Advanced Topics in Comp Sci — Activity—The `Queue` Class

ASSIGNMENT OVERVIEW

In this assignment you'll create a Python implementation of the abstract data type "queue."

This assignment is worth 20 points and is due on the *crashwhite.polytechnic.org* server at 23:59:59 on the date given in class.

BACKGROUND

A "queue," like a "stack," is an ordered collection of items. Items can be "queued" onto the end of the queue, in order, with items queued later being placed behind items that have arrived earlier. When items are removed from the queue they are "dequeued" from the front in a "first come, first served" fashion. The item at the front of the queue can be "peeked" at (identified without removing it from the queue), and the "size" of the queue can be identified at any time. Also, the queue can be identified as being "empty" (a size of 0).

Real-life examples of queues are can be found everywhere: Waiting in line to get into a movie, lining up to be served at a fast-food restaurant, etc. Because of the way items are *queued* and *dequeued* in a queue, it is sometimes called a *First In-First Out* (FIFO) structure.

PROGRAM SPECIFICATION

Write a Python class `Queue` that implements this *abstract data type*. The `Queue` class will include:

a. a constructor which creates an empty queue
b. the `.enqueue(item)` method to place an item onto the back of the queue
c. the `.dequeue()` method to remove an item from the front of the queue, and return that removed item as a result
d. the `.peek()` method which returns the value of the item at the front of the queue without removing it
e. the `.size()` method which returns the number of items in the queue
f. the `.is_empty()` method which returns `True` or `False`, depending on the state of the queue

DELIVERABLES

atds.py

This single file ("atds" = "Advanced Topics Data Structures") will now include implementations of both the `Stack` and `Queue` classes.

To submit your assignment for grading, copy your file to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before the deadline.

ASSIGNMENT NOTES

- Use Python's `list` data structure as a foundation for implementing the `Queue` class. Python's `list` methods adapt very nicely to the methods we're implementing for the `Queue` class.
- Should the "head" of the queue be at index 0 or at index -1? Your implementation might use either strategy, but one will be more efficient than the other.
- You have two ways of testing your development of the `Queue` class.
    - Begin writing the class, and after adding each new feature, start up Python in interactive mode, and issue commands one at a time to interact with your `Queue` class.

    - Write a full tester program of your own—a separate Python main program—that imports your `Queue` class, tries to construct a `Queue` object, and tries to interact with it.
- Ultimately, once you have upload your completed `atds.py` file, it will be analyzed with a tester similar to the one shown at the end of this document.

GETTING STARTED

1. Create a backup copy of your original `atds.py` file and archive it someplace safe.

2. Add the `Queue` class to your `atds.py` file and implement the constructor and one or two methods to start out.

3. Use Python in interactive mode to import the `atds` package, and then try to construct a `Queue` object. As you add methods to the `Queue` class, test them in interactive mode.

4. As the development of the `Queue` class proceeds, consider creating a full `queue_tester.py`, a Python main program that will run your `Queue` objects through a series of tests. It's more efficient than having to interactively create a `Queue` object and manipulate it every time you add a new feature.

5. When your program is completed (but before the deadline), copy `atds.py` to the server as indicated above.

EXTENSIONS

1. Write a program called `hot_potato.py` that takes a list of people, places them in a queue, and has them pass a virtual "hot potato" around until everyone but one person has been removed from the list. Refer to your textbook for strategies on how to implement this game.

2. Modify `hot_potato.py` so that each pass of the potato requires a "tick" of time. The `tick()` method will have the potato cool off a random amount. When the potato has completely cooled, the person holding it loses.

3. Write a program called `london_bridge.py` that takes a list of people, places them in a queue, and displays the lines to the song "London Bridge is Falling Down," one line per person. Use a rotation strategy similar to the one you used in `hot_potato.py` to rotate people through the queue.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. Computers use queues in lots of different ways: print jobs, for example, are delivered to a printer, and typically processed in the order they are received. But what do the `nice` and `renice` commands do in Linux?

REFERENCES

```
#!/usr/bin/env python3

"""
queue_tester.py
Demonstrates the use of the Queue class.
```

```
@author Richard White
@version 2016-12-17
"""

from atds import Queue

def main():
    print("Testing the Queue class")
    testsPassed = 0
    try:
        q = Queue()
        testsPassed += 1
        print("Test passed: queue created")
    except:
        print("Test failed: couldn't initialize queue")

    try:
        q.enqueue("hello")
        q.enqueue(3)
        testsPassed += 1
        print("Test passed: items queued")
    except:
        print("Test failed: couldn't push onto queue")

    try:
        result = q.dequeue()
        if (result == "hello"):
            testsPassed += 1
            print("Test passed: item dequeued")
        else:
            print("Test failed: incorrect dequeue result")
    except:
        print("Test failed: couldn't dequeue")

    try:
        result = q.is_empty()
        if (not result):
            testsPassed += 1
            print("Test passed: is_empty returned correct result")
        else:
            print("Test failed: queue has items, but indicated empty")
    except:
        print("Test failed: is_empty() method unavailable")

    try:
        result = q.size()
        if (result == 1):
            testsPassed += 1
            print("Test passed: correct size returned")
        else:
            print("Test failed: incorrect size returned")
    except:
        print("Test failed: .size() method unavailable")

    try:
        q.dequeue()
    except:
        pass

    try:
        result = q.is_empty()
        if (result):
            testsPassed += 1
            print("Test passed: is_empty() correctly indicating empty status")
        else:
            print("Test failed: queue failed to indicate empty status")
    except:
        print("Test failed: is_empty() unavailable")

    print(str(testsPassed) + "/6 tests passed")

if __name__ == "__main__":
    main()
```