UNIVERSITY OF TORONTO
Faculty of Arts and Science

Midterm 1 Solutions CSC148H1F

Duration: 50 min. Instructors: Diane Horton, David Liu. Examination Aids: Provided aid sheet

Name:

Student Number:

Please read the following guidelines carefully.

- Please print your name and student number on the front of the exam.
- This examination has 4 questions. There are a total of 17 pages, DOUBLE-SIDED.
- The last page is an aid sheet that may be detached.
- You may always write helper functions/methods unless explicitly asked not to.
- Docstrings are not required unless explicitly asked for.

Take a deep breath.

This is your chance to show us

How much you’ve learned.

We WANT to give you the credit

That you’ve earned.

A number does not define you.

<table>
<thead>
<tr>
<th>Question</th>
<th>Grade</th>
<th>Out of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>29</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. **[9 marks]** The following questions test your understanding of the terminology and concepts from the course. For the short-answer questions, you may answer in either point form or full sentences; you do not need to write much to get full marks!

(a) **[4 marks]** The following function attempts to filter a queue. It runs without crashing, but it doesn’t pass its doctests.

```python
def filter_queue(q: Queue[int], minimum: int) -> None:
    """Remove all items from <q> that are less than <minimum>."
    
    >>> q = Queue()
    >>> q.enqueue(2)
    >>> q.enqueue(21)
    >>> q.enqueue(5)
    >>> q.enqueue(1)
    >>> filter_queue(q, 10)
    >>> q.dequeue()
    21
    >>> q.is_empty()
    True
    ""
    temp_queue = Queue()

    while not q.is_empty():
        value = q.dequeue()

        if value >= minimum:
            temp_queue.enqueue(value)

    q = temp_queue
```

On the next page, we have drawn the state of memory after a `Queue` has been constructed and given contents as in the doctest:

```python
>>> q = Queue()
>>> q.enqueue(2)
>>> q.enqueue(21)
>>> q.enqueue(5)
>>> q.enqueue(1)
```
(i) Modify the diagram to show the state of memory if we have called `filter_queue(q, 10)` and have paused right before the function returns (immediately after executing the line `q = temp_queue`).
Note: this implementation of Queue uses the front of the _items list as the front of the queue.
(ii) The first doctest expects:
>>> q.dequeue()
21
What would actually be returned/happen?

Solution
The queue is now empty, so when we call dequeue it returns None. (The aid sheet says that this is its behaviour on an empty queue.)

(iii) The second doctest expects:
>>> q.is_empty()
True
What would actually be returned/happen?
(iv) Finally, fix the code by making changes directly on the given code on the previous page. Clearly cross out any parts you want to remove, write any new code that should be added, and clearly indicate where the new code should go.

Solution

We have to mutate the original queue object by putting the items from the temporary queue into it. Remove the line `q = temp_queue`, and replace it with:

```python
    while not temp_queue.is_empty():
        q.enqueue(temp_queue.dequeue())
```

Note that the filtering work could have been done in this loop instead of while creating the temporary queue. That approach wastes space, because it saves things in the temporary queue that do not have to be saved.
(b) [2 marks] State two differences between methods and top-level functions. The differences can be design-related or technical (code-related).

**Solution**

Possible answers include:

- A method is defined within a class (indented within the `class A` block), while a function is defined outside a class.
- A method is logically some behaviour/computation that is associated with instances of a class, while a function’s purpose can be more general.
- A method can access private attributes/methods inside a class, while top-level functions should only make use of the public interface of a class.
- A method is normally called using dot notation, while a function is not.
- The first parameter of a method should always be called `self`, referring to the instance of the class; a function has no such restriction on the order or names of its parameters.

(c) [3 marks] Suppose we have an abstract class defined as follows:

```python
class MyAbstractClass:
    def __init__(self, x, y):
        # Body omitted, but there is an implementation here.

    def method1(self):
        raise NotImplementedError

    def method2(self):
        # Body omitted, but there is an implementation here.
```

Now suppose we want to write a subclass of `MyAbstractClass` called `MySubclass` that is *not* abstract.

(i) What method(s) *must* be implemented by `MySubclass`? Only write the method name(s).
(ii) What might go wrong if we do not implement this/these method(s)?

**Solution**

If we create an instance of `MySubclass` and call `method1` on the instance, we’ll get an error (`NotImplementedError`).

(iii) What method(s) can we choose to override in `MySubclass`? Only write the method name(s).

**Solution**

`__init__` and `method2`

(iv) What is one reason we might not want to override this/these method(s)?

**Solution**

We want to use the default behaviour already implemented in `MyAbstractClass`.

(v) Suppose we choose not to override the initializer of `MyAbstractClass`, and then run the following code:

```python
>>> obj = MySubclass(10, 20)
>>> print(obj.x + obj.y)
```

What is the result of executing this code? If there is output, explain. If there is an error, explain. If it is impossible to tell from the information given, explain.

