CSC148 Lab #3, winter 2018

learning goals

In this lab you will:

- practice using stacks, implement a queue, then practice using them

You should begin working on these on your own well before Thursday, and you are certainly welcome to come and get some guidance from your TA on working through these exercises. There will be a short quiz during the second hour of your lab.

setup stacks

1. Create a directory called **lab3** and copy stack.py into it.
2. Open **stack.py** in PyCharm (or another IDE)
3. Create a new file in **lab3** called **stack_client.py**.

You'll probably need to import the **stack** module to get started in **stack_client.py**.

use stacks

Now write code in the **if __name__== '__main__:'** block of **stack_client.py** that will:

1. Create a new stack.
2. Read text typed from the keyboard, using **input("Type a string:")**.
3. Add the typed string to the stack.
4. Repeat the previous two steps until the string **end** is typed
5. Remove the strings, one-by-one, from the stack and print them.

Above the **if __name__== '__main__:'** block, write a function called **list_stack** that takes a list and a stack as arguments, has a **None** return, and does the following:

1. Adds each element of the list to the stack.
2. Once all elements are added, removes the top element from the stack. If the element is a non-list, it prints it. If the element is a list, it stores each of **its** elements on the stack.
3. Continue the previous step until the stack is empty. Check for an empty stack, rather than causing an exception to be raised!

Try out your **list_stack** function on:

- `[1, 3, 5]`
- `[1, [3, 5], 7]`
- `[1, [3, [5, 7], 9], 11]`

implement queue

A queue is another abstract data type (ADT) that stores a sequence of values. Unlike a stack, where the last item in is the first item out (LIFO), a queue makes sure that the first item in is the first item out (FIFO). This models the lineup at a coffee shop or vending machine.

The operations your queue will support are:

add: add an object at the end of the queue.

remove: remove and return the object at the beginning of the queue.

is_empty: return True if this queue is empty, False otherwise.

To implement a queue you should

1. Open csc148_queue.py in PyCharm or another IDE.
2. Complete all the unimplemented methods and store **csc148_queue.py** in your **lab3** directory.
3. Download testqueue.py, open it in PyCharm, and run it to see whether your implementation of **Queue** passes the unit tests in it.

Create an **if __name__== '__main__:'** block in a new file that you create, **queue_client.py,**[1] and add some more code to:

1. Create a new queue.
2. Prompt for an integer at the keyboard, and add it to the queue. Remember that the built-in function **input(...)** returns a string, from which you can construct an integer using **int(...)**.
3. Repeat the previous step until you have read in, but **not** stored, 148.
4. Print the sum of all the numbers that were in the queue.

Now above the **if __name__== '__main__:'** block, create a function **list_queue** which takes a list and a queue as arguments, and does the following:

1. Adds each element of the list to the queue.
2. Once they are all added, removes the top element from the queue. If the element is a non-list, print it. If the element is a list, store each of **its** elements on the queue.
3. Continue the previous step until the queue is empty. Check for an empty queue, rather than causing an exception to be raised!

Try out your **queue_list** function on:

- `[1, 3, 5]`
- `[1, [3, 5], 7]`
- `[1, [3, [5, 7], 9], 11]`

[1] Of course, you'll need to **import csc148_queue**

create unit tests for stack

Emulate (that is, copy **intelligently**) the unit tests in **testqueue.py** to create unit tests for our **Stack** class. Of course you will slightly modify the tests, since a stack isn't a queue. Save your stack tests in **teststack.py**.

In this week's lecture we have some pointers on how to create unit tests.

addition exercises

For the examples above that use **list_stack** and **list_queue**, draw a diagram that shows the elements remaining on the stack/queue after each **print** statement. Show the elements in order, labelling the top/bottom or front/back.