UNIVERSITY OF TORONTO
Faculty of Arts and Science

Term test #2

CSC148H1, Section L5101
Duration – 50 minutes

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

Last Name: _____

First Name: _____

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

This test consists of 4 questions on 12 pages (including this one). There is a bonus question on Page 12. Pages 13 to 22 are Python reference sheets including classes that we developed in lectures/labs. You may use any of the classes and functions from the reference sheets in your answers.

When you receive the signal to start, please make sure that your copy of the test is complete.

Please answer questions in the space provided. You will earn 20% for any question you leave blank or write "I cannot answer this question," on. We think we have provided a lot of space for your work, but please do not feel you need to fill all available space.

Write neatly and concisely. If we cannot read it, we cannot grade it.

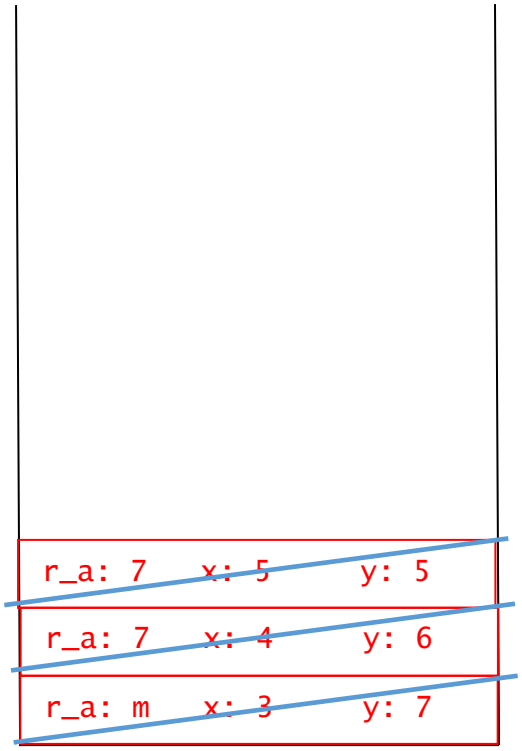
GOOD LUCK!

Question 1. Tracing Recursive Functions. [10 Marks]

Read the following code for the recursive function `mystery`.

```
def mystery(x: int, y: int):
    if x > y:
        return mystery(y, x-1)
    elif x == y:
        return y
    else:
        return mystery(x+1, y-1)
```

a) Trace `mystery(3, 7)`. Show the push and pop operations in the following stack of activation records, where `r_a` represents the return address, and `x` and `y` are the arguments. What is the final returned value from `mystery(3, 7)`? 5 **[4 Marks]**



stack of activation records

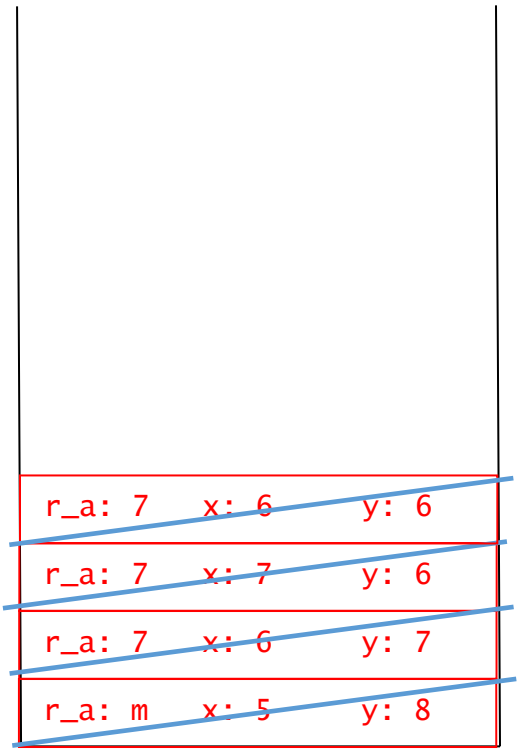
r_a:	x:	y:

Marking scheme:

-1 if the final returned value is wrong
-1 for every missing or wrong push/pop

Question 1. continued

b) Trace `mystery(5, 8)`. Show the push and pop operations in the following stack of activation records, where `r_a` represents the return address, and `x` and `y` are the arguments. What is the final returned value from `mystery(5, 8)`? 6 **[4 Marks]**



stack of activation records

r_a:	x:	y:

Observation: `mystery` finds the integer midpoint of `x` and `y`

Marking scheme:

-1 if the final returned value is wrong
-1 for every missing or wrong push/pop

Question 1. continued

c) Consider the following recursive function.

