PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

Midterm Test

CSC 148H1S, Winter 2019
Section L0101/L0102 (Horton)

Duration — 110 minutes
Aids allowed — Provided aid sheet

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 8 questions on 14 pages, DOUBLE-SIDED.
- There is a separate aid sheet provided. You do not need to hand it in, and nothing you write on it will be marked.
- 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!

1: _____/ 6
2: _____/ 6
3: _____/ 4
4: _____/ 4
5: _____/ 6
6: _____/ 6
7: _____/ 6
8: _____/ 6

TOTAL: _____/44

Question 1. [6 MARKS]

You are responsible for creating a class to represent a political party. A party has a name and a leader, and accepts donations. Each donor is identified by an integer, such as 253. Here is an example of how we want to use this class.

```
>>> p = Party('Muggle Party of Canada', 'Arthur Weasley')
>>> p.record_donation(253, 150.0)
>>> p.record_donation(987, 25.39)
>>> p.record_donation(253, 300.5)
>>> p.donations_of(253)
450.5
>>> p.donations_of(1)
0.0
>>> p.total_donations()
475.89
```

Below and on the next page 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.

Assume that the appropriate types have been imported from `typing`.

```
class Party:
    """A political party.
    === 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.
```

```
    # TODO: Implement the initializer here.
    # The method header must include a type contract, but a docstring is NOT required.
```

```
    # TODO: Implement method record_donation here.
    # The method header must include a type contract, but a docstring is NOT required.
```

```
    # TODO: Implement method donations_of here.
    # The method header must include a type contract, but a docstring is NOT required.
```

```
    # TODO: Implement method total_donations here.
    # The method header must include a type contract, but a docstring is NOT required.
```

Question 2. [6 MARKS]

We're working on program to play tic-tac-toe. We have decided to store the 3-by-3 game board as a list of lists. Here is the code we have so far:

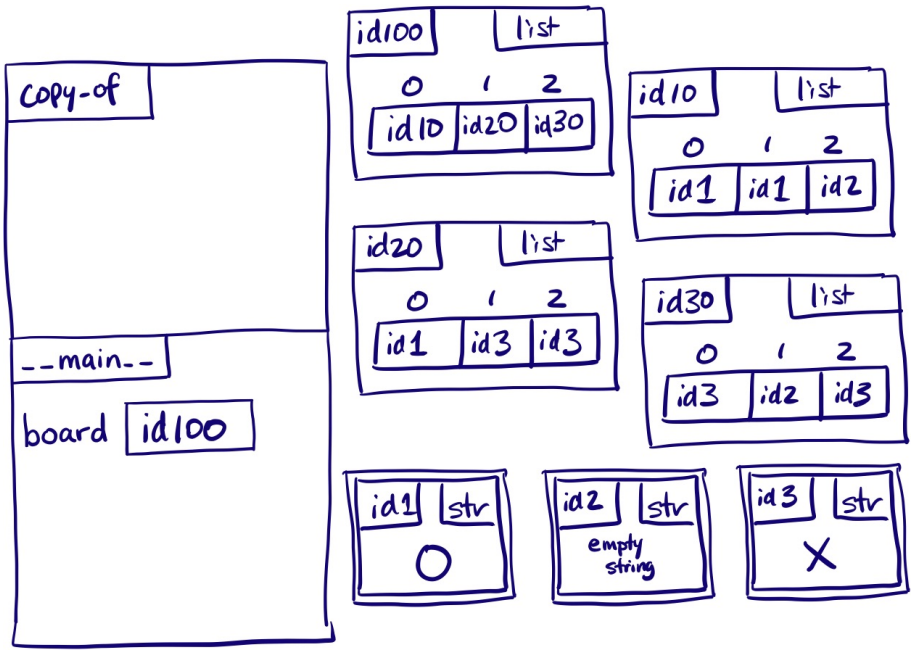
```
def analyze(board: List[List[str]], player: str) -> None:
    """Analyze move options for <player>.

    Modify <board>, putting an int between 0 and 10 inclusive in each spot where
    <player> could move (that is, each empty spot). The higher the number,
    the better the move.

    >>> board = [['O', 'O', ''], ['O', 'X', 'X'], ['X', '', 'X']]
    >>> analyze(board, 'X')
    >>> board[0][2] != ''
    True
    >>> board[2][1] != ''
    True
    """
    # Implementation omitted.

def best_move(board: List[List[str]]) -> Tuple[int, int]:
    """Among all the locations on <board> that hold an int, return the
    (x, y) location of the one with the highest value. If there is a
    tie, return any of the moves tied for highest value.

    >>> best_move([['O', 'O', 10], ['O', 'X', 'X'], ['X', 2, 'X']])
    (0, 2)
    """
    # Implementation omitted.

def copy_of(lst: List[Any]) -> List[Any]:
    """Return a copy of <lst>.

    >>> lst1 = [3, 2, 7]
    >>> lst2 = copy_of(lst1)
    >>> lst2
    [3, 2, 7]
    >>> id(lst2) == id(lst1)
    False
    """
    new_lst = []
    for item in lst:
        new_lst.append(item)
    return new_lst
```

