Midterm Test

CSC148H1 / LEC0301 — Badr

Winter 2020 — Duration: **110 minutes**

Aids Allowed: **Provided aid sheet**

First name (please write as legibly as possible within the boxes)

Last name

10-digit Student number

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 term test is out of 44 marks and consists of 7 questions on 14 pages (including this one), double-sided. *When you receive the signal to start, please make sure that your copy of the test is complete.*
- There is a separate aid sheet provided. Do not hand it in.
- There are blank pages at the end of the test for rough work. Do not remove them.
- You may always write helper functions unless asked not to. Documentation is not required unless asked for.
- You may write in either pen or pencil.

Good Luck! We want you to do well!

MIDTERM TEST

CSC148H1 / LEC0301 — Badr

Winter 2020 — Duration: **110 minutes**

Question 1. [10 MARKS]

Here's the start of some code for managing rooms in the department of Computer Science. It has a problem: When run, an error is raised.

```
class Room:
    """
    === Public Attributes ===
    users: the names of people who have key access to this Room.
    """
    users: List[str]

    def __init__(self) -> None:
        self.users = []


class Office(Room):
    """
    === Public Attributes ===
    users: the names of people who have key access to this Room.
    owner: the name of the person whose office this is
    bookshelves: the number of bookshelves in this office
    """
    owner: str
    bookshelves: int

    def __init__(self, owner: str, shelves: int) -> None:
        self.owner = owner
        self.bookshelves = shelves


class MeetingRoom(Room):
    """
    === Public Attributes ===
    users: the names of people who have key access to this Room.
    capacity: the number of people this room can seat.
    """
    capacity: int

    def __init__(self, capacity: int) -> None:
        self.capacity = capacity


if __name__ == '__main__':
    ba4260 = Office('Campbell', 3)
    # Memory model diagram is for this moment.
    print(ba4260)
    if len(ba4260.users) == 0:
        print('currently no one has card access')
```

Part (a) [4 MARKS]

Draw a memory model diagram that shows the state of memory immediately before the `print` statement in the main block. If any stack frames are popped prior to this moment, just cross them out rather than erase them.

Part (b) [1 MARK]

Circle the place in the code where an error is raised. Make the circle as specific as possible (for instance circle the relevant part of an expression or the relevant side of an assignment statement).

Part (c) [1 MARK]

What error is raised? (You don't have to give its exact name.) _____

Part (d) [1 MARK]

Correct the code so that this error is not raised. Do not correct or improve anything else; just ensure that the code runs without raising an error. Write your answer directly on the code.

Part (e) [3 MARKS]

Suppose we consider two meeting rooms to be equivalent iff they seat the same number of people. Modify the code on the previous page (you can add anything new on the right-hand side) so that we can compare meeting rooms as follows:

```
ba4290 = MeetingRoom(18)
ba4242 = MeetingRoom(8)
if ba4290 == ba4242:
    print('equivalent rooms')
```

A docstring is not necessary.

Question 2. [5 marks]

Below is an interaction with the Python console. The code runs without error. In each blank space, show the output that would appear. There is space at the bottom of the page for rough work, but it will not be marked.

```
>>> t = [33, 44, 55]
>>> x = [[1, 2], t]
>>> y = []
>>> for item in x:
...     y.append(item)
>>> id(t) == id(x[1])


>>> id(x) == id(y)


>>> id(x[1]) == id(y[1])


>>> t.append(10)
>>> x.append(11)
>>> y.append(12)
>>> y[0].append(13)
>>> x[1].append(14)
>>> x


>>> y
```

Rough work:

Question 3. [6 MARKS]