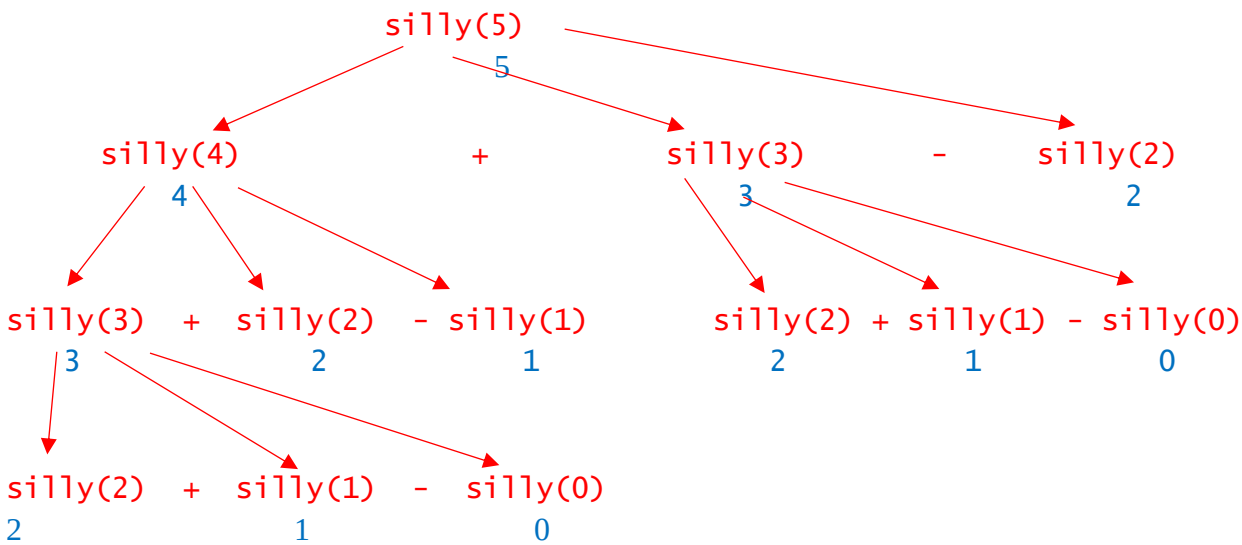
```
1. def silly(x: int):
2.     # pre-condition: x is a non-negative integer
3.     if x < 3:
4.         return x
5.     else:
6.         return silly(x-1) + silly(x-2) - silly(x-3)
```

When `silly(5)` is called,

what is the final returned value? 5 **[1 Mark]**

how many times the base case (line 4) is executed? 9 **[1 Mark]**

Note: Feel free to trace the `silly` function intuitively, formally (using a stack), or via any other approach you are comfortable with. We only grade your final answers in above boxes.



Marking scheme:
-1 for any wrong final answer

Note: leaf nodes represent the base case

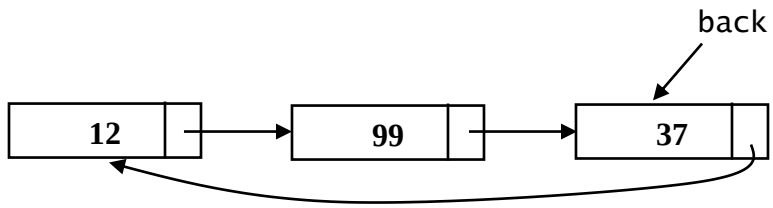
Observation: `silly(x)` returns x

Question 2. Circularly Linked Lists **[10 Marks]**

Consider the following class `CircularLinkedList`.

```
import LinkedListNode

class CircularLinkedList:
    """
    Circular collection of LinkedListNodes

    === Attributes ==
    :param: back: the lastly appended node of this CircularLinkedList
    :type back: LinkedListNode
    """

    def __init__(self, value):
        """
        Create CircularLinkedList self with data value.

        :param value: data of the front element of this circular linked list
        :type value: object
        """
        self.back = LinkedListNode(value)
        #back.next_ corresponds to front
        self.back.next_ = self.back

    def __str__(self):
        """
        Return a human-friendly string representation of CircularLinkedList self
        :rtype: str

        >>> lnk = CircularLinkedList(12)
        >>> str(lnk)
        '12 ->'
        """

        #back.next_ corresponds to front
        current = self.back.next_
        result = "{} ->".format(current.value)
        current = current.next_
        while current is not self.back.next_:
            result += " {} ->".format(current.value)
            current = current.next_
        return result

    def append(self, value):
        """
        Insert value before LinkedList front, i.e. self.back.next_.

        :param value: value for new LinkedList.front
        :type value: object
        :rtype: None

        >>> lnk = CircularLinkedList(12)
        >>> lnk.append(99)
        >>> lnk.append(37)
        >>> print(lnk)
        12 -> 99 -> 37 ->
        """
        self.back.next_ = LinkedListNode(value, self.back.next_)
        self.back = self.back.next_
```

In a circularly linked list, all nodes are linked in a continuous circle, i.e. once the list is created, no node references to a `None` next. As an example, in the following figure, the front of the list contains 12, and the last element appended to the list contains 37.



Read the following `docstring`. Then, implement method `reverse_print` for class `CircularLinkedList`, explained above in Page 5 .

```
def reverse_print(self, current):
    """
    Print the values of a linked list in reverse order, i.e. from back of
    the list to the node current

    :param current: a node in self
    :type current: LinkedListNode
    :rtype: None

    >>> lnk = CircularLinkedList(12)
    >>> lnk.append(99)
    >>> lnk.append(37)
    >>> lnk.reverse_print(lnk.back.next_)
    37
    99
    12
    >>> lnk.reverse_print(lnk.back)
    37
    """
    # to be developed by you

    if current is not self.back:
        self.reverse_print(current.next_)
    print(current.value)
```

Use the space on this “blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Other solutions could be based on 1) using a stack 2) using a Python list

If the solution is based on recursion (without using explicit loops or stacks), we give +2 bonus points.