We are happy with the interface and docstring of each function, but the board is getting messed up. The chunk of code on the next page demonstrates the problem:

```
>>> # A nearly full tic-tac-toe board.
>>> board = [['O', 'O', ''], ['O', 'X', 'X'], ['X', '', 'X']]
>>> # Make a copy of the board where move analysis can be recorded without affecting the real board.
>>> temp_board = copy_of(board)
>>> # Analyze all possible moves for player 'O', find the best one, and put the player there.
>>> analyze(temp_board, 'O')
>>> (x, y) = best_move(temp_board)
>>> board[x][y] = 'O'
>>> # Player 'O' has made a move. The board should be otherwise the same. Where did the 2 come from?!
>>> board
[['O', 'O', 'O'], ['O', 'X', 'X'], ['X', 2, 'X']]
```

Part (a) [3 MARKS]

We have begun to trace the call to `copy_of` in the example above. Complete the diagram to show the state of memory immediately before `copy_of` returns. (We assume in the diagram that Python uses the shortcut that allows it to create aliases to the same immutable object.)



Does `copy_of` pass its doctests? Circle one: Yes No

Part (b) [2 MARKS]

How does the 2 get into `board` even though `analyze` is passed the temporary board? Be specific and use correct terminology.

Question 3. [4 MARKS]

Part (a) [2 MARKS]

Which of the following snippets of code from Assignment 1 involve polymorphism? Circle Y or N for each.

Y N
```
# In class PhoneLine:
def new_month(self, month: int, year: int) -> None:
        ... self.contract.new_month(month, year, self.bills[(month, year)]) ...
```

Y N
```
# In class PhoneLine:
def new_month(self, month: int, year: int) -> None:
        ... self.bills[(month, year)] = Bill() ...
```

Y N
```
# In class Contract:
def bill_call(self, call: Call) -> None:
       # This was the whole method body:
       self.bill.add_billed_minutes(ceil(call.duration / 60.0))
```

Y N
```
# In class TermContract:
def __init__(self, start: datetime.date, end: datetime.date) -> None:
        ... super().__init__(start) ...
```

Part (b) [1 MARK]

Define two classes, called Blob and Block that have an inheritance relationship with each other. Each class must have no more than one line of code inside it.

Part (c) [1 MARK]

Below are two very simple classes. Modify the code so that there is a composition relationship between the classes. Hint: You can do this with very few changes.

```
class Monster:
    def __init__(self) -> None:
        self.hunger = 0


class Human:
    def __init__(self, name: str) -> None:
        self.name = name
```

Question 4. [4 MARKS]

Recall the Queue class that we have used in the course. You can find an implementation of it on the provided aid sheet.

Suppose we want to define a new kind of Queue called a PairQueue. It works like a regular Queue, and has the same public interface, except that it will dequeue either one or two items, to guarantee that there are an even number of items remaining in the queue. For example, if there are 6 items in the queue, it dequeues two items, so 4 remain. If there are 9 items in the queue, it dequeues only one item, so 8 remain. If there are no items in the queue, it ~~raises an error~~ behaves just as a regular Queue does.

If two items are dequeued, they are returned as a tuple. If only one item is dequeued, the item itself is returned (not in a tuple).

Write this new class. Part of the marks will be for good design. In particular, avoid repeated code, even a single line.

