C S 3M: INTERMEDIATE ALGORITHM & DATA STRUCTURE METHODOLOGIES IN PYTHON

Foothill College Course Outline of Record

Heading	Value
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Prerequisite:	C S 3A.
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 language 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.
- The successful student will be able to analyze the time complexity of a variety of algorithms and data structure access techniques and choose the best algorithm and/or data structure for the project at hand.

Description

Systematic treatment of intermediate data structures, algorithm analysis and abstract data types in the Python programming language intended for Computer Science transfer majors. Coding topics include large program software engineering design, multi-dimensional arrays, string processing, primitives, compound types, and allocation of instance and static data. Concept topics include dynamic memory, inheritance, polymorphism, hierarchies, recursion, linked-lists, stacks, queues, trees and hash tables.

Course Objectives

The student will be able to:
A. Use instance members, static members, and dynamic data structure allocation as appropriate in object-oriented Python class design.
B. Analyze and demonstrate the use of dynamic and static multi-dimensional arrays in Python.
C. Design, implement, and test Python programs that use object-orientation and class inheritance as a key ingredient to good software design, and explain why sub-classing is an example of the "is-a" relationship.
D. Describe the difference between deep copies and shallow copies in Python, and write programs that effectively handle deep memory.
E. Demonstrate a working knowledge of data abstraction through various data types, data structures and built-in Python classes, and choose the appropriate data structure to model a given problem.
F. Give examples of the proper use of recursion and describe when an iterative solution is more efficient.
G. Explain what abstract Python classes and pure virtual functions are and how they are used.
H. Describe declaration models for runtime storage allocation and garbage collection, and demonstrate how files are written-to and read-from in C++.
I. Use an external Python library to write an efficient and portable application program.
J. Design, implement, test, and debug intermediate-level Python programs that use each of the following fundamental programming constructs: user interaction and communication, string processing, numeric computation, simple I/O, arrays and the Python built-in classes.
K. Analyze the basic algorithms of a general tree ADT.
L. Use object-oriented programming (OOP) to create alternative implementations of binary search trees in Python, and verify or compare the logN behavior of each.
M. Analyze, classify and measure the main non-NlogN sorts and write a clear report of the results.
N. 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

A. The proper use of class members and methods
1. Primitive vs. compound types
2. Types as collection of values (data members) and operations (method members)
3. Type checking, duck typing, and incompatibility in classes
4. When to use instance members and methods
5. When to use static members and methods
6. Use of the "self" object and optional arguments
B. Multi-dimensional arrays
1. Fixed-size 2-D arrays
2. Dynamically allocated "ragged" 2-D arrays
3. Instantiation of objects in multi-dimensional arrays
C. Inheritance in software design
1. The "is a" relationship
2. Base classes
3. Derived classes (subclasses) and class hierarchy
4. Derived class constructors
5. Member method overriding
6. Simulating private instance variables in Python
7. Encapsulation and polymorphism
8. Separation of behavior and implementation and other attributes of good software design
D. Deep vs. shallow copies of objects
1. Instantiation of member objects in constructors
2. Cloning objects
3. Deep copy cloning and its uses
4. Shallow copy cloning and its uses
E. Topics in Abstract Data Types (ADTs)
1. The vector ADT
2. The linked-list ADT
3. The stack and queue ADT
4. Priority queues and heaps in Python
5. Implementing ADTs through inheritance
6. Using existing ADTs
7. Iterators
F. Recursion
1. Base case vs. general case

