Objectives:
What you should learn in CSCI141

Ben K Steele
email: bks@cs.rit.edu

The course schedule shows that this course has many topics. If we move up to a higher level of *abstraction*, we will see that the topics fit into a more general framework. This level provides a roadmap by which to outline the objectives for what you should learn in this course.

The words and phrases in *italics* are important terms and concepts in this course. By the end, you should be able to define and explain these terms and concepts with examples.

1 Programming with scalar data types using Turtle Graphics

By the end of this unit (weeks 1-4), you should be able to:

1. Write functions and procedures that use Python’s turtle graphics module.
2. Design and implement programs composed of multiple functions and procedures that deal with simple *scalar data types* (i.e. *integers* and *real numbers*).
3. Write functions that get user input and convert it to the appropriate data type, process it, and print results.
4. Write arithmetic expressions to calculate results, including using *modular arithmetic*.
5. Apply *recursion* to solving problems involving repetition of steps.
6. Apply *iteration* to solving problems involving repetition of steps.
7. Write *relational expressions* whose *Boolean* values determine the truthfulness of program state.
8. Write *conditional statements* that decide between multiple courses of execution.
9. Trace the execution of recursive and iterative Python code in multiple ways.
10. Create problem examples and implement test functions that realize the examples.
11. Begin to formulate, develop and apply your own, personal computational problem-solving process to solve graphical problems.

2 Algorithms and Data Structures with linear data types

By the end of this unit (weeks 5-11), you should be able to:

1. Write functions and programs that process *character strings* in Python.
2. Trace the execution of recursive and iterative functions on *data collections*.
3. Write programs that open and close *files* and read/write file-based data.
4. Identify the need for a *compound data structure* in a given problem.
5. Apply Python *lists*(*arrays*) and *tuples* to solve problems involving *data sequences*.
6. Compare and contrast the operation of several sorting algorithms.
7. Explain and draw pictures of the operation of several improved sort algorithms.
8. Implement one or more sort algorithm variations that strive for greater efficiency.
9. Identify and describe the complexity of several sort algorithms given various test scenarios.
10. Write programs that treat lists as a stack data structure.
11. Write programs that treat lists as a queue data structure.
12. Understand, explain, and apply the use of the linked structures to a given problem.
13. Understand, explain, and apply the use of the stack data structure to a given problem.
14. Understand, explain, and apply the use of the queue data structure to a given problem.
15. Draw pictures showing the conceptual structure of linked structures, stacks, queues, strings and files as they would look at some point in the execution of a program.
16. Apply the Python class construct to solve problems involving compound data.
17. Create examples and test cases for validating implementations involving compound data such as strings, lists, stacks and queues.
18. Analyze and explain the performance of algorithms that operate on strings, files, lists, stacks and queues.
19. Distinguish and discuss the similarities and differences between stacks and queues.
20. Identify and describe the complexity of algorithms that operate on strings, files, lists, stacks and queues, given various test scenarios.
21. Write algorithms using a simplified language known as pseudocode.

3 Algorithms and Data Structures with non-linear data types

By the end of this unit (weeks 12-14), you should be able to:
1. Design and implement solutions to problems that require tree structured data.
2. Trace execution and processing of code that constructs tree structure instances as well as code that traverses those structures.
3. Apply Python to the design and implementation of tree structures.
4. Draw pictures showing the conceptual structure of trees as they would look at some point in the execution of a program.
5. Understand and explain how a tree structure can improve performance when compared to a list data structure applied to maintain list item priorities.
6. Draw picture sequences showing the evolving structure of a tree as the tree would change during the execution of a program.
7. Identify and describe the complexity of algorithms that operate on trees given various problem and test scenarios.
8. Apply Python lists to the design and implementation of an associative array data structure for fast, keyed access to large collections of compound data.
4 Overall Objectives of the Course

There are some over-arching objectives.

By the end of this course, you should be able to:

1. Analyze problems requiring a computing solution.
2. Explain a problem’s data inputs, processing and outputs.
3. Assess, propose, and explain choices for the application of algorithms and data structures to solve a computing problem.
4. Write programs using appropriate algorithms and data structures to solve relatively simple computing problems.
5. Analyze the complexity of a given algorithm.
6. Trace the execution of a given program or function.
7. Create examples of inputs and expected outputs for a given problem.
8. Create test cases and test code from examples for a given problem.
9. Draw pictures of code in various states of execution.
10. Draw pictures of data structures as they would look at various states of program execution.
11. Perform basic debugging on programs.