**Solution**

Impossible to tell from the information given! Because we haven’t provided the documentation for `MyAbstractClass`. 
MyAbstractClass, or the implementation of its initializer, we don’t know whether \( x \) and \( y \) are instance attributes of the class, and even if they are, what their types are.
2. [4 marks] Here is a function that operates on a stack. Complete its docstring by adding three elements:

(a) An English description of what the function does.
(b) A doctest example that makes use of a stack of size at least 3.
(c) Any preconditions necessary to ensure the function will not raise an error.

def mystery(s: Stack) -> None:

    Solution

    Swap the top and bottom element of the given stack.

    Precondition: the input stack contains at least two elements.

    >>> s = Stack()
    >>> s.push(1)
    >>> s.push(2)
    >>> s.push(3)
    >>> s.push(4)
    >>> mystery(s)
    >>> s.pop()
    1
    >>> s.pop()
    3
    >>> s.pop()
    2
    >>> s.pop()
    4
    >>> s.is_empty()
    True

    one = s.pop()
    temp = Stack()
    while not s.is_empty():
        temp.push(s.pop())
    two = temp.pop()}
s.push(one)
while not temp.is_empty():
    s.push(temp.pop())
s.push(two)
3. **[9 marks]** You are responsible for designing a class to keep track of a simple guessing game in which people enter guesses for the number of jellybeans in a jar. The winner of the game is the one whose guess is closest to the actual number of jellybeans.

Here is an example of how we want to use it:

```python
>>> g = JellyBeanCompetition(1000)  # There are 1000 jellybeans in the jar.
>>> g.record_guess('homer', 'doh@gmail.com', 20)
>>> g.record_guess('marge', 'blue@gmail.com', 800)
>>> g.record_guess('lisa', 'sax@gmail.com', 1002)
>>> g.record_guess('bart', 'cow@gmail.com', 1500)
>>> g.winner()
sax@gmail.com
```

Below and on the next page, we have a very incomplete class design for this class. You have tasks marked ‘TODO’ in the code:

- Document all the attributes of the `JellyBeanCompetition` class. You may choose any reasonable way to store the necessary data. Make all attributes private.
- Implement the class initializer so that it is compatible with the example code above.
- Complete the docstring for `winner`, and implement the method. It’s up to you to decide what happens when there are multiple guesses that are the closest to the correct number of jellybeans.

**Note**: you do not need to write any documentation or code for `record_guess`; assume this has been implemented properly to be consistent with the above code and the attributes you’ve chosen.

Solution

```python
class JellyBeanCompetition:
    """A competition for guessing the number of jelly beans in a jar.

    === Attributes ===
    _correct_count
        The actual number of jelly beans in the jar.
    _guesses
        The guesses that have been made. Each guess
        is represented by a tuple consisting of the
guesser's name, email address, and their guess.
The guesses are in order according to when they
were recorded, with the most recent guess last.
"""
```
_correct_count: int
_guesses: List[Tuple[str, str, int]]
Solution

def __init__(self, count: int) -> None:
    self._guesses = []
    self._correct_count = count

def winner(self) -> None:
    """Print the email address of the winner of this jellybean competition.
    The winner is the person whose guess is closest to the actual number of jellybeans.
    
    Precondition: there is at least one guess in the competition.
    
    **If there is more than one winner, only print out the email address of the
    one who made the guess earliest.**
    """
    # Assumes there is at least one guess.
    best_i_so_far = 0
    best_diff_so_far = abs(self._correct_count - self._guesses[0][2])
    for i in range(len(self._guesses)):
        next_diff = abs(self._correct_count - self._guesses[i][2])
        if next_diff < best_diff_so_far:
            best_diff_so_far = next_diff
            best_i_so_far = i
    print(self._guesses[best_i_so_far][1])
4. [7 marks] Implement the following function according to its docstring.

For this question, you should refer to the documentation of the LinkedList class found on the aid sheet. You may use all attributes (public and private) of the LinkedList and Node classes, and you may use their initializers. You may not use any other linked list methods.

```python
def swap(lst: LinkedList, i: int, j: int) -> None:
    """Swap the values stored at indexes <i> and <j> in the given linked list.

    Precondition: i and j are >= 0.
    Raise an IndexError if i or j (or both) are too large (out of bounds for this list).
    NOTE: You don't need to create new nodes or change any "next" attributes.
    You can implement this method simply by assigning to the "item" attribute of existing nodes.
    """
    >>> linky = LinkedList([10, 20, 30, 40, 50])
    >>> swap(linky, 0, 3)
    >>> str(linky)
    '[40 -> 20 -> 30 -> 10 -> 50]'
```

Solution

```python
    # Go to the node at index i.
    curr_i = lst._first
    curr_index = 0
    while curr_i is not None and curr_index < i:
        curr_i = curr_i.next
        curr_index += 1
    
    # Go to the node at index j.
    curr_j = lst._first
    curr_index = 0
    while curr_j is not None and curr_index < j:
        curr_j = curr_j.next
        curr_index += 1

    if curr_i is None or curr_j is None:
        # At least one of i and j is out of bounds
        raise IndexError
```
else:
    # Both nodes are in bounds; swap their items.
    curr_i.item, curr_j.item = curr_j.item, curr_i.item
Use this page for rough work. If you want work on this page to be marked, please indicate this clearly at the location of the original question.
Use this page for rough work. If you want work on this page to be marked, please indicate this clearly at the location of the original question.