UNIVERSITY OF TORONTO
Faculty of Arts and Science

Term test #1

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 18 pages (including this one). Pages 13 to 18 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 a stack client function. **[10 Marks]**

Recall the `Stack` class we discussed in lectures and labs (see the reference sheets for details).

Read over function **`list_stack`** below.

```
def list_stack (list_, s):
    """
    :param list_: a Python list
    :type list_ : list
    :param s: an empty stack
    :type s : Stack
    :rtype: None
    """
    for i in list_:
        s.add(i)
    print(s)
    while not s.is_empty():
        el = s.remove()
        if isinstance(el, list):
            for j in el:
                s.add(j)
        else:
            print(el)
        print(s)
```

Assume that `L = ['a', ['b', ['c', 'd']], ['e', 'g'], 'f']` and that **`s`** is an empty **`Stack`**. Write the output of the function call below. (We have written the first line of the output for you.)

`list_stack(L, s)`

```
['a', ['b', ['c', 'd']], ['e', 'g'], 'f']
f
['a', ['b', ['c', 'd']], ['e', 'g']]
['a', ['b', ['c', 'd']], 'e', 'g']
g
['a', ['b', ['c', 'd']], 'e']
e
['a', ['b', ['c', 'd']]]
['a', 'b', ['c', 'd']]
['a', 'b', 'c', 'd']
d
['a', 'b', 'c']
c
['a', 'b']
b
['a']
a
[]
```

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 2. Reusing classes in our code. **[10 Marks]**

Let's define an expression as balanced if its opening and closing brackets (such as "()", "[]", and "{}") match. Your task is to read the following class `Expression` and the docstring for method `is_balanced(…)`, and develop the body of the method.

```
from stack import Stack


class Expression:
    """
    An expression containing brackets, such as (, {, and [

    ===+= Public Attributes =====
    :type content: str
         the string representation of an expression
    """

    def __init__(self, a_str):
        """
        Create a new Expression self with an initial string

        :param a_str: an initial string
        :type a_str: Str
        """
        self.content = a_str

    def is_balanced(self):
        """
        Verify whether or not Expression self is balanced in
        terms of (, [, and {

        :return: True if Expression self is balanced; otherwise return False
        :rtype: bool

        >>> e1 = Expression("(+){ghi[]}")
        >>> e1.is_balanced()
        True
        >>> e2 = Expression("(+))")
        >>> e2.is_balanced()
        False
        >>> e3 = Expression("({+)}")
        >>> e3.is_balanced()
        False
        >>> e4 = Expression("{b)")
        >>> e4.is_balanced()
        False
        >>> e5 = Expression("a")
        >>> e5.is_balanced()
        True
        """
        temp = self.content
        result = True
```

```
        stack = Stack()
        dic = {"(":")", "[":"]", "{":"}"}
        while temp:
            char = temp[0]
            temp = temp[1:]
            if char in ["(","[","{"]:
                stack.add(char)
            elif char in[")", "]", "}"]:
                if not stack.is_empty():
                    stack_top = stack.remove()
                    if not dic.get(stack_top) == char:
                        result = False
                else:
                    result = False
        if not stack.is_empty():
            result = False
        return result
```

Question 3. Designing classes. **[10 Marks]**

Recall that we discussed the `Queue` class in labs (see the reference sheets for details). Here, we introduce two more kinds of queues.

A "double ended queue", called `Deque`, supports element insertion and removal at both ends of a queue. Hence, a `Deque` has all properties of a `Queue`. In addition, a `Deque` has the following methods: `add_to_front(…)` and `remove_from_end(…)` for inserting an element to the front of the underlying queue and removing an element from the end of the underlying queue, respectively.

A "priority queue", called `Pqueue`, is different than a `Queue` in two ways: 1) when a `Pqueue` is being constructed, an attribute of the elements—that will be stored in the underlying queue—is declared for a "priority" of the elements; 2) when the method `remove()` is invoked, an element that has the smallest value in the priority attribute will be removed and returned from `Pqueue`. If there is a tie, for the smallest value, it is broken arbitrarily.

For example, assume attribute *profit* is declared as the priority attribute for a priority queue that will store persons (instances of class `Person` provided in the reference sheets). Also, assume the following 3 persons are added to the priority queue in order:

Person 1: name="Sara" profit=65
Person 2: name="Mike" profit=60
Person 3: name="Jessica" profit=75

If we remove all people from the priority queue, the first person that will be removed is Mike, next is Sara, and the last is Jessica.

