Programming Fundamentals Using Python

2018

Problem Set 9

Most recent updated: July 14, 2018

Objectives

1. Stack
2. Queue
3. Deque

Note: Solve the programming problems listed using your favorite text editor. Make sure you save your programs in files with suitably chosen names, **and try as much as possible to write your code with good style (see the style guide for python code)**. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems: Cohort sessions

1. *Stack* Create a class for Stack Abstract Data Type (ADT). The ADT should support the following:

   (a) `push(item)` : to push item into the stack.

   (b) `pop()`: to pop the item at the top of the stack. The function should return the item at the top of the stack.

   (c) `top()`: to return the item at the top of the stack. The item, however, is not popped out, so it remains in the stack.

   (d) `empty()`: to check whether the stack is empty or not. It returns a boolean value.

   (e) `size()`: to return the size of the stack.

   Design your own test cases.

2. *Stack: Postfix* Create a class called `Postfix` that stores a postfix expression and is able to evaluate such expression. The postfix expression is initialized during object instantiation. It should have at least one method called `evaluate()` which evaluates the current postfix expression and returns the result.

```
>>> p = Postfix('7 8 + 3 2 + /')
>>> assert p.evaluate() == 3
>>> p = Postfix('1 2 + 3 *')
>>> assert p.evaluate() == 9
```

3. *Queue* Create a class for Queue Abstract Data Type (ADT). The ADT should support the following:

   (a) `put(item)` : to enqueue the item into the queue.

   (b) `get()`: to remove the item at the beginning of the queue. The function should return the item.

   (c) `empty()`: to check whether the queue is empty or not. It returns a boolean value.

   (d) `qsize()`: to return the size of the queue.

   Design your own test cases.

4. *Queue: Radix Sorting Machine* Implement a radix sorting machine. A radix sort for base 10 integers is a mechanical sorting technique that utilizes a collection of bins, one main bin and 10 digit bins. Each bin acts like a queue and maintains its values in the order that they arrive. The algorithm begins by placing each number in the main bin. Then

it considers each value digit by digit. The first value is removed and placed in a digit bin corresponding to the digit being considered. For example, if the ones digit is being considered, 534 is placed in digit bin 4 and 667 is placed in digit bin 7. Once all the values are placed in the corresponding digit bins, the values are collected from bin 0 to bin 9 and placed back in the main bin. The process continues with the tens digit, the hundreds, and so on. After the last digit is processed, the main bin contains the values in order.

```
>>> items = [154, 26, 293, 17, 377, 31, 444, 55, 620, 65]
>>> result = radix_sort(items)
>>> resultlist = [result.get() for x in range(result.qsize())]
>>> assert resultlist == [17, 26, 31, 55, 65, 154, 293, 377, 444,
    620]
```

5. *Deque* Create a class for Deque Abstract Data Type (ADT). The ADT should support the following:

   (a) `append(item)` : to add item from the end (right).

   (b) `append_left(item)` : to add item from the front (left).

   (c) `pop()`: to remove the item at the end (right). The function should return the item.

   (d) `pop_left()`: to remove the item at the front (left). The function should return the item.

   (e) `__len__()`: to return the number of elements in deque.

   Design your own test cases.

6. em Deque: Palindrome Redo Palindrome problem using Deque. Design your own test cases.

End of Problem Set 9.