

CSC148, Lab #2

week of September 23rd, 2013

To earn your lab mark you must seriously participate in the lab. You are graded on *participation*, not results. Please note that participation includes showing up on time (10 minutes after the hour, at U of T).

General rules

Same general **rules** as Lab #1, in particular navigator and drivers. You are encouraged to begin work on this lab once you arrive, and not earlier. If you do happen to finish some of it earlier, bring no notes or electronic files, so that you can participate with a fresh perspective during the lab. Lab materials are found at <http://www.cdf.toronto.edu/~heap/148/F13/Labs/lab02/>

Show your work to your TA frequently, and feel free to them ask questions —that’s why they’re here! Don’t hesitate to also ask other students if you get stuck, and please be generous in helping others.

Using stacks

Getting started

(Student s1 drives and s2 navigates.)

1. Save the file `stack.py`, available at
<http://www.cdf.toronto.edu/~heap/148/F13/Labs/lab02/>
2. Start wing and open `stack.py`
3. Create a new file called `driver.py`, and add code to it so that it:
 - (a) creates a new Stack;
 - (b) puts the string 'Hello' in the stack;
 - (c) removes the string from the stack and prints it.

All of the above should be under the `if __name__ == '__main__':` statement.

Using a stack

(Switch roles: now, s2 drives and s1 navigates.)

1. Change `driver.py` to read in a list of lines from standard input, until the string "end" is typed, and put each line into a Stack. Remember that a “read and process loop” looks like this:

```
Read a line
while not done:
    Process the line that was read
    Read the next line
```

You can read a line using: `input("Type a string:")`.

2. Add code at the end of `driver.py` to remove and print the lines you stored in the stack. When the stack becomes empty, end the program.

Show your work to your TA before moving on to the next section.

Queues

A queue is another ADT that stores a sequence of items. Unlike a stack, items are added at the “back” and removed from the “front,” in first-in, first-out order (like a lineup at a grocery store). Here are the operations supported by the queue ADT:

- `enqueue(item)`: Add item to the back of the queue.
- `dequeue()`: Remove and return the front item. Raise `EmptyQueueError` if the queue is empty.
- `front()`: Return the front item without removing it. Raise `EmptyQueueError` if the queue is empty.
- `is_empty()`: Return `True` iff the queue is empty.

Implementation

(Switch roles again: `s1` drives and `s2` navigates.)

1. Write a `Queue` class in a file named `queue.py` to implement the queue ADT.
2. Download the file `testqueue.py`, open it in Wing, and run it to test your `Queue` class.

Using a queue

(Switch roles again: `s2` drives and `s1` navigates.)

NEWS FLASH: python has recently added a built-in module called `queue`, which in some environments (e.g. Wing) is seen before the `queue` module in your directory. Here are two possible work-arounds:

- You could re-name the file `queue.py` to something unlikely to be a built-in python module (perhaps `my_queue.py`, and also re-name your import statements in `testqueue.py`, `timequeue.py`, `driver.py` to match.
 - You could look at the two lines in `testqueue.py` that occur before any import statements (the first of the two lines says `"import sys"`, and use that to make sure the the current directory is examined first.
1. Modify `driver.py` so that it uses a queue instead of a stack, and expects input lines to contain integers (except for the “end” line).

- Recall that you can convert an input line to an integer by calling the `int` constructor.
 - Store all the integers in a queue.
2. Now make your program print the product of all the numbers that were in the queue. (Do this by removing the items in the queue one by one.)

Show your work to your TA before moving on to the next section.

Efficiency

(Switch roles once more, with feeling: `s1` drives and `s2` navigates.)

1. Download the file `timestack.py`, open it in Wing, and run it to get information about the efficiency of the Stack class.

How does the running time grow as a function of the number of items on the stack?

2. Download the file `timequeue.py`, open it in Wing, and run it to get information about the efficiency of your Queue class.

How does the running time grow as a function of the number of items in the queue?

3. Get out a piece of paper and spend some time with your partner trying to figure out why your Queue class behaves this way. Also brainstorm possible solutions to this $O(n)$ efficiency (where n is the number of items that are in the queue).

Hint: Draw a list, and think about keeping track of “front” and “back” indices, and ignoring some parts of your list. Don’t worry if you can’t come up with a solution, but do spend some time discussing!