

PLEASE HAND IN

UNIVERSITY OF TORONTO  
Faculty of Arts and Science

Midterm Test

CSC 148H1F, Fall 2019  
Section L0101/L0102 (Smith)

Duration — 110 minutes  
Aids allowed — Provided aid sheet

PLEASE HAND IN

Student Number: \_\_\_\_\_

UTORid: \_\_\_\_\_

Last (Family) Name: \_\_\_\_\_

First (Given) Name: \_\_\_\_\_

---

Do **not** turn this page until you have received the signal to start.  
In the meantime, please fill out the section above, and read the instructions below.

---

- This midterm consists of 7 questions on 14 pages, DOUBLE-SIDED. # 1: \_\_\_\_\_/ 5
- There is a separate aid sheet provided. You do not need to hand it in, and nothing you write on it will be marked. # 2: \_\_\_\_\_/ 6
- There are blank pages at the end of the test for rough work. # 3: \_\_\_\_\_/ 8
- **Do not remove them.** # 4: \_\_\_\_\_/ 6
- You may always write helper functions unless asked not to. # 5: \_\_\_\_\_/ 4
- Documentation is *not* required unless asked for. # 6: \_\_\_\_\_/ 8
- You may write in either pen or pencil. # 7: \_\_\_\_\_/ 7
- Good luck! We want you to do well! TOTAL: \_\_\_\_\_/44

**Question 1.** [5 MARKS]

Consider the following code:

```
class BagOfStuff:
    def __init__(self, stuff: List[str]) -> None:
        self.stuff = stuff

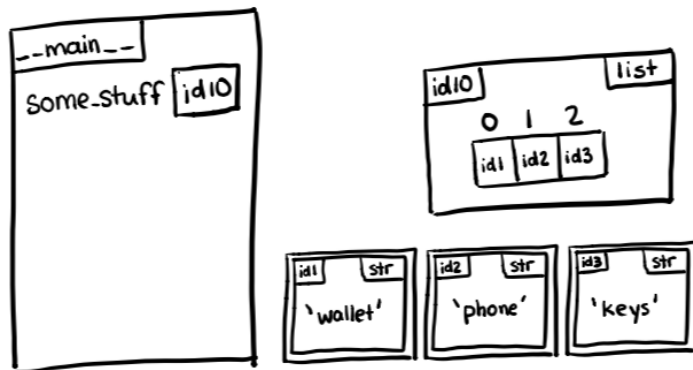
if __name__ == '__main__':
    some_stuff = ['wallet', 'phone', 'keys']
    my_stuff = BagOfStuff(some_stuff)
    some_stuff.append('snacks')
    your_stuff = BagOfStuff(some_stuff)
    assert len(my_stuff.stuff) == 3 and len(your_stuff.stuff) == 4
```

When we run this code, the assertion fails!

**Part (a)** [3 MARKS]

To debug the code, we have begun to trace the code in the `__main__` block. The memory model diagram shows the state of memory right after the line `some_stuff = ['wallet', 'phone', 'keys']` has been executed. Complete the diagram to show the state of memory immediately before the `assert` line runs.

You can trace through the calls on the call stack if it helps you, but you do not have to include them in your final diagram. You don't have to include anything from the `assert` line in your diagram.

**Part (b)** [2 MARKS]

What causes the assertion to fail, and what needs to be changed inside the class `BagOfStuff` to fix this issue? Be specific and use correct terminology.

**Question 2.** [6 MARKS]

**Part (a)** [3 MARKS] For each pair of classes from Assignment 0 or Assignment 1, check the box to indicate whether they have a Composition relationship, an Inheritance relationship, or neither.

Class 1	Class 2	Composition?	Inheritance?	Neither?
Chirper	Cheep			
Customer	Item			
ExpressLine	Customer			
ExpressLine	SelfServeLine			
PriorityQueue	Container			
GroceryStore	GroceryStoreSimulation			

**Part (b)** [1 MARK] Give an example from Assignment 1 of where you used polymorphism. You can write an example line of code, with the types indicated, or describe in words.

**Part (c)** [1 MARK] One of the subclasses of `CheckoutLine` was `RegularLine`, which represented a line with no special features. Why did we have to create this subclass, rather than just creating an instance of `CheckoutLine`?

**Part (d)** [1 MARK] Assume the code below has been run.

```
class A:
    def hey(self) -> None:
        print("I'm an A")

class B(A):
    def hey(self) -> None:
        print("I'm a B")

class C(A):
    def some_method(self) -> None:
        print('C is for Cat')

def type_checker(things: List[A]) -> None:
    for thing in things:
        thing.hey()
```

What happens when the following calls are executed? Either give the output if the call runs without error, or explain why an error occurs.

(i) `type_checker([A(), B()])`

(ii) `type_checker([C()])`

**Question 3.** [8 MARKS]

You are responsible for creating a class to represent a playlist containing songs. Each song in the list has a title, an artist, and a duration. None of those three attributes are unique, but each artist has at most one song with a particular title. That is, there are no (title, artist) duplicate pairs.

