C S 3B: INTERMEDIATE SOFTWARE DESIGN IN PYTHON

Foothill College Course Outline of Record

Effective Term:	Summer 2023
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Prerequisite:	C S 3A.
Advisory:	Demonstrated proficiency in English by placement via multiple measures OR through an equivalent placement process OR completion of ESLL 125 & ESLL 249.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU/UC
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to use the Python environment to define the basic abstract data types (stacks, queues, lists) and iterators of those types to effectively manipulate the data in his or her program.
- A successful student will be able to write and debug Python programs which make use of inheritance, i.e., the "is a" relationship, common to all OOP languages. Specifically, the student will define base and derived classes and use common techniques such as method chaining in his or her programs.

Description

Systematic treatment of intermediate concepts in computer science through the study of Python object-oriented programming (OOP). Coding topics include Python sequences, user-defined classes and interfaces, modules, packages, collection classes, threads, lambda expressions, list comprehensions, regular expressions and multi-dimensional arrays. Concept topics include OOP project design, recursion, inheritance, polymorphism, functional programming, linked-lists, FIFOs, LIFOs, event-driven parsing, exceptions, and guarded code.

Course Objectives

The student will be able to:

1. Configure a Python Development Environment for advanced Python programming
2. Use both instance members and class members, as appropriate, in class design
3. Analyze and demonstrate the use of multi-dimensional arrays in Python
4. Design, implement, and test Python programs that use class inheritance/specialization, and explain why this is an example of the "is-a" relationship
5. Demonstrate the use of advanced functional programming
6. Explain how guarded code is implemented in Python through exceptions
7. Express numbers in decimal, binary, and hexadecimal representations, and use bitwise logical operators to process data at the bit and byte level
8. Demonstrate a working knowledge of basic abstract data types and their Python-based collection classes
9. Produce end-user programs which feature event-driven techniques and multi-threading to provide a sensible and easy-to-use GUI
10. Explain what a Python abstract base class (ABC) is and how it is used
11. Describe declaration models for run-time storage allocation, garbage collection, and type checking, and compare with other languages like C++ or Java
12. Use some of the Python collections to write efficient and portable application programs
13. Write to and read from files using intermediate file I/O operations in a Python program
14. Design, implement, test, and debug intermediate-level Python programs that use each of the following fundamental programming constructs: string processing, numeric computation,

simple I/O, arrays, and the Python API

15. Write applications that solve problems in one or more application area: mathematics, physics, chemistry, cellular automata, 3-D simulation, astronomy, biology, business, internet

Course Content

1. Setting up a complete Python environment
    a. Using the Python.org website
    b. Configuring the PATH and other environment variables
2. The proper use of class and instances members and methods
    a. Class attributes
    b. Instance attributes
    c. Scope and namespace usage of class and instance attributes
    d. Modules vs. classes
    e. Comparison of instance, class, and static methods
    f. Deep vs. shallow copies with copy() and deepcopy()
    g. Operator overloading
3. Multi-dimensional arrays and the NumPy Module
    a. Multi-dimensional arrays using lists of list
    b. Multi-dimensional arrays using NumPy
    c. Limitations of Python array class
    d. List iterators vs. explicit index values
    e. Binary searching as a logarithmic time complexity example of recursion
4. Inheritance
    a. The "is a" relationship
    b. Extending existing base classes
    c. Derived classes (subclasses) and class hierarchy
    d. Member method overriding
    e. Intended private, public, and default members
    f. Encapsulation and Polymorphism

5. Advanced OOP
    a. Constructor chaining
    b. Member method chaining
    c. Key (named) parameters
6. Error handling and event-driven programming
    a. Built-in Python exception objects
    b. Custom exception declarations
    c. Manually raising exceptions
    d. Alternatives to exceptions
7. Non-decimal arithmetic and related types
    a. Bitwise numeric operators
    b. Bitwise logical operators
    c. Binary and hexadecimal constants
    d. Enum types (int and non-int)
8. Container datatypes
    a. Abstract Data Types (ADTs) and Python Collections
    b. List-like, dict-like, and wrapper containers
    c. The vector and linked-list ADT and Python's deque objects
    d. The "hash-able" ADT and Python's Counter objects
    e. Implementing ADTs through inheritance
9. Topics in Graphical User Interfaces (GUI) and multi-threading
    a. GUI through Tkinter module
    b. Tkinter widgets (message, button, label, text, canvas)
    c. Layout options (pack, grid)
    d. Event handlers: built-in vs. custom
    e. Multi-threaded using the Python threading.py library
10. Abstract Base Classes (ABCs)
    a. Metaclass for defining ABCs
    b. Defining and using interfaces
11. Storage allocation models and typing in Python

a. Python memory management, symbol tables, and execution model
b. Run time object binding and storage management of activation records in Python
c. Binding, visibility, persistence, and lifetime of variables
d. Strong typing vs. dynamic typing
e. Duck typing vs. isinstance() and hasattr()