Your task is to design `Deque` and `Pqueue`. For the methods, only provide the docstring. You DO NOT need to provide the implementation of the methods. Note that an implementation of `remove()` for `Pqueue` is provided in Question 4 if you want to see it first.

```
from queue import Queue
from person import Person

class Deque(Queue):
    """
    A double ended queue, called deque
    """

    def add_to_front(self, obj):
        """
        Add the obj to the front of Deque self

        Extends class Queue. This is a new method.

        :param obj: object to add
```

```
        :type obj : object
        :rtype: None
        >>> d = Deque()
        >>> d.add(1)
        >>> d.add(2)
        >>> d.add_to_front(3)
        >>> print(d)
        [3, 1, 2]
        """


    def remove_from_end(self):
        """
        Remove and return the last object from Queue self

        Deque self must not be empty.

        Extends class Queue. This is a new method.

        :rtype: object
        >>> d = Deque()
        >>> d.add(1)
        >>> d.add(2)
        >>> d.remove_from_end()
        2
        """


class Pqueue(Queue):
    """
    A priority queue, called Pqueue
    """
    def __init__(self, a_priority):
        """
        Create and initialize new Pqueue self.

        Precondition: a_priority must be one of the attributes that all
        elements that will be added to Pqueue self have

        :param a_priority: one of the attributes of the elements that will
        be added to the Pqueue self
        :type a_priority: str
        """

    def remove(self):
        """
        Remove and return the element from Pqueue self, based on priority

        Pqueue self must not be empty.

        Overrides Queue.remove()

        :rtype: object
```

```
>>> pq = Pqueue("profit")
>>> pq.add(Person("Sara",65))
>>> pq.add(Person("Mike",60))
>>> pq.add(Person("Jessica",75))
>>> print(pq.remove())
Mike 60
>>> print(pq.remove())
Sara 65
>>> print(pq.remove())
Jessica 75
"""
```

Question 4. Unit Test **[10 Marks]**

This is an implementation of method `remove()` for class `Pqueue` discussed in Question 3.

```
def remove(self):
    """
    Remove and return the best priority element from Pqueue self.
    Pqueue self must not be empty.

    # We have omitted rest of the docstring. You may want to refer
    # to Question 3 for further information on Pqueue and remove().

    """

    index = 0
    # see the note below for further information on __getattribute__
    priority = self._content[0].__getattribute__(self._priority)
    i = 1
    while i < len(self._content):
        if self._content[i].__getattribute__(self._priority) < priority:
            priority = self._content[i].__getattribute__(self._priority)
            index = i
        i+=1

    return self._content.pop(index)
```

Note. `__gettattribute__(x)` returns the object attribute whose name is in `x`. For instance, if the value of `self._priority` is "profit", then profit of persons in the underlying queue is assigned to `priority`.

Write code for a complete `unittest` file for function `remove(…)` above. For full marks, you should have at least two test cases: one when the body of the "`while`" loop does not execute, and at least one for other cases. For this question, comments and docstring are NOT required.

```
import unittest
from q3 import Pqueue, Person     # assume Pqueue and Person are from q3.py

class TestPqueue(unittest.TestCase):
    def setUp(self):
        self.pq = Pqueue("profit")

    def tearDown(self):
        self.pq = None

    def test_one_item_removal(self):
        self.pq.add(Person("Sara", 65))
        removed= self.pq.remove()
        assert removed.name == "Sara"
```

```
        assert self.pq.is_empty()

    def test_more_items1(self):
        self.pq.add(Person("Sara", 65))
        self.pq.add(Person("Jesse", 70))
        self.pq.add(Person("Monica", 75))
        removed= self.pq.remove()
        assert removed.name == "Sara"
        assert not self.pq.is_empty()

    def test_more_items2(self):
        self.pq.add(Person("Sara", 65))
        self.pq.add(Person("Jesse", 55))
        self.pq.add(Person("Monica", 75))
        removed= self.pq.remove()
        assert removed.name == "Jesse"
        assert not self.pq.is_empty()


if __name__ == '__main__':
    unittest.main()
```

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

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

Short Python function/method descriptions:

```
__builtins__:
  len(x) -> integer
      Return the length of the list, tuple, dict, or string x.
  max(L) -> value
      Return the largest value in L.
  min(L) -> value
      Return the smallest value in L.
   range([start], stop, [step]) -> list of integers
      Return a list containing the integers starting with start and ending with stop - 1 with step specifying
      the amount to increment (or decrement). If start is not specified, the list starts at 0.
      If step is not specified, the values are incremented by 1.
    sum(L) -> number
      Returns the sum of the numbers in L.

dict:
   D[k] -> value
      Return the value associated with the key k in D.
   k in d -> boolean
      Return True if k is a key in D and False otherwise.
   D.get(k) -> value
      Return D[k] if k in D, otherwise return None.
   D.keys() -> list of keys
      Return the keys of D.
   D.values() -> list of values
      Return the values associated with the keys of D.
   D.items() -> list of (key, value) pairs
      Return the (key, value) pairs of D, as 2-tuples.

float:
   float(x) -> floating point number
      Convert a string or number to a floating point number, if possible.

int:
   int(x) -> integer
      Convert a string or number to an integer, if possible. A floating point argument will be truncated
      towards zero.

list:
   x in L -> boolean
      Return True if x is in L and False otherwise.
   L.append(x)
      Append x to the end of list L.
   L1.extend(L2)
      Append the items in list L2 to the end of list L1.
   L.index(value) -> integer
      Return the lowest index of value in L.
   L.insert(index, x)
      Insert x at position index.
```

L.pop()
Remove and return the last item from L.
L.remove(value)
Remove the first occurrence of value from L.
L.sort()
Sort the list in ascending order.

