UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2016 EXAMINATIONS

CSC148H1F

Duration - 3 hours

Instructors: Diane Horton and David Liu

No Aids Allowed

Name:

Student Number:

Please read the following guidelines carefully!

- Please print your name and student number on the front of the exam.
- This examination has **10** questions. There are a total of **22 pages, DOUBLE-SIDED**.
- The last two pages are an aid sheet that may be detached.
- You may always write helper functions/methods unless explicitly asked not to.
- Docstrings are *not* required unless explicitly asked for.
- You must earn a grade of **at least 40% on this exam to pass this course**.

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.

It's been a real pleasure
teaching you this term.

Good luck!

Question	Grade	Out of
Q1		11
Q2		6
Q3		6
Q4		5
Q5		14
Q6		10
Q7		11
Q8		8
Q9		4
Q10		5
Total		80

1. **[11 marks]** The following questions test your understanding of the terminology and concepts from the course. You may answer in either point form or full sentences; **you do not need to write much to get full marks**!

   (a) **[2 marks]**
   Name one **mutable** and one **immutable** data type.

   Mutable type:

   Immutable type:

   What is the difference between mutable and immutable types?

   (b) **[2 marks]**
   Give one real-world example of something that we could model using a stack, and briefly explain why a stack is appropriate.

   Give one real-world example of something that we could model using a queue, and briefly explain why a queue is appropriate.

   (c) **[2 marks]**
   Suppose we use a linked list to implement a stack. Would it be faster to represent the top of the stack as the *front* of the list, or as the *end* of the list?

   Circle one: front end

   Explain your answer.

   (d) **[2 marks]**
   Define the term **abstract class**, and give an example of one that you have seen in this course.

(e) [**3 marks**]
Explain the difference between **inheritance** and **composition**.

In English, give an example where it would be appropriate to use inheritance. Do not write any code.

In English, give an example where it would be appropriate to use composition. Do not write any code.

2. **[6 marks]** For each of the following code snippets, write its Big-Oh worst-case running time below it. **No explanation is necessary**.

```
# Given an int m > 0 and
# list L of length n > 0.
for i in range(m):
    L.append(i+1)
sum = 0
for i in range(len(L)):
    sum = sum + L[i]
```

Worst-case running time:

```
# Given a list L of length n > 0.
n = len(L)
for i in range(n):
    if L[i] > 0:
        for j in range(n):
            answer = answer + L[i]
    else:
        for j in range(n):
            answer = answer - L[i]
```

Worst-case running time:

```
# Given lists L1 of length a > 0
# and L2 of length b > 0.
if len(L1) > len(L2):
    for k in range(len(L1)):
        L2[0] += L1[k]
else:
    for k in range(len(L1)):
        for j in range(len(L1)):
            L2[0] += L1[k] + L2[j]
```

Worst-case running time:

```
# Given a list L of length b > 0.
i = 1
while i < len(L) and L[i] <= L[0]:
    i += 1
```

Worst-case running time:

```
# Given positive integers n and m.
answer = 0
for a in range(n):
    for b in range(100):
        answer = a + m
```

Worst-case running time:

```
# Given an int n > 0, and a list L of length a > 0.
for i in range(n):
    L.insert(0, n)
    L.pop(0)
```

Worst-case running time:

3. **[6 marks]** Explain what each of the following error messages means, and write a short program that generates that error when run.

(a) `AttributeError: 'NoneType' object has no attribute 'x'`

Meaning:

```
if __name__ == '__main__':
    # Write code here that generates the error.
```

(b) `RecursionError: maximum recursion depth exceeded`

Meaning:

```
# Define whatever you need to generate the error in the main block.
```

```
if __name__ == '__main__':
    # Write code here that generates the error.
```

(c) `NotImplementedError`

Meaning:

```
# Define whatever you need to generate the error in the main block.
```

```
if __name__ == '__main__':
    # Write code here that generates the error.
```

4. **[5 marks]**

(a) **[2 marks]**
Consider this small program:

```
def double(x):
    print(id(x))
    x = x + x
    print(id(x))
    return x

if __name__ == '__main__':
    x = [27, 10]
    print(id(x))
    x = double(x)
    print(id(x))
```

Assume `id` returns a four-digit number. Which of the following could be the output from this program? Circle yes or no for each option. **No explanation is necessary.**

```
7392            7392            7392            7392
8256            7392            1234            8256
7392            8256            9876            8256
8256            8256            8258            7392
```

Yes No   Yes No   Yes No   Yes No

(b) **[3 marks]**
Consider this program.

```
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add_to_x(self, amount):
        # Add <amount> to this x attribute.
        self = A(self.x + amount, self.y)

if __name__ == '__main__':
    my_a = A(10, 20)
    print(my_a.x)
    my_a.add_to_x(13)
    print(my_a.x)
```

Write what this program would output, or describe the error that would occur. **Hint:** `self` behaves like any other parameter.