Marking scheme:

-1 return statement
-1 no print statement
-2 wrong list traversal mechanism
-1 bad method call or use
-2 no base case if recursion or does not check if current is reached
-4 if printed list forward instead of backward
-1 for bad error values
-1 for off by 1
-2 mix while loop with recursion
-1 for checking values instead of node (problems with duplicates)
-1 bad syntax
-2 for hardcoding for example
-4 for bad loop structure or doesn't do anything with the nodes

Question 3. General Trees **[10 Marks]**

A **full *n*-ary tree** is a general tree in which all *internal* nodes have exactly *n* children. Read the following `docstring` for function `is_full(t, n)`. Then, implement it.

```
from tree import Tree

def is_full(t, n):
    """
     Return whether tree t is full with respect to arity n

    :param t: a general tree
    :type t: Tree
    :param n: a given branching factor for the tree
    :type n: int
    :rtype: bool

    >>> t1 = descendants_from_list(Tree(0), [], 3)
    >>> is_full(t1, 3)
    True
    >>> t2 = descendants_from_list(Tree(0), [1, 2, 3], 3)
    >>> is_full(t2, 3)
    True
    >>> t3 = descendants_from_list(Tree(0), [1, 2, 3, 4], 3)
    >>> is_full(t3, 3)
    False
    >>> t4 = descendants_from_list(Tree(0), [1, 2, 3, 4, 5, 6], 3)
    >>> is_full(t4, 3)
    True
    """
    # to be developed by you

    if len(t.children) == 0:
        return True
    elif len(t.children) != n:
        return False
    else:
        return all([is_full(c,n) for c in t.children])
```

Marking scheme:

+3 for each base case
+4 for the recursive case

-1 for each minor mistakes
-5 for major mistakes such as an iterative solution that is not in a right track

Use the space on this “blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Question 4. Binary Trees **[10 Marks]**

Provide a function, named `max_value(t)`, to return the maximum value in the binary tree `t`.

Examples: `max_value(t1)` returns 9; `max_value(t2)` returns 12; `max_value(t3)` returns 10; `max_value(t4)` returns 6;

```
t1                t2                t3              t4
  \                 \                 \               \
    8                 12                5               6
   / \               /                 / \
  2   9             10                3   8
 / \                                 / \
1   7                               10  10
   /
  4
```

You may assume that class `BinaryTree` from the API have been imported. Provide both the `docstring` and implementation of `max_value(t)`.

```python
from binary_tree import BinaryTree

def max_value(t):

    """
    Return the max value in BinaryTree t

    :param t: a not None binary tree
    :type t: BinaryTree
    :return: the maximum value in the tree
    :rtype: object

    >>> t1 = BinaryTree(8)
    >>> max_value(t1)
    8
    >>> t2=BinaryTree(8,BinaryTree(7,BinaryTree(12),BinaryTree(5)),BinaryTree(11))
    >>> max_value(t2)
    12
    """
    if t.left is None and t.right is None:
        return t.data
    if t.left and t.right is None:
        return max(t.data, max_value1(t.left))
    if t.left is None and t.right:
        return max(t.data, max_value1(t.right))
    else:
        return max([t.data, max_value1(t.left), max_value1(t.right)])
```

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Marking scheme:

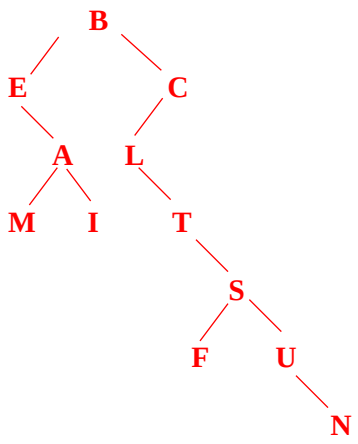
+5 for docstring:
- +1 brief description
- +2 type contract
- +1 examples
- +1 rtype

+5 for implementation
- +2 base case
- +3 recursive case

- -0.5 use wrong variable name, member name
- -0.5 improper use of max() function
- -0.5 other grammar issue
- -1 fail to compare the left_max and right_max values with the root value
- -1 fail to check the existence of left or right children
- -1 loop over t.children instead of t.left and t.right
- -1 use 0 as the initial value before comparison (fails in the test cases which contain negative values)

Bonus Question. Tree Traversals **[2 Marks]**

Draw a binary tree for which the *preorder* traversal is `BEAMICLTSFUN` and the *inorder* traversal is `EMAIBLTFSUNC`.



Marking scheme:

-0.5 for one mistake
-1 for two mistakes
0 for three or more mistakes

Question	Initial	Mark
1		/10
2		/10
3		/10
4		/10
Bonus		/2
Total		/40