CSI33 Data Structures

Sharon Persinger

Fall 2019

August 28, 2019 Day 1

Introduction

- Review of syllabus
- Course webpage at https://fsw01.bcc.cuny.edu/sharon.persinger/

Functional abstraction

- What is abstraction?
- Functional abstraction: Decompose the tasks of a program into functions in order to hide the details of programming.
- A programming function performs a task. The function might take some input values and might compute and return some value or values.
- Write a function to accomplish a specific task given by a function specification.

Examples of function specifications

- The file stats.py has specifications for several functions in **precondition** and **postcondition** format.
- The function **precondition** tells what is true before the function is called. Most often, these are assumptions about the arguments of the function.
- The function **postcondition** tells what is true after the function finishes executing. Frequently, the postcondition tells the significance of the return value of the function. A postcondition can also describe the effect of a function on some argument.
- Functions should not produce undocumented side effects. A function should not print a value unless the postcondition says that is what the function does. A function should not change an argument unless the postcondition says that is what the function does.

Review examples from stats.py

Preconditions and function code

- Write definitions for some of the functions for stats.py

- How do you think about the preconditions when writing the function?

Enforcing preconditions

- What should your function do if the precondition is not true?

- One good way for your function to actually recognize and enforce preconditions:
  - Raise an exception if the precondition isn't true.

Intro to analysis of algorithms

- An algorithm is a step-by-step procedure for solving a problem. Usually there are many different algorithms to solve one problem. How do we compare algorithms? What makes one algorithm better than another?
  - Ideas?

- Problem: Find the index of a certain target number in a list of numbers. If the number isn't in the list, return an impossible index, -1.
- Precondition and postcondition?

Time analysis of algorithms

- We can look at the actual running time of a program.
- We can also analyze how many steps are needed to carry out an algorithm. We will be interested in how the number of steps I the algorithm increases as the size of the problem increases.

Two familiar approaches to searching

- Linear Search
  - Compare your target number to the first number in the list, then to the second, and so on, until you find the target number. Return the index where you found it. If you examine every number in the list and don't find the number, return -1.
- Binary search
  - This search procedure requires that the list of data is in sorted order. Find the index of the item in the middle of the list. If the target is equal to this middle item, return that index. If the target is less than the middle item, continue the search procedure, looking only at the first half of the list. If the target is greater than the middle item, continue the search procedure, looking only at the second half of the list.

Worst case analysis

- For searching problems, the worst case happens when the target is not in the list.
- For linear search on a list of 100 items, how many items must be examined? On a list of 10,000 items? On a list of 1,000,000 items?
- For binary search on a list of 100 items, how many items must be examined? On a list of 10,000 items? On a list of 1,000,000 items?

Run the search programs and collect some data