12. Specifics of Python Collections
    a. The namedtuple() method
    b. Deque used to implement queue and stack
    c. Counter for rapid tallies
    d. Defaultdict for key-value pairs and lists
13. Topics in Python file I/O
    a. Output formatting
    b. File objects
    c. JSON use in Python
14. Essential examples and assignment areas in Python
    a. String/text processing
    b. Numeric computation
    c. User interaction
    d. Multi-class projects and compound data types
    e. Inheritance-based projects
    f. Representative GUI and/or threaded project with event-driven design
15. Applications used throughout course in selected areas
    a. Math
    b. Physics
    c. Chemistry
    d. Biology
    e. Astronomy
    f. Business and finance
    g. Internet

Lab Content

1. Familiarization with the intermediate-level online development environment
    a. Modify and customize the environment variables that affect the Python installation
    b. Create multi-file programming projects
    c. Gain experience with the steps needed to edit a complex program
    d. Modify editor settings to produce an industry standard code style
2. Organizing and debugging multi-class projects
    a. Demonstrate the ability to debug programs that contain multiple classes
    b. Distinguish between interface and implementation in projects by creating classes that are independent of I/O modality
    c. Write individual component classes that are independent of client use and can serve several client programs
    d. Emulate symbolic constants by not providing setters
    e. Implement static and instance members into classes in a way that demonstrates a mature understanding of object-oriented programming (OOP) in a lab project
3. Exploring advanced array constructs in class design
    a. Gain experience in effectively using single and multi-dimensional arrays as class members
    b. Apply nested loops to process multi-dimensional arrays
    c. Use the Python environment to debug errors in multi-dimensional arrays
    d. Solve problems using fixed-size and dynamic sized arrays, as appropriate
4. Demonstrating competence in intermediate level algorithm design using classes within the Python environment
    a. Use Python tools and text editors to implement a multi-faceted algorithm and/or simulation that makes effective use of OOP
    b. Evaluate and comment on other students' algorithms
    c. Utilize a combination of string processing and numeric processing to address various aspects of the algorithm implementation
    d. Produce clear program runs which demonstrate that the algorithm addresses a variety of cases and/or input states
    e. Incorporate bitwise and logical operations to address binary logic tasks within an algorithm
5. Building a program that uses class inheritance to demonstrate how re-use is handled in OOP

a. Create a project that contains at least one class intended to be used as a base class
b. Derive (sub-class) one or more classes from the base class
c. Use method overriding to avoid code duplication between base classes and derived classes

6. Incorporating basic abstract data types in programming projects
   a. Implement a fundamental abstract data type (ADT), such as a queue or stack in a programming lab
   b. Use a previously written ADT from the Python's collection application programmer interface (API)
   c. Incorporate inheritance in a project that uses ADTs
   d. Provide a client program that tests and demonstrates the correct behavior of the ADT
7. The proper use of modules and classes, in conjunction with inheritance and other advanced techniques learned in previous labs
   a. Create the proper set of methods within a class that enables an object to be copied deeply
   b. Utilize inheritance to reinforce the segregation of data into base- and derived-level behavior
   c. Utilize at least one other lab concept, such as binary logic or multi-dimensional arrays, to further improve integration of intermediate concepts
   d. Separate I/O and data storage in advanced programs
8. Building projects that make effective use of Python's duck typing
   a. Demonstrate the difference between deriving from a base class and specializing using duck typing
   b. Practice using a duck typed method or class by invoking the method with different client types and test that the duck typing is working as expected
   c. Use duck typing to exercise some aspect of ADTs, such as specializing a generic ADT to make its behavior specific to an assigned project specification
   d. Employ debugging techniques to solve problems that arise when designing with duck typing
9. Exception handling and file I/O in programming
   a. Write a class that has methods which use exception handling to report errors to the client
   b. Write a test client that uses a try/except template to detect exceptions
   c. Implement an algorithm or simulation that reads from and/or writes to external files
   d. Demonstrate various ways that exceptions are handled and reported besides exception objects being raised and excepted

Special Facilities and/or Equipment

1. Access to a computer laboratory with Python interpreters.
2. Website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
3. When taught via Foothill Global Access on the internet, the college will provide a fully functional and maintained course management system through which the instructor and students can interact.
4. When taught via Foothill Global Access on the internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

Methods of Evaluation may include but are not limited to the following:

Tests and quizzes
Written laboratory assignments which include source code, sample runs, and documentation
Final examination

Method(s) of Instruction

Methods of Instruction may include but are not limited to the following:

Lectures which include motivation for syntax and use of the Python language and OOP concepts, example programs, and analysis of these programs
Online labs (for all sections, including those meeting face-to-face/on-campus), consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet environment. Here, the students will review the specification of each programming assignment and submit their completed lab work
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members
Detailed review of programming assignments which includes model solutions and specific comments on the student submissions

In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing, and analyzing programs
When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices

Representative Text(s) and Other Materials

Ramalho, Luciano. Fluent Python. 2022.

Downey, Allen. Think Python, 2nd ed.. 2020.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

1. Reading
    a. Textbook assigned reading averaging 30 pages per week
    b. Reading the supplied handouts and modules averaging 10 pages per week
    c. Reading online resources as directed by instructor though links pertinent to programming
    d. Reading library and reference material directed by instructor through course handouts
2. Writing
    a. Writing technical prose documentation that supports and describes the programs that are submitted for grades

Discipline(s)

Computer Science