructions, to achieve a desired effect.

10.2. Project 1: Types, Variables, Functions

- Assign values to variables that are utilized in a Python program and observe the effects of the assignment on the behavior of the program.
- Utilize primitive data types, operators, and built-in functions to implement simple user-defined functions.
- Import built-in modules and use the available functions and constants to implement simple user-defined functions.
- Observe the difference between a function returning a value and a function having a side-effect.
- Write and run simple Python scripts utilizing variables and user-defined functions to improve code organization and readability.
- Distinguish code from comments in a small-sized Python file.
- Use assert statements to implement test functions to evaluate the correctness of provided functions.

10.3. Project 2: Iteration, Conditionals, Strings and Basic I/O

- Use iteration to solve computational problems involving repetition.
- Implement for loops and while loops in Python.
- Trace execution of a program involving iteration.
- Design and implement algorithms using nested iteration.
- Use conditional statements to solve computational problems involving decisions.
- Apply relational and logical operators to form Boolean expressions.
- Recognize the effects of flow control mechanisms (iteration and conditionals) on code organization and readability.
- Create, manipulate, and format strings in Python.
- Read, write, and modify text files using Python.

10.4. Project 3: Lists, Sets, Tuples, Dictionaries

- Create an empty list, dict, set or tuple in Python.
- Access elements of a list and a tuple by an index, access elements of a dict by a key, and test for element membership in a set.
- Add an element to an existing list, dict and set, and produce a new tuple from an existing one with the extra element.
- Remove an element from an existing list, dict and set, and produce a new tuple from an existing one with the element removed.
- Update an element in an existing list, dict, set and tuple or produce a new tuple from an existing one with the modified element if needed.
- Perform a conversion among list, dict, set and tuple in Python.
- Load data from multiple files into a nested custom Python data structure for further processing.
- Transform a custom nested data structure into a different data structure.
- Sort data in a container per specified criteria.
- Use custom data structures to generate a report persisted as a static file.

10.5. Project 4: Classes, Objects, Attributes and Methods

- Implement classes in Python, with attributes and methods.
- Experiment with class inheritance for code reuse.
- Implement and use custom Error types.

- Re-organize flat code from a single file into several files (modules) and classes.
- Demonstrate the usefulness of classes and modules for code organization and readability.

10.6. Project 5: Style, Testing, Packaging

- Fix style mistakes in a Python program to improve the code quality and maintainability using automatic style checkers and linting.
- Improve the quality of a complex code written by someone else.
- Design and implement complete test suites achieving 100% test coverage using Python test and coverage libraries.
- Generate a built distribution and source archive from a custom Python project.
- Install and uninstall a custom module from a local package using pip.

10.7. Project 6: Files and Data Stores

- Utilize common Python libraries to load, manipulate, and store data in CSV, XML, and JSON formats.
- Load data in one format and store them in another.
- Filter data based on different criteria.
- Create a MySQL database and a table.
- Load data from a CSV file into the MySQL table.
- Query the MySQL table from Python.
- Create a MongoDB database and a collection.
- Load data from a JSON file to MongoDB collection.
- Query the MongoDB collection from Python.

10.8. Project 7: Web Scraping and Office Document Processing

- Iteratively download information from multiple websites using the Python requests package.
- Extract relevant information from an HTML document using the Python beautifulsoup4 package.
- Automatically create an Excel spreadsheet using the Python openpyxl package.
- Read data from an Excel spreadsheet using the Python pandas package.
- Automatically generate a batch of Word documents using the Python python-docx package.

10.9. Project 8: Data Analysis

- Prepare data stored in various formats so that they can be conveniently loaded into a pandas DataFrame.
- Use several pandas loading functions in order to conveniently ingest data stored in various formats (and having various shapes) into a pandas DataFrame with a desired shape (i.e., index and columns/fields).
- Merge multiple pandas DataFrame objects into a single one and thereby achieve integration of data from multiple widely different sources into a unified data set.
- Persist pandas DataFrame using Python pickle module.
- Examine the created data set stored in pandas DataFrame to obtain basic descriptive statistics.
- Assess the data set on its usability and basic properties of the captured statistics.
- Generate systematic reports on the usability and basic properties of the statistics and store it as an Excel spreadsheet.
- Utilize regular expression to extract patterns (email addresses) from a large data set of text documents (email messages).
- Compare the expressiveness of regular expressions to that of simple text matching.

11. Schedule

The tentative schedule is as follows (specific deadlines are posted on Sail()):

Week	Conceptual Content	Quiz	Hands-on Projects
1	M0: Programming and Python	Quiz 0	M0: First Interaction with Python Code
2	M1: Python Basics and Introduction to Functions	Quiz 1	M1: Types, Variables, Functions
3	M2: Flow Control, Strings, Input and Output	Quiz 2	M2: Iteration, Conditionals, Strings, Basic I/O
4	No new content week		
5	M3: Data Structures	Quiz 3	M3: Lists, Sets, Tuples, Dictionaries
6	M4: Object-oriented Programming	Quiz 4	M4: Classes, Objects, Modules
7	M5: Software Development	Quiz 5	M5: Style, Testing, Packaging
8	No new content week		
9	M6: Data Manipulation	Quiz 6	M6: Files and Datastores
10	M7: Web Data and Office Documents	Quiz 7	M7: Web Scraping and Office Document Processing
11	M8: Data Analysis	Quiz 8	M8: Exploratory Analysis
12	No new content week		
13	Final Exam		

12. Accommodations for Students with Disabilities

If you have a disability and have an accommodations letter from the Disability Resources office, I encourage you to discuss your accommodations and needs with me as early in the semester as possible. I will work with you to ensure that accommodations are provided as appropriate. If you suspect that you may have a disability and would benefit from accommodations but are not yet registered with the Office of Disability Resources, I encourage you to contact them at access@andrew.cmu.edu.

13. Take care of yourself

Do your best to maintain a healthy lifestyle this semester by eating well, exercising, avoiding drugs and alcohol, getting enough sleep, and taking some time to relax. This will help you achieve your goals and cope with stress.

All of us benefit from support during times of struggle. You are not alone. There are many helpful resources available on campus and an important part of the college experience is learning how to ask for help. Asking for support sooner rather than later is often helpful.