Midterm Test

CSC 148 H1 / LEC 0101/0102/0201 — Horton and Badr

Winter 2020 — Duration: **110 minutes**

Aids Allowed: **Provided aid sheet**

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 34 marks and consists of 7 questions on 13 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!

Question 1.

Here's the start of some code for managing rooms in the department of Computer Science.

```
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 = []

    def give_key_access(self, person: str) -> None:
        """Record that <person> has key access to this room, if they don't
        already have access.
        """
        if person not in self.users:
            self.users.append(person)

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
    """
    owner: str

    def __init__(self, owner: str) -> None:
        Room.__init__(self)
        self.owner = owner

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:
        Room.__init__(self)
        self.capacity = capacity
```

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.

```
>>> inside = {'c': 6}
>>> l1 = [{'hi': 3, 'bye': 1}, {}, inside]
>>> l2 = [inside]
>>> id(inside) == id(l2)


>>> id(inside) == id(l1[2])


>>> inside['x'] = 9
>>> l1[1]['y'] = 10
>>> l2[0]['z'] = 11
>>> inside


>>> l1


>>> l2
```

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
    """
```

Jaisie has implemented the method as follows:

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

    for time in self._schedule:
        if time1 <= time <= time2:
            for room_name in self._schedule[time]:
                instr_id = self._schedule[time][room_name][0].get_id()
                if instr_id not in result:
                    result[instr_id] = 0
                result[instr_id] += 1

    return result
```

Sophia has implemented the method as follows:

```
def instructor_hours_sophia_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:
            lst = []
            if time1 <= time <= time2:
                for room in self._schedule[time]:
                    instr = self._schedule[time][room][0]
                    lst.append(instr.get_id())

                result[instr_id] = lst.count(instr_id)

    return result
```

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
30	Jonathan

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

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 Jaisie's and Sophia's implementations.

Arguments		Return Value		
time1	time2	Expected	Jaisie's version	Sophia's version
2020-02-14 10:00:00	2020-02-14 11:00:00	{ 25: 0, 30: 1 }		
2020-02-14 08:00:00	2020-02-14 09:00:00			
2020-02-14 13:00:00	2020-02-14 14:00:00			

Part (b) [3 marks]

Create a **new** test case that will fail using Sophia'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]

Suppose all attributes in a class are public and we decide to change the types of some of the attributes to create a more efficient implementation. We have already changed all methods in the class to use the new types properly. What is the key negative consequence that remains?

Part (b) [1 mark]

Under what circumstances do "step into" ( ) and "step over" ( ) do the same thing in the PyCharm debugger?

Part (c) [1 mark]

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

1. Indexing to examine the item at index 0.
2. Indexing to examine the item at index 999.
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 Queue class using an instance of `LinkedList` where the front of the queue is at the front of the linked list, and we have a queue with 1000 items. Which queue operation is faster? Circle one answer:

1. enqueue
2. dequeue
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[0] / 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 puzzle(s: Stack, k: int) -> None:
    s1 = Stack()
    for _ in range(k):
        s1.push(s.pop())
    s2 = Stack()
    while not s.is_empty():
        s2.push(s.pop())
    while not s1.is_empty():
        s.push(s1.pop())
    while not s2.is_empty():
        s.push(s2.pop())
```

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]

Complete this one doctest example so that it demonstrates all aspects of what the function does. You may continue your answer in a second column if needed.

```
    >>> stuff = Stack()
    >>> stuff.push('A')
    >>> stuff.push('B')
    >>> stuff.push('C')
    >>> stuff.push('D')
    >>> puzzle(stuff, 2)
```

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 remove_less(self, value: int) -> int:
        """Remove all nodes from this linked list that contain items less
        than <value>.  Return the sum of the items in the removed nodes.

        Precondition: this linked list contains only integers and is sorted
        in non-descending order.

        >>> linky = LinkedList([-3, 14, 15, 18, 20, 28, 72])
        >>> linky.remove_less(16)
        26
        >>> print(linky)
        [18 -> 20 -> 28 -> 72]
        >>> linky = LinkedList([4, 8, 9])
        >>> linky.remove_less(4)
        0
        >>> print(linky)
        [4 -> 8 -> 9]
        >>> linky = LinkedList([10, 20, 30, 40])
        >>> linky.remove_less(41)
        100
        >>> print(linky)
        []
        """
        curr = ___________________________________________________

        sum = 0

        while ___________________________________________________:

            sum = ___________________________________________________

            curr = ___________________________________________________

        # 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 all_longer_than(obj: Union[str, List], n: int) -> bool:
    """Return True iff all the strings in <obj> have a length greater than n.

    >>> all_longer_than('a', 3)
    False
    >>> all_longer_than('star', 3)
    True
    >>> all_longer_than([], 2)
    True
    >>> all_longer_than(['some', 'body', 'once'], 2)
    True
    >>> all_longer_than(['told', ['me', 'the', ['world']], [['is', []], 'gonna']], 2)
    False
    """
```

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.

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

CSC 148 H1 / LEC 0101/0102/0201 — Horton and 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.

Total Marks = 34