Term Test

My score

94% (47/50)

D8F562D4-E6C5-4EA5-BEAA-95BD95F83777

term-test-da56f

THE UNIVERSITY OF TORONTO – MISSISSAUGA – CSC148H5S

Duration: 1 hour and 50 minutes. **Date:** February 24, 2025

Make sure you have read the instructions on the first page.

Make sure you read each question **CAREFULLY**.

Make sure you follow each question instructions
and restrictions **EXACTLY**.

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.

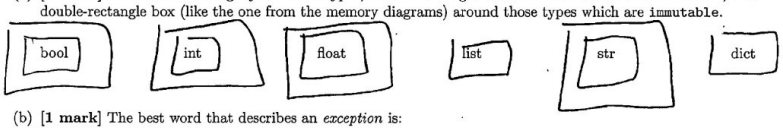
A number does not define you.

Good luck!

MC 1a-e 7

1. **[11 marks] General knowledge - Short answers**

(a) **[1 mark]** Given the following Python data types, draw a rectangle box around those which are `mutable`, and double-rectangle box (like the one from the memory diagrams) around those types which are `immutable`.



(b) **[1 mark]** The best word that describes an *exception* is:

A. recursion
B. control flow
C. function
D. annotation
E. iteration

(c) **[3 marks]** Fill in the blanks with 3 different choices from the options given below, in order to *correctly* complete this sentence:

The interface of a class includes A., C., and F.

A. all public attributes
B. methods whose names start with a leading underscore
C. the class docstring
D. `__init__`'s implementation
E. type contracts for private methods
F. docstrings of public methods

(d) **[1 mark]** Consider that you are given the `Queue` class described in the preps and in lab4. As a reminder, the methods of a `Queue` object are `enqueue(self, item: Any)` and `dequeue(self)`. What items remain in the queue after executing the following code?

```
q = Queue()
q.enqueue(4)
q.enqueue(1)
q.enqueue(3)
q.dequeue()
q.enqueue(8)
q.dequeue()
```

A. [4, 1]
B. [3, 8]
C. [1, 3]
D. [4, 8]
E. None

(e) **[1 mark]** Consider the `LinkedList` class that you worked with in the preps, lectures, and labs. You are given a `LinkedList` object referred to by a variable `rhett`. Which of the following expressions evaluate to `True` when the linked list is empty and False otherwise? Select all that apply.

A. rhett == [ ]
B. rhett is None
C. rhett._first._item is None
D. rhett._item is None
E. rhett._first is None

(f) [4 marks] Consider the following classes. We have omitted the docstrings and attribute type contracts for space constraints.

```
class Point:

    def __init__(self) -> None:
        self.x = 2
        self.y = 3

    def move(self, dx: int, dy: int) -> None:
        self.x += dx
        self.y += dy

    def jump(self, dy: int) -> None:
        self.y -= dy

class Point3D(Point):

    def __init__(self) -> None:
        self.z = 4
    def jump(self, dy: int) -> None:
        self.y += dy
        self.z += 100

class ColourPoint(Point):

    def __init__(self, colour: str) -> None:
        Point.__init__(self)
        self.colour = colour

    def jump(self, dy: int) -> None:
        self.y += dy
        self.colour = "red"
```

does not have an x, y attribute

MC 1f 4

For each of the following programs, circle the correct output from the list of options immediately below it.

PROGRAM 1:

```
if __name__ == '__main__':
    c = ColourPoint("blue")
    c.move(4, 4)
    c.jump(8)
    print(c.colour, c.x, c.y)
```

2, 3
6 7
6 15 red

A. blue 4 12
B. red 4 12
C. blue 6 15
D. red 6 15
E. blue 6 7
F. red 6 7
G. red 4 -1

correct! good job! :D 2

H. red 6 -1
I. An exception is raised
J. None of the above.

PROGRAM 2:

```
if __name__ == '__main__':
    cp = Point3D()
    cp.move(5, 3)
    cp.jump(7)
    print(cp.x, cp.y, cp.z)
```

(2, 3, 4) (7, 6, 4) (7, 13, 104)

A. 5 10 104
B. 5 13 104
C. 7 10 104
D. 7 13 104
E. 2 10 104
F. 2 13 104
G. An exception is raised in the move metho
H. An exception is raised in the jump metho
I. An exception is raised in the print function
J. None of the above.