In the space below, explain your answer in two or three sentences.

5. **[14 marks]** Suppose you are designing a class to implement a spreadsheet. It must support the following:

- A spreadsheet is a table of numbers.
- A spreadsheet has 0 or more rows and 1 or more columns.
- Each spot in the spreadsheet (at a particular row and column) is called a “cell”.
- Each cell either contains no value, or contains an integer value.
- Each column has a name, such as “Unit Price”, and no two column names can be the same.
- You can set the value of a cell.
- You can compute the average value of the cells in a column.
- You can determine the number of rows in a spreadsheet.

You should not design any other classes for this question.

(a) **[2 marks]**

Write client code that creates a spreadsheet with columns “Year”, “Sales”, and “Expenses”. Record in the spreadsheet that in 2015, sales were 125,000 and in 2016 Sales were 150,000, and expenses in both years were 100,000. Then print out the average value in column “Sales”.

Your client code must **not** access any instance attributes; it should only call methods. Use your judgment to pick good method names and appropriate parameters. Where behaviour is not specified in our description, make a reasonable choice.

(b) [**6 marks**]

Define the interface for the public methods of this class. Your documentation must support all required behaviours given in the problem description, and should be consistent with your client code from part (a).

You should write complete method signatures and docstrings, but doctests are not required. **Do not** implement any of these methods. Do not write a class docstring (yet).

(c) **[3 marks]**

Suppose we are going to store the content of a spreadsheet in a Python list of lists. Each element of the list represents a row of the spreadsheet. We will keep a “parallel” list for the column names. For the example spreadsheet described in part (a), these two attributes would have the following values:

```
# the rows
[[2015, 125000, 100000], [2016, 150000, 100000]]
# the column names
["Year", "Sales", "Expenses"]
```

Write the portion of the class docstring that documents these private attributes of the class.

(d) **[3 marks]**

Write the portion of the class docstring that expresses all appropriate representation invariants of the class.

6. **[10 marks]** In this question, you will implement a `LinkedList` method and analyse its running time.

   (a) **[8 marks]**
   Implement the following `LinkedList` method. You may not use any other `LinkedList` methods other than `is_empty`.

```
def insert_list(self, other, i):
    """Insert a *copy* of <other> into this list immediately after position <i>.

    Do NOT mutate <other>, not even its nodes.

    Precondition: 0 <= i < len(self)
      Hint: this means that <self> is not empty. But <other> could be empty.

    @type self: LinkedList
    @type other: LinkedList
    @type i: int
    @rtype: None

    >>> linky = LinkedList([0, 1, 2, 3, 4])
    >>> other = LinkedList([100, 101])
    >>> linky.insert_list(other, 3)
    >>> print(linky)
    [0 -> 1 -> 2 -> 3 -> 100 -> 101 -> 4]
    >>> print(other)
    [100 -> 101]
    """
```

(b) **[2 marks]**

Let n be the length of `self`, and m be the length of `other`. What is the Big-Oh worst-case running time of your implementation of `insert_list`, in terms of n and m?

Extra space is provided below for rough work. Your answer to the previous question part should not be very long.

7. **[11 marks]** Consider this method in class `LinkedList`:

```
def mystery(self):
    if len(self) < 2:
        return None
    else:
        previous = self._first
        current = previous.next
        while current is not None and current.item >= previous.item:
            previous = current
            current = current.next

        # (A) What do we know at this line?
        if current is not None:
            # (B) What do we know at this line?
            return current.item
        else:
            # (C) What do we know at this line?
            return None
```

(a) **[3 marks]**
What do we know about the state of `previous` and `current` at the following point in the code:

(A)

What do we know about the state of `previous`, `current`, and all the nodes in the linked list up to and including `current` at the following points in the code:

(B)

(C)

(b) **[3 marks]**
Write a good docstring for this method. Include one doctest on a linked list with five elements.