Recall the WorkoutClass, Instructor and Gym classes from Assignment 0. As a reference, below is the documentation from the Gym class. Also, note the class `Instructor` has a method called `get_id`, which is used in this question.

```
"""A gym that hosts workout classes taught by instructors.

All offerings of workout classes start on the hour and are 1 hour long.

=== Public Attributes ===
name: The name of the gym.

=== Private Attributes ===
_instructors: The instructors who work at this Gym.
    Each key is an instructor's ID and its value is the Instructor object
    representing them.
_workouts: The workout classes that are taught at this Gym.
    Each key is the name of a workout class and its value is the
    WorkoutClass object representing it.
_rooms: The rooms in this Gym.
    Each key is the name of a room and its value is its capacity, that is,
    the number of people who can register for a class in this room.
_schedule: The schedule of classes offered at this gym.  Each key is a date
    and time and its value is a nested dictionary describing all offerings
    that start then. Each key in the nested dictionary is the name of a room
    that has an offering scheduled then, and its value is a tuple describing
    the offering. The tuple elements record the instructor teaching the
    class, the workout class itself, and a list of registered clients. Each
    client is represented by a unique string.

=== Representation Invariants ===
- Two workout classes cannot be scheduled at the same time in the same room.
- No instructor is scheduled to teach two workout classes at the same time.
- If an instructor is scheduled to teach a workout class, they have the
necessary qualifications.
- If there are no offerings scheduled at date and time <d> in room <r> then
<r> does not occur as a key in _schedule[d]
"""
name: str
_instructors: Dict[int, Instructor]
_workouts: Dict[str, WorkoutClass]
_rooms: Dict[str, int]
_schedule: Dict[datetime,
                Dict[str, Tuple[Instructor, WorkoutClass, List[str]]]]
```

In this question, we will consider two buggy implementations of the `Gym.instructor_hours` method. You may assume all other methods are implemented correctly.

```
def instructor_hours(self, time1: datetime, time2: datetime) -> \
        Dict[int, int]:
    """Return a dictionary reporting the hours worked by instructors
    between <time1> and <time2>, inclusive. Every instructor in this gym
    should be included.

    Each key is an instructor ID and its value is the total number of hours
    worked by that instructor between <time1> and <time2>. Both <time1> and
    <time2> specify the start time for an hour when an instructor may have
    taught.

    Precondition: time1 < time2
    """
```

Diane has implemented the method as follows:

```
def instructor_hours_diane_version(self, time1: datetime, time2: datetime) -> \
        Dict[int, int]:
    result = {}

    for instr_id in self._instructors:
        result[instr_id] = 0
        for time in self._schedule:
            if time1 <= time <= time2:
                result[instr_id] += 1

    return result
```

Misha has implemented the method as follows:

```
def instructor_hours_misha_version(self, time1: datetime, time2: datetime) -> \
        Dict[int, int]:
    result = {}

    for instr_id in self._instructors:
        result[instr_id] = 0

    for time in self._schedule:
        for room_name in self._schedule[time]:
            instr_id = self._schedule[time][room_name][0].get_id()
            result[instr_id] += 1

    return result
```

Midterm Test

CSC148H1 / LEC0301 — Badr

Winter 2020 — Duration: **110 minutes**

Part (a) [3 marks]

Consider a gym with the following instructors and schedule. We are showing only the relevant subset of the data.

Instructor	
ID	Name
25	Mario

Schedule	
Start Time	Instructor
2020-02-14 09:00:00	Mario
2020-02-14 10:00:00	Mario

Assume that a complete instance of `gym` has been created containing this data, and that all representation invariants are satisfied. In the table below, three test cases are described for this instance of `gym`, each with different values for `time1` and `time2`. Fill in the expected return value, and what the actual return values are from Diane's and Misha's implementations.

Arguments		Return Value		
time1	time2	Expected	Diane's version	Misha's version
2020-02-14 10:00:00	2020-02-14 11:00:00	{ 25: 1 }		
2020-02-14 08:00:00	2020-02-14 09:00:00			
2020-02-14 08:00:00	2020-02-14 11:00:00			

Part (b) [3 marks]

Create a **new** test case that will fail using Diane's version of the method. First, fill in the tables below with all instructors that belong to the gym and with its schedule. You do not have to use all the rows. Second, write a Pytest that implements your new test case. No docstring is required.

Instructor	
ID	Name

Schedule	
Start Time	Instructor

```
def ____________________________________________________________________________ -> None:
    g = create_gym()  # Assume that g is a gym object as described in your tables above.

    # All datetime arguments are integers in this order: year, month, day, hour, minute
    t1 = datetime(______, ______, _______, _______, 0)
    t2 = datetime(______, ______, _______, _______, 0)
```

Question 4. [5 MARKS]

Part (a) [1 MARK]

State one advantage of making some instance attributes in a class private.