correct! good job! :D 2

Q2 a 3

2. [15 marks] Object-Oriented Programming

Consider that you are given the following Animal class. We have omitted all method implementations, so you may not make any assumptions beyond what you get from the docstrings.

```
class Animal:
    """ An Animal class.

    === Public Attributes ===
    state: the state of the animal, indicating whether the Animal
           is either awake or asleep.
    location: the location of the animal, represented as its
              longitude and latitude.
    """
    state: str
    location: Tuple[float, float]

    def __init__(self) -> None:
        """Initialize an Animal."""

    def sleep(self) -> None:
        """ Change this Animal's state to "asleep". """

    def eat(self) -> None:
        """ Instruct this Animal to feed. """

    def move(self, new_location: Tuple[int, int]) -> None:
        """ Move this Animal to the <new_location>. """
```

add it to bottom of docstring

Representation Invariant

Each of the following sub-questions are independent of each other, unless otherwise stated.

(a) **[3 marks]** For the questions below, **do not make any assumptions that are not stated explicitly in the above code.**

i) Write a representation invariant for the code above and describe also *where* you would add the invariant. The invariant must be based **only** on what you know from the code you are given.

In the Class docstring, I would say that Animal

1 mark: Add to the class docstring self.state ==

ii) Describe in your own words at least 2 precondition invent new preconditions which are not implied in

The type contract is an implicit

1 mark: correct precondi

"awake" or self.state == "asleep" 1

dition for location 1

Q2 b 3

(b) **[3 marks]** You can assume now that the initializer of class `Animal` sets the animal's state to "awake" and its position to some random location on the globe (between -180 and 180 for the longitude, and between -90 and 90 for the latitude).

We now want to create two more classes which are related to the `Animal` class – a `Deer` class, and a `Goose` class. Both the `Deer` and the `Goose` can be awake or asleep, and start at a random location on the globe.

We give you 4 possible implementations of the `Deer` class below.

i. Choose the one implementation you consider to be the best design (circle the class name). You may assume that the `Goose` implementation would also follow the same approach you choose for `Deer`.

ii. Explain (on the right of the circled class name) *clearly* **why** you chose this implementation. Provide detail as to why other options are not adequate.

```
class Deer:
    """ ... """
    animal: Animal

    def __init__(self, new_animal: Animal) -> None:
        """Initialize a Deer object."""
        self.animal = ne

class Deer2(Animal):
    """ ... """
    def __init__(self) -
        """Initialize a
        self.state = "aw
        long = random.randint(-1800, 1800) / 10
        lat = random.randint(-900, 900) / 10
        self.location = (long, lat)

class Deer3(Animal):
    """ ... """
    def __init__(self) -> None:
        """Initialize a Deer3 object."""
        Animal.__init__(self)

class Deer4(Animal):
    """ ... """
    animal: Animal

    def __init__(self, animal: Animal) -> None:
        """Initialize a Deer4 object."""
        self.animal = animal
        Animal.__init__(self)
```

✓ encapsulation X This is clearly an inheritance relationship

Since Deer

must of the initializer code by inheriting from Animal. we don't need to add anything else beyond the parents code.

1 mark: Explaining why reusing the initializer is best cho

1 mark: Explaining why

1 mark: circling the correct class 1

Q2 c 6

(c) **[7 marks]** Consider now that `Animal` is an abstract class where all its methods raise a `NotImplementedError`, and that `Deer` and `Goose` are subclasses of `Animal`.

You are given the following class that keeps track of *all* `Deer` and `Goose` sightings, in a dictionary attribute, indexed by the day of sightings. Multiple animals may have been sighted in a given day.

Complete the parts marked with "TODO" below. *Be specific, detailed, and accurate!* You may not make any additional assumptions other than what you can infer from the code we gave you.

```
class Sightings:
    # docstring omitted
    sights: Dict[date, List[Animal]]

    def __init__(self) -> None:
        """Construct a new Sightings object that keeps track of all animal sightings,
        corresponding to the day when the animal was sighted."""
        self.sights = {}

    def mystery_method(self, day: date, animal: Animal) -> None:
        """
        # TODO: Complete this method's docstring. You must add at least 2 full test
        scenarios as doctests. Each full test scenario must include instantiating any
        necessary objects for the test to work.
        Record that ... <animal>
        >>> sighting = Sightings()
        >>>
        >>>
        >>>
        """
        else:
            self.sights[day] = [animal]

    def feed_all(self, today: date) -> None:
        """Instruct each of the animals sight
        to feed iff they are awake.
        Precondition: there is at least one animal sighted on the date
        """
        # TODO: Write the code (function body) for this method.
        # Feel free to use additional space on next page.
```

0.5 - (my... mal > in

0.5 - (myst... > in the de

1 - (mystery_method) description of the method is accurate and complete for what the method does (ex. add a new sighting of an animal on the date) **1**

Goose/Deer instantiated in each doctest **2**

[(date, Animal

Additional space provided for previous question...

```
if today in self.sights:
    ...
else:
```

1 - (feed_all) getting t... date: self.sights[today]

0.5 - (feed_all) checking if state i...