Module random: randint(a, b)
Return random integer in range [a, b], including both end points.

str:
x in s -> boolean
Return True if x is in s and False otherwise.
str(x) -> string
Convert an object into its string representation, if possible.
S.count(sub[, start[, end]]) -> int
Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.
S.find(sub[,i]) -> integer
Return the lowest index in S (starting at S[i], if i is given) where the string sub is found or -1 if sub does not occur in S.
S.split([sep]) -> list of strings
Return a list of the words in S, using string sep as the separator and any whitespace string if sep is not specified.

set:
{1, 2, 3, 1, 3} -> {1, 2, 3}
s.add(...)
Add an element to a set
set()
Create a new empty set object
x in s
True iff x is an element of s

list comprehension:
[<expression with x> for x in <list or other iterable>]

functional if:
<expression 1> if <boolean condition> else <expression 2>
-> <expression 1> if the boolean condition is True, otherwise <expression 2>

======Class Container ====================

```
class Container:
    """
    A data structure to store and retrieve objects.
    This is an abstract class that is not meant to be instantiated itself,
    but rather subclasses are to be instantiated.
    """
    def __init__(self):
        """
```

```
    Create a new and empty Container self.
    """
    self._content = None
    raise NotImplemented ("This is an abstract class, define or use its subclass")

def add(self, obj):
    """
    Add object obj to Container self.

    :param obj: object to place onto Container self
    :type obj: Any
    :rtype: None
    """
    raise NotImplemented ("This is an abstract class, define or use its subclass")

def remove(self):
    """
    Remove and return an element from Container self.
    Assume that Container self is not empty.
    :return an object from Container slef
    :rtype: object
    """
    raise NotImplemented ("This is an abstract class, define or use its subclass")

def is_empty(self):
    """
    Return whether Container self is empty.
    :rtype: bool
    """
    return len(self._content) == 0

def __eq__(self, other):
    """
    Return whether Container self is equivalent to the other.

    :param other: a Container
    :type other: Container
    :rtype: bool
    """
    return type(self)== type(other) and self._content == other._content

def __str__(self):
    """
    Return a human-friendly string representation of Container.
    :rtype: str
    """
    return str(self._content)
```

======Class Stack ======================

```
from container import Container

class Stack(Container):
    """Last-in, first-out (LIFO) stack.
    """
    def __init__(self):
        """Create a new, empty Stack self.

        Overrides Container.__init__

        """
        self._content = []

    def add(self, obj):
        """ Add object obj to top of Stack self.

        Overrides Container.add

        :param obj: object to place on Stack
        :type obj: Any
        :rtype: None
        >>> s = Stack()
        >>> s.add(1)
        >>> s.add(2)
        >>> print(s)
        [1, 2]
        """
        self._content.append(obj)

    def remove(self):
        """
        Remove and return top element of Stack self.

        Assume Stack self is not empty.

        Overrides Container.remove

        :rtype: object
        >>> s = Stack()
        >>> s.add(5)
        >>> s.add(7)
        >>> s.remove()
        7
        """
        return self._content.pop()
```

======Class Queue ======================

```
from container import Container

class Queue (Container):
    """A first-in, first-out (FIFO) queue.
    """

    def __init__(self):
        """
        Create and initialize new Queue self.

        Overrides Container.__init__

        """
        self._content = []

    def add(self, obj):
        """
        Add object at the back of Queue self.

        Overrides Container.add

        :param obj: object to add
        :type obj: object
        :rtype: None
        >>> q = Queue()
        >>> q.add(1)
        >>> q.add(2)
        >>> print(q)
        [1, 2]
        """
        self._content.append(obj)

    def remove(self):
        """
        Remove and return front object from Queue self.

        Queue self must not be empty.

        Overrides Container.remove

        :rtype: object

        >>> q = Queue()
        >>> q.add(3)
        >>> q.add(5)
        >>> q.remove()
        3
        """
        return self._content.pop(0)
```

======Class Person ====================

```
class Person:
    """
    A person

    ====== Public Attributes =======
    :type name: str
        name of the person
    :type profit: int
        the profit that the person makes
    """
    def __init__(self, name="", p=0):
        """
        Create a new Person self

        :param name: name of the person
        :type name: str
        :param p: profit that the person makes
        :type p: int
        """
        self.name = name
        self.profit = p

    def __str__(self):
        """
        Return a string representation of a Person
        :return:
        :rtype: str
        >>> p = Person("Jessica", 75)
        >>> print(p)
        Jessica 75
        """
        return "{} {}".format(self.name,str(self.profit))
```