CSI33 Data Structures

Sharon Persinger

Fall 2019

Rational ADT

- An ADT for rational numbers p/q, p and q integers, q not zero.
- Functions – overload all the arithmetic operators and comparison operators

Unit testing with the Python unittest module

- See example test_Rational.py

Programming assignment 1: Complete the Dataset ADT and test it

- Use the unit-testing module
- Test constructor, add method – for one element, two elements, three elements, all the other methods

Python lists

- Python list - list1, list2– a sequence of data elements with operations
- Access with []: list1[2]
- Concatenate with +: list1 + list2
- len(list1) function
- Slicing - list1[2:5}
- list1.append(x)
- list1.index(x[, start[, end]]),
- list1.extend(iterable)
- list1.insert(i, x)
- list1.remove(x)
- list1.pop([i])
- list1.count(x)
- list1.sort(key=None, reverse=False)
- list1.reverse()
- list1.copy()
- list1.clear()

Sequential Collection: Deck of cards

```
# Deck.py

from random import randrange
from Card import Card

class Deck(object):

    #------------------------------------------------------------

    def __init__(self):

        """post: Creates a 52 card deck in standard order"""

    #------------------------------------------------------------

    def size(self):

        """Cards left
        post: Returns the number of cards in self"""

        return

    #------------------------------------------------------------

    def deal(self):

        """Deal a single card
        pre:  self.size() > 0
        post: Returns the next card in self, and removes it from self."""

        return

    #------------------------------------------------------------

    def shuffle(self):

        """Shuffles the deck
        post: randomizes the order of cards in self"""
```

Bridge Hand ADT

- We want to be able to represent a bridge hand in sorted order. Bridge hands are arranged by suit in decreasing order, and then the cards in each suit are arranged in decreasing order.
- First, modify and improve the Card ADT.
  - In bridge, Aces are high.
  - Overload the comparison operations for the Card class:
    - Overload == by defining __eq__(self, other)
    - Overload < by defining __lt(self, other)
    - Overload != by defining __ne__(self, other)
    - Overload <= by defining __le__(self, other)

Look at the Hand specification

```
# Hand.pyclass Hand(object):
    """A labeled collection of cards that can be sorted"""
    #-------------------------------------------------------------
    def __init__(self, label=""):
        """Create an empty collection with the given label."""
    #-------------------------------------------------------------
    def add(self, card):
        """ Add card to the hand """
    #-------------------------------------------------------------
    def sort(self):
        """ Arrange the cards in descending bridge order."""

    #-------------------------------------------------------------
    def dump(self):
        """ Print out contents of the Hand."""
```

Somethings are easy for Hand because Card class takes care of the work.

- We overloaded < so we can use the Python sorting function. It is very efficient - $\Theta(n\ log\ n)$. That doesn't really matter since bridge hands have only 13 cards.
- We can print out a Hand, since Card has a string representation function.