(c) **[1 mark]**

Would it make any difference to method's behaviour if we were to reverse the order of the two conditions on the `while` loop? Circle one. Yes No

Explain.

(d) **[2 marks]**

Suppose we have implemented the `__len__` method for `LinkedList` by storing an extra "length" instance attribute, so that calling `len(self)` always takes constant time.

What is the *best-case* running time for `mystery` on a list of length n? Write the Big-Oh expression.

Explain your answer.

(e) **[2 marks]**

Using the same assumption as in part (d), what is the *worst-case* running time for `mystery` on a list of length n? Write the Big-Oh expression.

Explain your answer.

8. [**8 marks**] Consider the following BinarySearchTree method.

```
    def distribution(self):
        """Return the distribution of values in this binary search tree.

        Precondition: every value in this binary search tree is an int.

        @type self: BinarySearchTree
        @rtype: dict[int, int]
            Each key is a number that occurs in this binary search tree,
            and the corresponding value is the number of occurrences
            of this number.

        >>> bst = BinarySearchTree(None)
        >>> bst.insert(39)
        >>> bst.insert(39)
        >>> bst.insert(-4)
        >>> bst.insert(39)
        >>> bst.insert(105)
        >>> bst.insert(-4)
        >>> bst.distribution() == {39: 3, -4: 2, 105: 1}
        True
        """
```

Your task is to implement distribution on the following page. Your implementation should make use of the following helper:

```
def absorb(d1, d2):
    """Absorb the contents of d2 into d1. Do not mutate d2.

    For each key in d2 that is not in d1, add the key and its value to d1.
    For each key in d2 that is in d1, add its value in d2 to the value for it in d1.

    @type d1: dict[object, int]
    @type d2: dict[object, int]
    @rtype: None

    >>> d1 = {1: 1, 2: 2, 3: 3}
    >>> d2 = {3: 10, 10: 10}
    >>> absorb(d1, d2)
    >>> d1 == {1: 1, 2: 2, 3: 13, 10: 10}
    True
    >>> d1 = {}
    >>> d2 = {50: 60}
    >>> absorb(d1, d2)
    >>> d1 == {50: 60}
    True
    """
```

Write your `distribution` *method here. Use the helper function from the previous page – assume that it has been implemented correctly.*

```
    def distribution(self):
        """The docstring is on the previous page."""
```

9. [**4 marks**] Here is a recursive method that is intended to return the sum of the numbers in a nested list.

```
def nested_sum(obj):
    """Return the sum of the numbers in a nested list.

    Note that a nested list is one of two things:
      1. a number
      2. a list of (smaller) nested lists

    @type obj: int | list
    @rtype: int

    >>> nested_sum([4, [1, 2, 3], [10, [20]], 4])
    44
    """

    sum = 0

    if isinstance(obj, int):

        sum = sum + obj

    else:

        for lst_i in obj:

            nested_sum(lst_i)

        return sum
```

Unfortunately, this method does not work as intended. Describe or define a list on which this method would fail:

Describe what would be returned, or what error would occur, if we called `nested_sum` on this list.

Fix the problem(s) by writing changes directly on the code above.

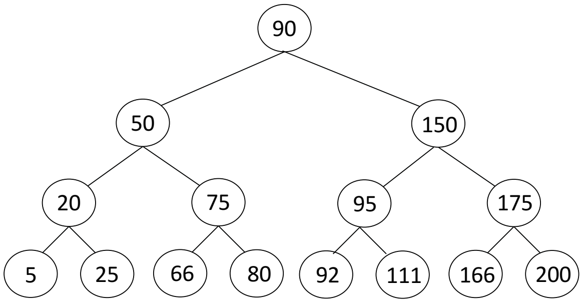
10. [**5 marks**] **Warning**: this question is tougher, and not worth as many marks. We recommend attempting it last.

Consider the following `BinarySearchTree` method `levels`, which you saw before on an exercise.

```
def levels(self):
    """Return a list of items in the tree, separated by level.

    @type self: BinarySearchTree
    @rtype: list[(int, list)]
    """
```

This method returns a list of tuples of the form `(d, items)`, where `d` is a positive integer and `items` is a sorted list of the items in the BST at depth `d`. For example, calling `levels` on this BST:



should produce the following list:

```
[(1, [90]), (2, [50, 150]), (3, [20, 75, 95, 175]), (4, [5, 25, 66, 80, 92, 111, 166, 200])]
```

The returned list should be sorted in ascending order of `d` values, and should have exactly h elements, where h is the height of the tree.