0.5 - ...

Q2 d 0

(feed_all) using a for loop to iterate through the sight

Incorrect - the efficiency of this is the same between the two implementations.

(d) **[2 marks]** In part (c), assume that the sights dictionary was instead a list of tuples (date, animal) tuples, where date is a date object, and animal is any type of animal des assume that the implementation would be adjusted accordingly to access this attribute not assume anything about this list's elements, other than it stores all sightings correct

What would be the benefit or downside of such a design for the sights attribute?

The benefit/downside (circle one) of designing the sights attribute as a list of tuples is:

accessing all the animal sightings on a specific d

If you wanted to add new functionality to Sightings (new methods, no new attributes), a new functionality (a new method of Sightings) that would be more efficient or less ef this design versus the one from point (c)?

```
def 'Count - Species'(self, species: Str) -> i
    """
    # TODO: describe what the method does, you may omit doctests here.
    This method would return the total number of animals of type
    <species> ever recorded in Sch. Sights. As a precondition,
    <species> string must be either
    This would be more efficient since we only traver
    the entire list once
    """
    # TODO: Explain below in words how the implementation would work.
    # No need to write actual code for this implementation but you may do so.
    First, I would check if <species> precondition holds and repres
    Valid Animal, which we have implemented.
    Then if it's not satisfied, fail silently, else, we
    create an accumulator and traverse each tuple check
    the 1st index for the animal object checking if the
    matches <species>. Then, I would return the total a
```