Here is sample usage of the `Playlist` class.

```
>>> pl = Playlist()
>>> pl.play_next() # nothing happens
>>> pl.add_song('Free Falling', 'Tom Petty', 4.5)
>>> pl.add_song('Zombie Zoo', 'Tom Petty', 3.5)
>>> pl.add_song('Hello', 'Lionel Richie', 4.0)
>>> pl.add_song('Hello', 'Adele', 5.0)
>>> pl.add_song('Hello', 'Adele', 5.0) # not added again
>>> pl.get_total_duration()
17.0
>>> pl.play_next()
'Free Falling by Tom Petty'
>>> pl.play_next()
'Zombie Zoo by Tom Petty'
>>> pl.reset()
>>> pl.play_next()
'Free Falling by Tom Petty'
>>> pl.get_songs_by('Tom Petty')
['Free Falling', 'Zombie Zoo']
```

Below and on the next pages complete the implementation of this class so that the example code above will run as shown. You may choose any reasonable way to store the necessary data.

Do not add anything to the public interface for this class beyond what is demonstrated in the example code. You can add private attributes and private helper methods.

We have imported the types we think you will need from `typing`, however you can assume any others you need for your type contracts are also available, and you do not need to use all the imported ones.

You do not need to do any additional handling of errors or behaviours beyond what is demonstrated in the sample usage.

```
from typing import List, Tuple, Optional
```

```
class Playlist:
    """A class to represent a list of songs.

    === Attributes ===
    # TODO: Describe all instance attributes here.
    # Be clear and precise; this will help us understand your code.

    """
    # TODO: Write type annotations for your attributes here.
```

**Question 3.** (CONTINUED)

# TODO: Write the headers and bodies for the 6 required methods on this and the following page.  
# Each method header must include a type contract, but docstrings are NOT required.

# TODO: Write the initializer here.

# TODO: Write the play\_next method here.

# TODO: Write the add\_song method here.

# TODO: Write the reset method here.

**Question 3.** (CONTINUED)

```
# REMINDER: Each method header must include a type contract, but docstrings are NOT required.  
# If you want to write any helper methods, you can include them at the bottom of this page.  
# You do not need to use helpers in your solution if you do not want to.  
  
# TODO: Write the get_total_duration method here.
```

```
# TODO: Write the get_songs_by method here.
```

**Question 4.** [6 MARKS]

Recall the `Stack` class that we have used in the course. You can find an implementation of `Stack` on the provided aid sheet, and you can assume everything from the aid sheet has been imported for you to use.

Suppose we want to define a new kind of `Stack` called a `MegaStack`. A `MegaStack` has the same public interface as a `Stack`, plus two more operations, implemented as public methods.

- The method `mega_push` takes a `Queue` object and pushes everything from the `Queue` onto the `MegaStack` in the order it is removed from the `Queue`. When the method returns, the `Queue` will be empty, and all of its contents will now be on the `MegaStack`. `mega_push` returns nothing.
- The method `mega_pop` takes an integer value, and pops up to that many items from the `MegaStack` and returns them in a `Queue` object. If there are less than the given number of items in the `MegaStack`, the `MegaStack` is empty and all its items are returned in the `Queue`. The items are added in the `Queue` in the same order they are popped from the `MegaStack`.

Write this new class. Part of the marks will be for good design. In particular, avoid repeated code. Your solution should not contain even a single line that has been previously written.

You must write complete type contracts, but you do not need docstrings (but you can if it helps you!)

**Question 5.** [4 MARKS]

Consider the following code:

```
def search_queue(q: Queue, item: Any) -> bool:
    """Return True iff item is contained in q.

    >>> queue = Queue()
    >>> queue.enqueue(1)
    >>> queue.enqueue(2)
    >>> search_queue(queue, 2)
    True
    >>> search_queue(queue, 3)
    False
    """
    found = False
    temp_q = Queue()
    while not q.is_empty():
        cur = q.dequeue()
        if cur == item:
            found = True
            temp_q.enqueue(cur)
    q = temp_q
    return found
```

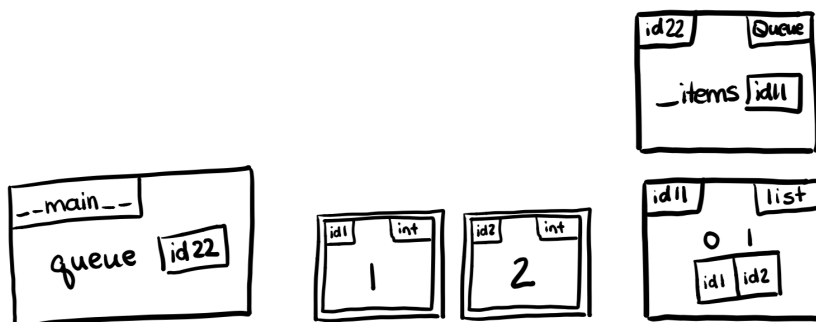
**Part (a)** [1 MARK]

Does this function pass its doctest? Circle one:      Yes      No

**Part (b)** [3 MARKS]

We have begun to trace the code in the docstring in the code block above. The memory model diagram shows the state of memory right *before* `search_queue(queue, 2)` is executed. Complete the diagram to show the state of memory immediately *after* `search_queue(queue, 2)` returns.