In the space below, implement `levels` recursively. You may not define or use any helper methods other than `is_empty`.

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.*

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.*

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.*

Basic operators

```
True and False, True or False, not True
1 + 3, 1 - 3, 1 * 3
5 / 2 == 2.5, 5 // 2 == 2, 5 % 2 == 1
'hi' + 'bye'            # 'hibye'
[1, 2, 3] + [4, 5, 6]  # [1, 2, 3, 4, 5, 6]
```

List methods

```
lst = [1, 2, 3]
len(lst)             # 3
lst[0]               # 1
lst[0:2]             # [1, 2]
lst[0] = 'howdy'     # lst == ['howdy', 2, 3]
lst.append(29)       # lst == ['howdy', 2, 3, 29]
lst.pop()            # lst == ['howdy', 2, 3], returns 29
lst.pop(1)           # lst == ['howdy', 3], returns 2
lst.insert(1, 100)   # lst == ['howdy', 100, 3]
lst.extend([4, 5])   # lst == ['howdy', 100, 3, 4, 5]
3 in lst             # returns True
```

Dictionary methods

```
d = {'hi': 4, 'bye': 100}
d['hi']            # 4
d[100]             # raises KeyError!
'hi' in d          # True
4 in d             # False
d['howdy'] = 15    # adds new key-value pair
d['hi'] = -100     # changes a key-value pair
```

Control flow

```
if x == 5:
    y = 1
elif 4 <= 100:
    z = 2
else:
    y = 100

for i in [0, 1, 2, 3]:     # or, "for i in range(4):"
    print(i)

j = 1
while j < 10:
    print(j)
    j = j * 2
```

Classes

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def size(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5

p = Point(3, 4)    # constructor
p.x                # attribute access: returns 3
p.size()           # method call: returns 5.0

class MyWeirdClass(Point):  # inheritance
    pass
```

Linked List

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

    === Attributes ===
    @type item: object
        The data stored in this node.
    @type next: _Node | None
        The next node in the list, or None if there are
        no more nodes in the list.
    """
    def __init__(self, item):
        """Initialize a new node storing <item>,
        with no 'next' node.

        @type self: _Node
        @type item: object
        @rtype: None
        """

class LinkedList:
    """A linked list implementation of the List ADT.

    === Private Attributes ===
    @type _first: _Node | None
       The first node in the list,
       or None if the list is empty.
    """
    def __init__(self, items):
        """Initialize a linked list with the given items.

        The first node in the linked list contains the
        first item in <items>.

        @type self: LinkedList
        @type items: list
        @rtype: None
        """
```

Exceptions

```
raise IndexError
```

Tree

```
class Tree:
    """
    === Private Attributes ===
    @type _root: object | None
        The tree's root item, or None.
    @type _subtrees: list[Tree]
        A list of all subtrees of the tree.

    === Representation Invariants ===
    - If _root is None then _subtrees is empty.
      This represents an empty Tree.
    - _subtrees doesn't contain any empty trees
    """

    def __init__(self, root, subtrees):
        """Initialize a new Tree with the given root
        and subtrees.

        If <root> is None, the tree is empty.

        @type self: Tree
        @type root: object | None
        @type subtrees: list[Tree]
        @rtype: None
        """

    def is_empty(self):
        """Return whether this tree is empty.
        @type self: Tree
        @rtype: bool
        """
```

Stacks and Queues

```
s = Stack()
s.is_empty()
s.push(10)
s.pop()

q = Queue()
q.is_empty()
q.enqueue(10)
q.dequeue()
```

BinarySearchTree

```
class BinarySearchTree:
    """
    === Private Attributes ===
    @type _root: object | None
        The BST's root value, or None.
    @type _left: BinarySearchTree | None
        The left subtree, or None.
    @type _right: BinarySearchTree | None
        The right subtree, or None.
    === Representation Invariants ===
    - If _root is None, then so are _left and _right.
      This represents an empty BST.
    - If _root is not None, then _left, _right are BSTs.
    - Every item in _left is <= _root, and
      every item in _right is >= _root.
    """
  def __init__(self, root):
       """Initialize a new BST with a given root value.
       If <root> is None, the BST is empty.

       @type self: BinarySearchTree
       @type root: object | None
       @rtype: None
       """

   def is_empty(self):
       """Return whether this tree is empty.

       @type self: BinarySearchTree
       @rtype: bool
       """
```