Q3 9

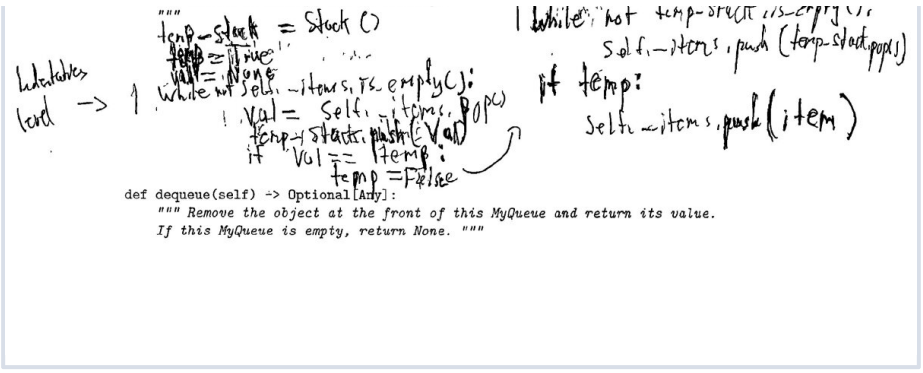
3. **[9 marks] Abstract Data Types**

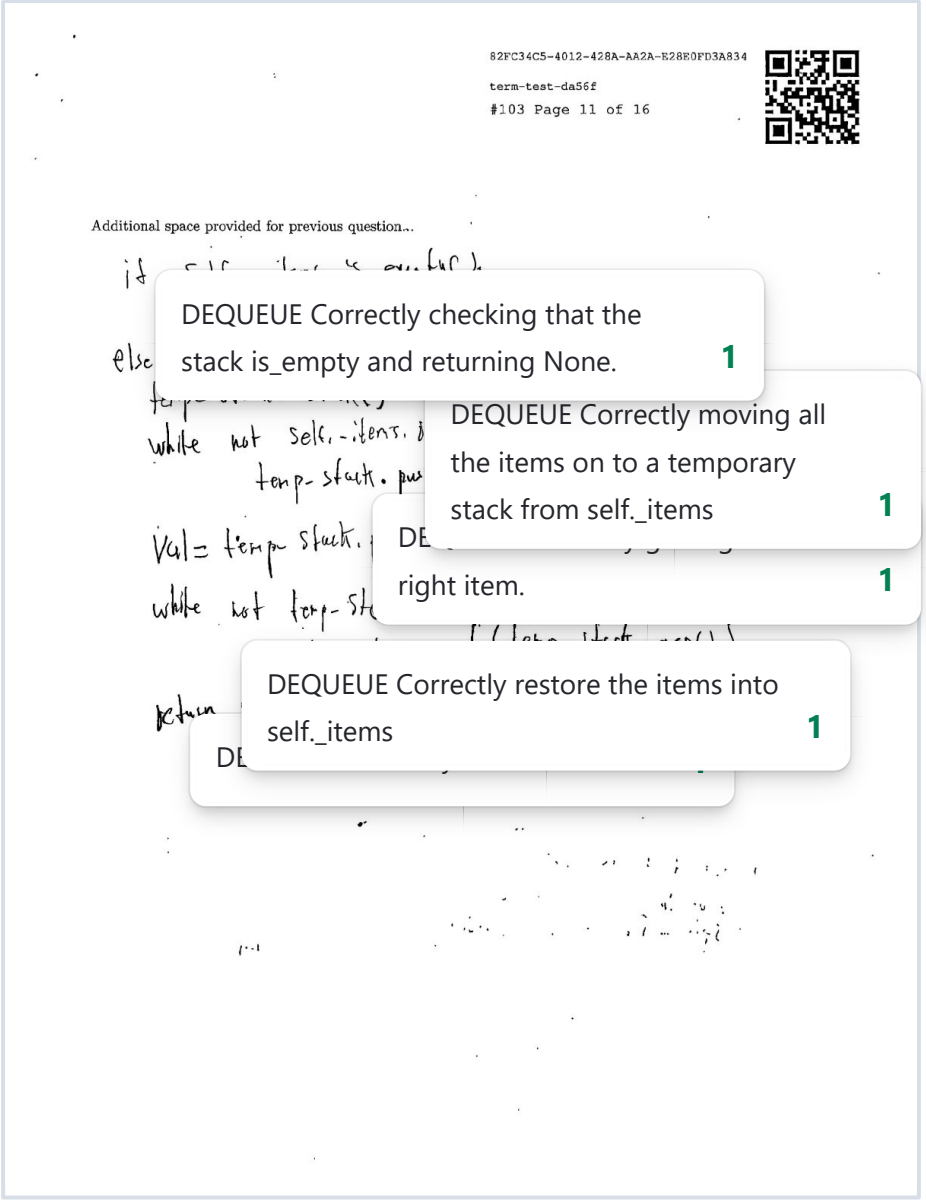
Implement the **enqueue** and **dequeue** methods of the custom MyQueue class below. Read all docstrings carefully.

Restrictions (not following these restrictions will result in zero marks for this question):

- The MyQueue MUST NOT have any other methods aside from those you must implement. The Stack interface is provided in the aid sheet (nothing else can be assumed or modified in it).
- You MUST NOT assume that ANY other method exists for MyQueue.
- You MUST NOT define any helper functions or methods.
- You MUST NOT create any new objects other than Stacks or booleans.
- Your code MUST NOT crash by raising any exception, including EmptyStackError.
- You MAY use new variables.

```
from typing import Any, Optional
from adts import Stack

class MyQueue:
    """ A custom implementation of a queue.

    === Private Attributes ===
    _items: stores the elements of this MyQueue. The back of this MyQueue (where
            elements are enqueued) is represented by the top of the _items Stack.
            The front of the MyQueue (where elements get dequeued) is at the
            bottom of the _items Stack.
    """
    _items: Stack

    def __init__(self) -> None:
        self._items = Stack()

    def enqueue(self, item: Any) -> None:
        """ Add the object referenced by <item> to the back of this MyQueue, if
            and only if the <item> is unique. That is, if the <item> already
            exists in the MyQueue do nothing.
```


"""
temp_stack = Stack()
temp = True
val = None
while not self._items.is_empty():
val = self._items.pop()
temp_stack.push(val)
if val == item:
temp = False
while not temp_stack.is_empty():
self._items.push(temp_stack.pop())
if temp:
self._items.push(item)
def dequeue(self) -> Optional[Any]:
""" Remove the object at the front of this MyQueue and return its value.
If this MyQueue is empty, return None. """


82FC34C5-4012-428A-AA2A-E28E0FD3A834
term-test-da56f
#103 Page 11 of 16
Additional space provided for previous question...
DEQUEUE Correctly checking that the stack is_empty and returning None. 1
DEQUEUE Correctly moving all the items on to a temporary stack from self._items 1
right item. 1
DEQUEUE Correctly restore the items into self._items 1

Q4 8

4. **[8 marks] LinkedLists**

Consider the following code implementing a doubly-linked list (the `DLLinkedList` class), which has `_DLLNode` objects as nodes. A `_DLLNode` object has both a `next` reference to the following node in the list, as well as a `prev` reference to the preceding node in the list.

