College of San Mateo
Official Course Outline

1. **COURSE ID:** CIS 501 **TITLE:** (CS2) Data Structures: Python
   Units: 4.0 units **Hours/Semester:** 48.0-54.0 Lecture hours; 48.0-54.0 Lab hours; and 96.0-108.0 Homework hours
   Method of Grading: Grade Option (Letter Grade or Pass/No Pass)
   Prerequisite: CIS 117

2. **COURSE DESIGNATION:**
   Degree Credit
   Transfer credit: CSU; UC

3. **COURSE DESCRIPTIONS:**
   Catalog Description:
   Abstract data type implementation and usage techniques for computer science majors and computer professionals. Object-oriented approach to a variety of abstract data types including: lists, stacks, queues, priority queues, trees, maps and graphs. Also includes advanced sorting and searching topics, and algorithmic analysis using Big-O notation. This course conforms to the ACM CS2 standards. A materials fee as shown in the Schedule of Classes is payable upon registration.

4. **STUDENT LEARNING OUTCOME(S) (SLO'S):**
   Upon successful completion of this course, a student will meet the following outcomes:
   1. Apply object-oriented techniques to the implementation of abstract data types.
   2. Characterize an algorithm using Big-O notation.
   3. Select an appropriate data sort, based on characteristics of data to be sorted together with frequency of sort.
   4. Employ algorithmic patterns to array, linked and recursive structures.
   5. Construct reliable, robust solutions to problems involving the storage, retrieval and update of large quantities of data.

5. **SPECIFIC INSTRUCTIONAL OBJECTIVES:**
   Upon successful completion of this course, a student will be able to:
   1. Apply object-oriented techniques to the implementation of abstract data types.
   2. Determine the appropriate abstract data type to utilize for storing a quantity of data, based on the characteristics of the application.
   3. Evaluate the trade-offs between static and dynamic implementations of an ADT, based on hardware speed/memory specifics.
   4. Characterize an algorithm using Big-O notation.
   5. Implement abstract data types using both static and dynamic data storage techniques.
   6. Select an appropriate data sort, based on characteristics of data to be sorted together with frequency of sort.
   7. Employ algorithmic patterns to array, linked and recursive structures.
   8. Construct reliable, robust solutions to problems involving the storage, retrieval and update of large quantities of data.

6. **COURSE CONTENT:**
   Lecture Content:
   Review
   1. Object-oriented design
   2. Basic algorithm design
   3. Class implementation/object usage
   4. Professional and ethical issues

   Algorithmic analysis
   1. Identifying differences among best, average, and worst case behaviors
   2. Big-O notation
   3. Standard complexity classes
   4. Time/space/complexity tradeoffs in algorithms

   Fundamental abstract data types (array and dynamic implementations)
   1. Random access linear structures (lists/sets)
   2. Stacks
   3. Queues
   4. Trees
   5. Graphs

6. Heaps
7. Hash tables

Fundamental computing algorithms
1. Insertion/Deletion/Search
2. Traversals
3. Paths
4. Update

Advanced Recursion
1. Recursive mathematical functions
2. Recursive procedures
3. Divide-and-conquer strategies using recursion, recursive backtracking
4. Recursion on trees and graphs

Searching and sorting
1. Linear vs binary search
2. Quadratic sorts
3. Recursive sorts
4. Criteria for choosing the right data structure

Software engineering
1. Software project management
2. Building a medium-to-large sized system
3. Working teams

Lab Content:

Students complete small programming exercises which relate to the current class lecture. Lab exercises involve:
1. writing a method which exhibits a particular Big-O measurement
2. adding methods to an object-oriented data type which has been covered in lecture
3. examining an ADT provided by the Python API and using it in a small application
4. coding an application which utilizes an object-oriented type which was covered in lecture

The data structures (in sequence) which are used for lab exercises are:
1. simple array-based collection class (bags, lists, sorted lists, sets)
2. linked list class (linked bag, list, sorted list, set)
3. stack (array based, linked)
4. queue (array based, linked)
5. binary tree (linked, recursive exercise)
6. binary search tree (linked implementation)
7. heap (array-based implementation)
8. recursive sorts (quicksort partition)
9. hash tables
10. graph

7. **REPRESENTATIVE METHODS OF INSTRUCTION:**

Typical methods of instruction may include:
- A. Lecture
- B. Lab
- C. Activity
- D. Directed Study
- E. Discussion
- F. Individualized Instruction
- G. Observation and Demonstration
- H. Other (Specify): The course includes the following instructional methods as appropriate: Lecture, to introduce new topics; “Models” for problem-solving techniques; In class group problem solving, each person contributing a potential "next step"; Student participation in short in-class projects (in teacher-organized small groups) ; Q/A sessions in which the students provide both the question AND the answers; Small groups of students working together to solve significant programming assignments.

8. **REPRESENTATIVE ASSIGNMENTS**

Representative assignments in this course may include, but are not limited to the following:

Writing Assignments:

Project 1 Simple ADT implementation (review of class implementation including Object overrides, static class members and object usage)

Project 2 List/Set collection class implementation and application

Project 3 Linked list class implementation and application

Project 4 Stack/Queue class implementation and application

Project 5 Binary Search tree class implementation and application

Project 6 Priority queue (heap)class implementation and application

Project 7 Hash Table class implementation and application
Project 8 Graph class implementation and application

Reading Assignments:

Text book reading, 2 chapters per week and related project example.

9. **REPRESENTATIVE METHODS OF EVALUATION**

Representative methods of evaluation may include:

A. Class Participation
B. Exams/Tests
C. Group Projects
D. Lab Activities
E. Projects
F. Quizzes
G. Written examination
H. Weekly textbook readings, textbook exercises, and bi-weekly programming projects* comprise the majority of the out-of-class assignments. One or more of the programming projects will be a small-group project to provide experience in a realistic program development environment, in order to improve technical communication skills and simulate a more realistic programming environment. Student participation during collaborative activities (group project work and class discussion) will be considered. Midterm and final exams, constituted of short answer and general problem solving exercises will be administered. ** Although programming projects will vary from semester to semester, each semester project will be assigned which include implementation and/or usage of the priority queue (heap), hash table and graph to satisfy UCB articulation requirements.

10. **REPRESENTATIVE TEXT(S):**

Possible textbooks include:

A. Michael H. Goldwasser, Michael T. Goodrich, and Roberto Tamassia. *Data Structures and Algorithms in Python*, 1 ed. Wiley, 2013
B. Kent D. Lee, Steve Hubbard. *Data Structures and Algorithms with Python*, ed. Springer, 2015
C. Hemant Jain. *Problem Solving with Algorithms and Data Structures Using Python*, ed. Independently published, 2019

Origination Date: October 2019
Curriculum Committee Approval Date: December 2019
Effective Term: Fall 2020
Course Originator: Kamran Eftekhari