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

Question 5. [6 MARKS]

Write this function (outside any class) to turn a Queue into a "palindrome," something that reads the same in one direction as the other. That is, the Queue should be changed to have its original items, followed by its original items again, but in reverse order.

You may make use of the Stack and Queue classes defined on the provided aid sheet to create temporary Stack and/or Queue objects, but you should not create any other new objects such as lists or dictionaries.

You must not access the instance attributes of any Stack or Queue. Use the public interface instead.

```
def make_palindrome(q: Queue) -> None:
    """Modify <q> so that, at the back, it contains a second copy of its items, in reverse order.
    Thus, if the queue is emptied, the items come out in the form of a palindrome (the same forwards
    and backwards).

    >>> stuff = Queue()
    >>> stuff.enqueue(1)
    >>> stuff.enqueue(2)
    >>> stuff.enqueue(3)
    >>> make_palindrome(stuff)
    >>> stuff.dequeue()
    1
    >>> stuff.dequeue()
    2
    >>> stuff.dequeue()
    3
    >>> stuff.dequeue()
    3
    >>> stuff.dequeue()
    2
    >>> stuff.dequeue()
    1
    """
```

Question 6. [6 MARKS]

Suppose we implement the queue ADT using just one instance attribute — an instance of the LinkedList class as defined on the provided aid sheet — and we put the front of the queue at the front of the LinkedList.

Part (a) [3 MARKS]

What will be the big-oh time-complexity of calling the enqueue method on a queue containing n items?

Explain.

What will be the big-oh time-complexity of calling the dequeue method on a queue containing n items?

Explain.

Part (b) [2 MARKS]

Say we can add a single new instance attribute to the LinkedList class. What would you recommend in order to improve the performance of your Queue class?

What Queue method could be rewritten to run faster? _____

What would be its new time-complexity, in big-oh terms? _____

Part (c) [1 MARK]

In terms of big-oh, which change best improves the performance of your Queue class? Circle one.

1. Using the new instance attribute from Part (c)
2. Changing your queue implementation to use a Python list.

Question 7. [6 MARKS]

The method below is being added to the LinkedList class on the provided aid sheet. Assume the class also has a `__str__` and a `__len__` method defined.

Fill in the boxes with the necessary code to complete the method according to its docstring. Do not add any code outside of the boxes. You must not create any new Node objects.

```
def swap_halves(self) -> None:
    """Move the nodes in the second half of this list to the front.

    Precondition: len(self) >= 2

    >>> lst = LinkedList([5, 10, 15, 20, 25, 30])
    >>> print(lst)
    [5 -> 10 -> 15 -> 20 -> 25 -> 30]
    >>> lst.swap_halves()
    >>> print(lst)
    [20 -> 25 -> 30 -> 5 -> 10 -> 15]
    >>> lst = LinkedList([5, 10])
    >>> lst.swap_halves()
    >>> print(lst)
    [10 -> 5]
    >>> lst = LinkedList([5, 10, 15, 20, 25])
    >>> lst.swap_halves()
    >>> print(lst)
    [15 -> 20 -> 25 -> 5 -> 10]
    """
    # Compute the index of the node that will be the new first node.
    mid_index = len(self) // 2

    # Set first_end to refer to the node at the end of the first half
    first_end = [                    ]
    pos = 0
    while pos < mid_index - 1:
        first_end = [                    ]
        pos += 1

    # Set second_end to refer to the node at the end of the second half
    second_end = [                    ]
    while second_end.next is not None:
        second_end = [                    ]

    # Swap the halves
    second_end. [                    ] = [                    ]
    self. [                    ] = [                    ]
    first_end. [                    ] = [                    ]
```

Question 8. [6 MARKS]

Write the body of the following recursive function according to its docstring. You can (and should!) use the Recursive code template from the provided aid sheet as a starting point.

```
def count_matches(obj: Union[int, List], n: int) -> int:
    """Return the number of times that n occurs in obj.

    >>> count_matches(100, 100)
    1
    >>> count_matches(100, 3)
    0
    >>> count_matches([10, [[20]], [10, [10]]], 10)
    3
    >>> count_matches([10, [[20]], [10, [10]], 20)
    1
    >>> count_matches([10, [[20]], [10, [10]], 30)
    0
    """
```

[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.]

[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.]

[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.]