The list does not wrap around! That is, the last node's `next` attribute references None, and the first node's `prev` references None. Implement the `append` method below.

```
class _DLLNode:
    """A node in a linked list.

    === Public Attributes ===
    item: The data stored in this node
    next: The next node in the list, or None if there are no further nodes after this node.
    prev: The previous node in the list, or None if there is no node before this one.
    """
    item: Any
    next: Optional[_DLLNode]
    prev: Optional[_DLLNode]

    def __init__(self, item: Any) -> None:
        """Initialize a new node storing <item>, with no next or prev node."""
        self.item = item
        self.next = None
        self.prev = None

class DLLinkedList:
    """ A doubly-linked list. """
    _first: Optional[_DLLNode]

    def __init__(self) -> None:
        """Initialize this doubly-linked list."""
        self._first = None

    def append(self, item: Any) -> None:
        """Append <item> to the end of this list."""
```

if self._first is None:
    new_node = _DLLNode(item)
    self._first = new_node
else:

check if _first is None 1

a new node initialized with the item 1

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.*

Continued (repeating else; for indent clarity) :

```
else:
    curr = self._first
    while curr.next is
        curr = curr
    new_node = _DLLNode(item)
    new_node.prev = curr
    curr.next = new_node
```

before list traversal set curr to _first 1

for the loop 1

creating a new _DLLNode 1

sett

setting up curr.next correctly 1

0CE6EA62-DBD2-4267-B56C-7BBE1BA5F701

term-test-da56f

5. [7 marks] **OOP - Assignment 1**

Implement on the next page the `CustomerFilter`'s apply method below, as in Assignment 1. You are provided with a snippet of the interfaces for Call, Customer, and Filter. You may not modify these interfaces, nor add any new attributes or new methods to any class.

```
class Call:
    """ A call made by a customer to another customer.
    === Public Attributes ===
    src_number: source number for this Call
    dst_number: destination number for this Call
    time: date and time of this Call
    duration: duration in seconds for this Call
    src_loc: location of the source of this Call; a Tuple containing the longitude
        and latitude coordinates
    dst_loc: location of the destination of this Call; a Tuple containing the
        longitude and latitude coordinates

    === Representation Invariants ===
    -   duration >= 0
    """
    src_number: str
    dst_number: str
    time: datetime.datetime
    duration: int
    src_loc: tuple[float, float]
    dst_loc: tuple[float, float]
```

```
class Customer:
    """ A MewbileTech customer.
    """
    def get_phone_numbers(self) -> list[str]:
        """ Return a list of all of the numbers this customer owns """

    def get_id(self) -> int:
        """ Return the id for this customer """

    def __contains__(self, item: str) -> bool:
        """ Check if this customer owns the phone number <item> """

    def get_history(self) -> tuple[list[Call], list[Call]]:
        """ Return all the calls from the call history of this
        customer, as a tuple in the following format:
        (outgoing calls, incoming calls) """
```

Q5 7


```
class Filter:
    """ A class for filtering customer data on some criterion. A filter is applied to a set of calls.
    This is an abstract class. Only subclasses should be instantiated. """
    def __init__(self) -> None:
        pass

    def apply(self, customers: list[Customer], data: list[Call], filter_string: str) -> list[Call]:
        raise NotImplementedError

class CustomerFilter(Filter):
    """ A class for selecting only the calls from a given customer. """

    def apply(self, customers: list[Customer], data: list[Call], filter_string: str) -> list[Call]:
        """ Return a list of all calls from <data> made or received by the
        customer with the id specified in <filter_string>.

        The <customers> list contains all customers from the input dataset.

        The filter string is valid if and only if it contains a valid customer ID.
        - If the filter string is invalid, return the original list <data>
        - If the filter string is invalid, your code must not crash.

        Precondition:
        - <customers> contains the list of all customers from the input dataset
        - all calls included in <data> are valid calls from the input dataset
        """
    # TODO: add implementation here
```

dict Comprehension

ids = {customer.get_id() for customer in customers}

try:
    filtered_id = int(filter_string)
except ValueError:
    return data
if filtered_id not in ids:
    return data
else:

Page

extracting customer id from string as int 1

the filter matches any 1

found and returning

the original 'data', or using a try-except to raise a KeyError and handle it accordingly when the customer is invalid **1**

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.*

```
all_numbers = ids[filtered_id].get_phone_numbers()
filtered_calls = []
for call in data:
    if call.src_
                    filtered_calls.append
return filtered_calls
```

getting the phone numbers using the get_phone_numbers() method of the customer **1**

ber or dst_num-
ber is in the cus-
tomers' numbers **1**

tered
call list **1**