Part (b) [1 MARK]

Suppose we are using the debugger in PyCharm and have paused execution on a line of code. Describe two different ways we can continue program execution, saying for each where the program would stop.

Part (c) [1 MARK]

Consider a Python list of 1000 items. Which is faster? Circle one answer:

1. Insertion at the front.
2. Insertion at the back.
3. Neither. They take about the same amount of time.

Part (d) [1 MARK]

Consider the `LinkedList` class defined on the aid sheet. Suppose we implement a Stack class using an instance of `LinkedList` where the bottom of the stack is at the front of the linked list, and we have a stack with 1000 items. Which stack operation is faster? Circle one answer:

1. push
2. pop
3. Neither. They take about the same amount of time.

Part (e) [1 MARK]

What is the value of `x` just before the following code terminates?

```
x = [10, 0, 1, 4]
try:
    x[2] = x[4] / x[1]
except AttributeError:
    x.append(3)
except IndexError:
    x[1] = 5
except ZeroDivisionError:
    x.extend([2, True])
except NotImplementedError:
    x = 'not implemented'
except Exception:
    x = None
```

Question 5. [6 MARKS]

Below is a function that is missing a docstring. You will provide pieces that belong in the docstring.

```
def weird(s: Stack) -> Tuple:
    temp = Stack()
    while not s.is_empty():
        temp.push(s.pop())
    answer = (temp.pop(), temp.pop())
    while not temp.is_empty():
        s.push(temp.pop())
    return answer
```

Part (a) [2 MARKS]

What preconditions are needed to ensure the function does not raise an error. If none are necessary, write “None".

Part (b) [2 MARKS]

Write an English description of the function, suitable for the docstring. Assume all preconditions are met.

Part (c) [2 MARKS]

Give one doctest example that uses a stack of size 3 and demonstrates all aspects of what the function does. You may write your answer in two columns if necessary.

Question 6. [7 MARKS]

The method below is to be added to the `LinkedList` class on the aid sheet. Complete this method according to its docstring. Add code only where indicated by dotted lines, and do not change any of the existing code. Complete the "we know that" comment with whatever we can infer must be true about the variables.

```
    def chop_front(self, n: int) -> int:
        """Remove the first n nodes from this linked list, if possible.
        If n > the length of this linked list, remove all the nodes.

        Return the number of nodes that were removed.

        precondition: n >= 1

        >>> linky = LinkedList([1, 2, 3, 4, 5, 6, 7, 8])
        >>> linky.chop_front(3)
        3
        >>> print(linky)
        [4 -> 5 -> 6 -> 7 -> 8]
        >>> linky = LinkedList([22, 1, 5, 4, 7, 4])
        >>> linky.chop_front(13)
        6
        >>> print(linky)
        []
        """
        if ____________________________________________________:

            return 0

        else:

            curr = ____________________________________________________

            i = 0

            while ____________________________________________________:

                curr = ____________________________________________________

                i += 1

            # We know that: ____________________________________________________

            # Add below whatever is needed to complete the method.
```

Question 7. [5 MARKS]

Consider a nested list of strings. It is just like the nested lists of integers that we have worked with, except that the items in it are of type `str`.

Write the body of the following function according to its docstring.

```
def any_longer_than(obj: Union[str, List], n: int) -> bool:
    """Return True iff at least one string in <obj> has a length greater
    than n.

    >>> any_longer_than('it', 3)
    False
    >>> any_longer_than('star', 3)
    True
    >>> any_longer_than([], 2)
    False
    >>> any_longer_than(['a', 'b', 'frog'], 2)
    True
    >>> any_longer_than(['told', ['me', 'the', ['world']], [['is', []], 'gonna']], 2)
    True
    """
```

Midterm Test

CSC148H1 / LEC0301 — Badr

Winter 2020 — Duration: **110 minutes**

Use this page for rough work. If you want the work on this page to be marked, indicate this clearly at the location of the original question.

Midterm Test

CSC148H1 / LEC0301 — Badr

Winter 2020 — Duration: **110 minutes**

Use this page for rough work. If you want the work on this page to be marked, indicate this clearly at the location of the original question.

Midterm Test

CSC148H1 / LEC0301 — Badr

Winter 2020 — Duration: **110 minutes**

Use this page for rough work. If you want the work on this page to be marked, indicate this clearly at the location of the original question.