2. Divide-and-conquer strategies
3. Analysis of recursive solutions which are not appropriate due to exponential time complexity vs. iterative polynomial complexity
4. Recursive backtracking
G. Abstract classes and interfaces
1. Abstract classes
2. Abstract methods
3. Using the Python module abc
4. Create an interface using an abstract class and an abstract method
H. Storage allocation methods
1. Run time binding and storage management of activation records
2. Consequences of the values/pointer mechanism when passing parameters
3. Strong type-checking and run-time vs. syntax error detection
4. Effect of declaration strategy on binding, visibility and lifetime of variables
5. Effect of declaration strategy on scope and persistence of variables
I. Essential examples and assignment areas
1. String/text processing
2. Numeric computation
3. User interaction
4. Multi-class projects and compound data types
5. Inheritance-based projects
J. General trees
1. Tree nodes, roots, leaves, children and siblings
2. Binary node implementation of a general tree
3. Insertion and deletion in general trees
4. Traversal with recursion
K. Searching and Binary Search Trees (BSTs)
1. Ordering condition and structure condition
2. OOP (object-oriented-programming) implementation
3. Time complexity consequence of the divide-and-conquer algorithm in of BSTs
L. NlogN sorts
1. Merge sort
2. Heap sort
3. Quicksort
M. Applications used throughout course in selected areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet

Lab Content

A. Exploring advanced array constructs in class design
1. Gain experience in effectively using single and multi-dimensional arrays as class members.
2. Apply nested loops to process multi-dimensional arrays.
3. Use the IDE to debug errors in multi-dimensional arrays.
4. Solve problems using fixed-size and dynamic sized arrays, as appropriate.
B. Building a program that uses class inheritance to demonstrate how re-use is handled in OOP
1. Create a project that contains at least one class intended to be used as a base class.
2. Derive (sub-class) one or more classes from the base class.
3. Use function chaining to avoid code duplication between base classes and derived classes.
4. Use method overriding in a class design.
C. Incorporating basic queue and stack abstractions in programming projects
1. Implement a fundamental queue and stack abstract data type (ADT) in a programming lab.
2. Use a previously written ADT from the programming language's application programmer interface (API).
3. Incorporate inheritance in a project that uses ADTs.
4. Provide a client program that tests and demonstrates the correct behavior of the ADT.
D. Using linked-list ADTs to optimize for size-varying or space-sensitive data types
1. Demonstrate the ability to use programming language-supplied linked-list structures in a problem that is not easily solved using fixed-size ADTs, such as arrays.
2. Try different sized data for the linked-list and demonstrate that it handles growth properly.
E. Demonstrating competence with binary search trees
1. Implement a binary search tree (BST) from scratch, or make significant assigned adjustments to an existing BST data structure supplied by the instructor.
2. Use recursion as appropriate for some of the BST methods.
F. Incorporating hash tables into programs
1. Produce a lab that creates or modifies a hash table and hashing function.
2. Write a client that tests out the hash table on various data.
3. Using a large data set, demonstrate that near-constant time access is produced by the hashing function and hash table.
4. Supply runs and report results with varying sized data sets.
G. Analysis of a single sort algorithm
1. Implement a single sort algorithm as directed by the instructor.
2. Experiment with coding adjustments to try to improve the performance.
3. Compare the known time complexity of that algorithm with what you observe using increasingly larger data sets.
4. Attempt to explain any discrepancies in the expected vs. observed growth rate of the sort algorithm.
H. Analysis of multiple sort algorithms
1. Implement multiple sort algorithms, at least two of which involve shell sort and quicksort.
2. Experiment with coding adjustments to try to improve the performance on any one of them to see if you can beat the fastest of the algorithms.
3. Time the algorithms on very small to very large data sets.
4. Report on which algorithms work best on small sets, and which on large sets.

Special Facilities and/or Equipment

A. Access to a computer laboratory with Python interpreters.
B. 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.
C. 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.
D. 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:

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

Method(s) of Instruction

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

A. Lectures, which include motivation for syntax and use of the Python language and OOP concepts, example programs, and analysis of these programs.
B. 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, submit their completed lab work and get feedback from the instructor.
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.
C. Detailed review of programming assignments, which includes model solutions and specific comments on the student submissions.
D. In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs.
E. 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

Baka, Benjamin. Python Data Structures and Algorithms: Improve Application Performance with Graphs, Stacks, and Queues. Packt Publishing, 2017.
Miller and Ranum. Problem Solving with Algorithms and Data Structures Using Python. 2nd ed. Franklin Beedle, 2011.

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

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

Discipline(s)

Computer Science