You may find it helpful to draw out all stack frame and any objects that are created during the function call, however you do not need to include them in your final diagram. You can assume in the diagram that Python uses the shortcut that allows it to create aliases to the same immutable object.





**Question 6.** [8 MARKS]

Answer the questions below about LinkedLists. You can assume these are part of the `LinkedList` implementation on the aid sheet. Note the preconditions – you do not need to worry about an index out of range for this question. **You will likely find it very helpful to draw yourself some pictures.**

**Part (a)** [4 MARKS]

The following implementation of `LinkedList.insert` does not work correctly, although it is pretty close. In fact, there are only two lines in the entire method that need to be fixed.

Circle the two lines that need to be corrected, and rewrite the corrected lines next to them. Do not rewrite the other lines.

```
def insert(self, index: int, item: Any) -> None:
    """Insert the given item at the given index in this list.

    Precondition: 0 <= index <= len(self)
    Note that adding to the end of the list is okay.
    """
    new_node = _Node(item)
    if index == 0:
        new_node.next = self._first.next
        self._first = new_node
    else:
        cur = self._first
        i = 0
        while cur is not None and i < index - 1:
            cur = cur.next
            i += 1
        new_node.next = cur.next
        cur = new_node
```

**Question 6.** (CONTINUED)**Part (b)** [4 MARKS]

Complete the implementation of `LinkedList.pop` below. Your solution must follow the structure outlined in the starter code.

```
def pop(self, index: int) -> Any:
    """Remove and return the item at position <index>.

    Precondition: 0 <= index < len(self)

    >>> lst = LinkedList([1, 2, 10, 200])
    >>> lst.pop(1)
    2
    >>> lst.pop(2)
    200
    >>> lst.pop(0)
    1
    """
    # pop the first node
    if index == 0:
        removed = self._first
        self._first = self._first.next
        return removed.item
    # pop another node
    else:
        cur = self._first
        i = 0
        # complete the while loop condition and body to find the node to modify

        while

        # remove the node at index from the list and return its item
```

**Question 7.** [7 MARKS]

In Assignment 1, you implemented your `PriorityQueue` using an underlying Python list, as in the code below. For this question, you can assume all items have unique priority – there are no duplicate priorities.

```
class PriorityQueue:
    def __init__(self) -> None:
        """Initialize an empty PriorityQueue"""
        self._items = []

    def remove(self) -> Any:
        """Remove and return the highest priority item from this PriorityQueue"""
        return self._items.pop(0)

    def add(self, item: Any) -> None:
        """Insert <item> into this PriorityQueue"""
        i = 0
        while i < len(self._items) and self._items[i] < item:
            i += 1
        self._items.insert(i, item)
```

**Part (a)** [1 MARK] Write down the Big-Oh expression for the running time in terms of  $n$  for **removing** one item from a `PriorityQueue` with  $n$  items using the implementation above.

**Part (b)** [1 MARK] Write down the Big-Oh expression for the running time in terms of  $n$  for **adding** one item from a `PriorityQueue` with  $n$  items using the implementation above.

Now, assume you change the `PriorityQueue` class above to be implemented with an underlying `LinkedList`, as defined on the aid sheet. Assume the `insert` and `pop` methods are the ones from Question 6, and that they are implemented correctly.

Review the implementation and answer the questions on the following page.

**Question 7.** (CONTINUED)

```

class PriorityQueue:
    def __init__(self) -> None:
        """Initialize an empty PriorityQueue"""
        self._items = LinkedList([])

    def remove(self) -> Any:
        """Remove and return the highest priority item from this PriorityQueue"""
        return self._items.pop(0)

    def add(self, item: Any) -> None:
        """Insert <item> into this PriorityQueue"""
        i = 0
        cur = self._items._first
        while cur is not None and cur.item < item:
            cur = cur.next
            i += 1
        self._items.insert(i, item)

```

**Part (c)** [1 MARK] Write down the Big-Oh expression for the running time in terms of  $n$  for removing one item from a `PriorityQueue` with  $n$  items using the `LinkedList` implementation.

**Part (d)** [2 MARKS] Describe the situation where adding one item to a `PriorityQueue` with  $n$  items using the `LinkedList` implementation would take the *fewest* steps. Then, give the Big-Oh expression for the running time in terms of  $n$ .

**Part (e)** [2 MARKS] Describe the situation where adding one item to a `PriorityQueue` with  $n$  items using the `LinkedList` implementation would take the *most* steps. Then, give the Big-Oh expression for the running time in terms of  $n$ .

*[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark. **Do not remove this page!**]*

*[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark. **Do not remove this page!**]*