First Name:
Last Name:
Student #

University of Toronto, Mississauga
CSC 148, Midterm Exam 1
13th February 5:10pm to 6:00pm
Good Luck!

<u>In addition to the correct answer, you MUST show all your work in order to receive full credit.</u>

Questions	Mark:
Question1)	/10
Question 2)	/10
Question 3)	/10
Question 4)	/10

CIRCLE YOUR INSTRUCTOR:

T.Tiffany (10am to 11am, MWF)

A.Attarwala (11am to 12pm, MWF)

THIS EXAM CONTAINS A TOTAL OF 7 PAGES. PAGE 7/7 IS THE APPENDIX OF YOUR EXAM

First Name:
Last Name:
Student #

Question 1) [10]

Consider the `Stack` and the `Queue` class with standard set of operations (also reproduced in the appendix of this exam). Using this `Stack` and `Queue` class, what items are contained in them (**clearly mark the top and bottom of your stack and queue**) just before the `mysteryFunction` is called AND just after the `mysteryFunction` is called?

```python
def mysteryFunction(s,q):
    q.enqueue('csc148')
    q.enqueue(True)
    q.enqueue(q.front())
    q.enqueue('abstract data type')

    for i in range(q.size()):
        s.push(q.dequeue())

    while not s.is_empty():
        q.enqueue(s.pop())


if __name__ == '__main__':
    s=Stack()
    q=Queue()

    #About to call mysteryFunction
    #What are contents of s and q at this point?
    mysteryFunction(s,q)
    #mysteryFunction has been called.
    #What are contents of s and q at this point?
```

Before mysteryFunction called	After mysteryFunction called

First Name:
Last Name:
Student #

Question 2) **[10]**

Consider the following program below:

```
def division(a,b):
    try:
        output_file = open('math.txt','w')
        answer = a / b
        output_file.write(str(answer))
    except ZeroDivisionError:
        print ('Cannot divide by zero')
    except TypeError:
        print ('Wrong type')
    except NameError:
        print ('Name not defined inside')
    except IOError:
        print ('Invalid file')
    else:
        output_file.close()
        print('Answer written to file')
    finally:
        print('End of program')
```

Complete the following table based on the given function call. Indicate the type(s) of error followed by its corresponding print statement.

Function Call	**Errors Raised**	**Print Statement**
`division(6,0)`		
`division(0,42)`		
`division(100,50)`		
`division('science',1)`		

`try:`
`   division(x,1)`
`except NameError:`
`   print ('Name not defined outside')`		

First Name:
Last Name:
Student #

Question 3) **[10]**

Given two lists of equal length, a dot product between two lists is defined as follows:

$a = [a_1, a_2, a_3, a_4, \ldots, a_n]$
$b = [b_1, b_2, b_3, b_4, \ldots, b_n]$

$\text{dotProduct}(a,b) = (a_1 \times b_1)+(a_2 \times b_2)+(a_3 \times b_3)+\ldots+(a_n \times b_n)$

Use recursion to compute the dot product between two lists. If the lists are of unequal length, you must `raise` the `UnequalLists` exception (see appendix).

YOU MUST CLEARLY MARK YOUR BASE CASE AND THE RECURSIVE CASE IN YOUR CODE.

```
def dotProduct(a, b):
    ‘’'(list , list) -> int
    dot product between list a and list b'''
```

First Name:
Last Name:
Student #

Question 4) **[10]**

Given two lists of equal length, the addition between two lists is defined as follows:

$a = [a_1, a_2, a_3, a_4, \ldots., a_n]$
$b = [b_1, b_2, b_3, b_4, \ldots., b_n]$

$\text{addTwoLists}(a,b) = [(a_1 + b_1),(a_2 + b_2),(a_3 + b_3),\ldots,(a_n + b_n)]$

Use recursion to compute the addition between two lists. If the lists are of unequal length, you must `raise` the `UnequalLists` exception (see appendix).

YOU MUST CLEARLY MARK YOUR BASE CASE AND THE RECURSIVE CASE IN YOUR CODE.

```
def addTwoLists(a, b):
    ‘’’(list, list) -> list
    addTwoLists between list a and list b and
    return a new list c that represents the
    addition of list a and list b'''
```

First Name:
Last Name:
Student #

EXTRA PAGE

First Name:
Last Name:
Student #

Appendix:

```
#You can assume all methods in the Stack and Queue
#class have been correctly implemented. The constructor creates an empty Stack and
#empty Queue.
class Queue:
    def enqueue(self,e):
        '''Add an element e to the back of the queue’''
          .
          .
          .

    def dequeue(self):
        '''Remove and return the first element of the queue'''
          .
          .
          .

    def front(self):
        '''Returns a reference to the element at the front of
        the queue’''
          .
          .
          .

    def size(self):
        '''Returns the number of elements in the queue’''
          .
          .
          .



class Stack:
    def push(self,e):
        '''Add element e to the top of the stack’''
          .
          .
          .

    def pop(self):
        '''Remove and return the element from the top of the stack’''
          .
          .
          .

    def is_empty(self):
        '''Returns True if the stack is empty otherwise returns False’''
          .
          .
          .

class UnequalLists(Exception):
    def __init__(self,s):
     '''Returns instance of UnequalLists exception where s is a string’’
     .
     .
     .
```