8/23 Objectives

- Be able to describe the need that originally motivated the discipline of computer science, and some of the repercussions of that need which make it a valid field of study
- Be able to define the terms *abstraction*, *encapsulation*, *interface*, *information hiding*, *client*, *mutable*, *immutable, keyed*
- Be able to describe what abstraction means to the software professional
- Be able to describe the primitive and built-in data types of Python, and to answer questions about the operations on them
- Be able to describe what is meant by a keyed collection and to say what data structures in Python are keyed collections

8/25 Objectives

- Be able to specify the uses and effects of the Python operators and to use them in natural ways, taking into account their precedences
- Be able to describe what is meant by a keyed collection and to say what data structures in Python are keyed collections
- Be able to specify the uses and effects of the built-in Python data structures (collections) and to use them in natural ways
- Be able to use Python subscripting and slicing
- Be able to take advantage of Python tuple assignment
- Be able to classify Python data types as mutable or immutable, and to demonstrate what is meant by those terms

8/30 Objectives

- Be able to use the while statement and the for statement to form loops in a Python program
- Be able to construct list and set comprehensions
- Be able to define the term *exception*, and to construct try:except: blocks in Python
- Be able to match a raise statement with its corresponding except: block
- Be able to define a Python function and to use function calls
- Be able to translate a pseudo-code function into Python
- Be able to construct a Python class definition
- Be able to create operator functions and to use the corresponding operators
- Be able to define the term scope, and to list some different kinds of scope in Python

9/1 Objectives

- Be able to translate a pseudo-code function definition into Python
- Be able to construct a Python class definition
- Be able to create operator functions for a class, and to use the corresponding operators
- Be able to define the term *scope*, and to list some different kinds of scopes in Python
- Be able draw the tree form of an expression involving binary operators
- Be able to convert expressions between any two of the standard notations: infix, prefix, and postfix
- Be able to describe the public interface to the primary data structures: Stack, Queue, and Deque
- Be able to use Stacks and Queues in client code

9/6 Objectives

- Be able to convert expressions between any two of the standard notations: infix, prefix, and postfix
- Be able to describe the public interface to the primary data structures: Stack, Queue, and Deque
- Be able to use Stacks and Queues in client code (Python)
- Be able to describe the postfix evaluation algorithm
- Be able to code in Python both versions of the list implementation of a Stack (“has a” and “is a”)
- Be able to describe the class SingleNode and how it would be used to create a singly linked list
- Be able to describe the class Node and the public interface of class LinkedList
- Be able to describe the abstract data type OrderedList

9/8 Objectives

- Be able to describe the class Node and the public interface of class LinkedList
- Be able to write the addFirst(item), removeFirst, insertAfter(node), insertBefore(node), search(item) and pop(node) methods of class LinkedList
- Be able to describe the abstract data type OrderedList
- Be able to implement OrderedList as a derived class of LinkedList
- Be able to directly implement any of the classes Stack, Queue, Deque, and UnorderedList using doubly linked lists

9/13 Objectives

- Be able to discuss the issue of when one algorithm is better than another
- Be able to identify the “size of the problem” for a given problem
- Be able to determine when a problem’s size can be thought of in terms of the value of a single integer parameter
- Be able to identify “significant operations” in an algorithm
- Be able to say how a time function T(n) is “measured” by another, simpler, “ideal function” f(n)
- Be able to precisely define the statement “T(n) is O(f(n))
- Be able to define what is meant by the constant function, the linear function, the quadratic function, the cubic function, and in general what is meant by a polynomial function, as ideal functions
- Be able to define what is meant by the log function, the linear log function, an exponential function, and the factorial function, as ideal functions
- Be able to identify the shape
- Given an algorithm or Python function, be able to determine its complexity in “Big-O” terms

9/15 Objectives

- Be able to describe how logarithms with different bases relate to each other, from a complexity standpoint
- Be able to describe how exponentials with different bases relate to each other, from a complexity standpoint
- Given an algorithm or Python function, be able to determine its complexity in "Big-O" terms
- Be able to say when an algorithm which is "inferior" from a time complexity standpoint might be a better choice for a particular application
- Be able to write code to time the execution of a function or block of code
- Be able to say what capabilities of the built-in collection types list and set are O(1) and which are O(n).

9/20 Objectives

- Be able to define: recursion, direct recursion, indirect recursion, base case, system stack, stack frame, stack overflow
- Be able to describe the conditions under which a recursive call will ultimately terminate
- Be able to argue why recursion can be a useful thing
- Be able to reformulate an algorithm that loops as a recursive algorithm
- Be able to write a technical recursive algorithm that is meant to be used by a simpler non-recursive algorithm
- Be able to code a recursive algorithm in Python
- Be able to describe the three storage classes

9/22 Objectives

- Be able to classify algorithm analysis by best-case, worst-case, and average-case
- Be able to suggest a graphical way of analyzing recursive algorithms
- Be able to say what the Big-O analyses of binary search and sequential search are, and to justify those analyses
- Be able to define the algorithm classifications known as precise, heuristic, greedy, and brute-force
- Be able to describe a situation in which a recursive algorithm is too slow to be viable
- Be able to describe what is meant by “memoizing” a recursive algorithm
- Be able to describe how a memoized algorithm may be converted into a faster non-recursive dynamic programming algorithm
- Be able to code the greedy change-making algorithm

9/27 Objectives

- Be able to explain what dynamic programming is, and how it can improve on a recursive solution
- Be able to compare and contrast the sequential and binary search strategies
- Be able to define the terms hashing, hash table, perfect hash function, uniform key distribution
- Be able to comment on how hash tables achieve a constant-time (O(1)) performance for the search problem
- Be able to talk about various scrambling strategies for hash functions
- Be able to discuss hash table sizes and how they relate to key distributions

9/29 Objectives

- Be able to define the terms hashing, hash table, perfect hash function, uniform key distribution
- Be able to comment on how hash tables achieve a constant-time (O(1)) performance for the search problem
- Be able to talk about various scrambling strategies for hash functions
- Be able to discuss hash table sizes and how they relate to key distributions
- Be able to define what a probe is, and to discuss probe sequences
- Be able to define and illustrate linear open addressing
- Be able to define quadratic open addressing
- Be able to do an analysis of hash table efficiency, based on load factor λ, of a linear open addressing hash table and of a separate chaining hash table
- Be able to discuss how *dict* and *set* achieve their ability to store any Python object