CSCA08 Fall 2014 Final Exam
Duration — 160 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_| **A**

Instructor: Brian Harrington

Last Name: ____________________

First Name: ____________________

UtorID (Markus Login): ____________________

*Do **not** turn this page until you have received the signal to start.*

This exam consists of 7 questions on 18 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.* Proper documentation is required for all functions and code blocks. If you use any space for rough work, indicate clearly what you want marked. Please read all questions thoroughly before starting on any work.

We have provided you with grids for your answers, this is simply to help you show the indentation of your code and you are not required to adhere to the grids in any specific way.

The University of Toronto's Code of Behaviour on Academic Matters applies to all University of Toronto Scarborough students. The Code prohibits all forms of academic dishonesty including, but not limited to, cheating, plagiarism, and the use of unauthorized aids. Students violating the Code may be subject to penalties up to and including suspension or expulsion from the University.

# 1:	______/ 5
# 2:	______/ 5
# 3:	______/ 5
# 4:	______/ 5
# 5:	______/10
# 6:	______/10
# 7:	______/10
TOTAL:	______/50

[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.]

Question 1. [5 marks]

Write the output of the following code in the space provided:

```
my_list = ['A', 'B', 'C', 'D', 'E']
my_str = "Hello World!"
my_dict = {}
count = 0
for i in range(len(my_list) - 1, 0, -1):
    my_dict[my_list[i]] = my_str[count]
    count += 2

for next_element in my_list:
    if(next_element in my_dict):
        print(next_element, " -> ", my_dict[next_element])
    else:
        print(next_element, " -> OOPS")
```

[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.]

Question 2. [5 marks]

Write the output of the following code in the space provided.

```
class CoolClass():
    def __init__(self, a, b):
        self._a = a
        self._b = b

    def __str__(self):
        return "Cool:" + self._a +"-"+ self._b

    def set_ab(self, a, b):
        self._a = a
        self._b = b

    def get_vals(self):
        return [self._a, self._b]

class BoringClass(CoolClass):
    def __init__(self, a, b, c):
        CoolClass.__init__(self, a, b)
        self._c = c

    def __str__(self):
        ret = CoolClass.__str__(self)
        ret += "-" + self._c
        return ret

    def get_vals(self):
        return [self._a, self._b, self._c]


class CrazyClass(BoringClass):
    def __init__(self, a, b, c):
        CoolClass.__init__(self, c, a)
        self._d = b
        self._c = BoringClass(c, b, a).get_vals()

    def __str__(self):
        return "CRAZY:" + str(self.get_vals())

if(__name__ == "__main__"):
    v1 = CoolClass('A', 'B')
    print(v1)
    v2 = BoringClass('A', 'B', 'C')
    print(v2)
    v2.set_ab('D', 'E')
    print(v2)
    v3 = CrazyClass('A', 'B', 'C')
    print(v3)
    v3.set_ab('A', 'B')
    print(CoolClass.get_vals(v3))
```

```
def scramble(simple_pwd):
    """ (str) -> str

    Returns a scrambled version of simple_pwd.
    """

    ALPHABET = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    #reversed: 'ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210'
    last_alpha = len(ALPHABET) - 1


    first_char_index = ALPHABET.find(simple_pwd[0])
    obscure_pwd = ''
    i = 1
    while i < len(simple_pwd):
        if simple_pwd[i] != simple_pwd[i-1]:
            curr_char_index = ALPHABET.find(simple_pwd[i])
            if curr_char_index > first_char_index:
                obscure_pwd = ALPHABET[curr_char_index] + obscure_pwd
            else:
                obscure_pwd = ALPHABET[last_alpha - curr_char_index] + obscure_pwd
        i += 1

    return obscure_pwd + simple_pwd[0]
```

Question 3. [5 marks]

With so many systems requiring a password, and experts telling us that we should not re-use passwords, Nick is having trouble remembering all his passwords. Using simple, easy to remember passwords would be a solution, except those same experts advise against using obvious passwords. So Nick came up with something clever.

He wrote a function to scramble his simple passwords into obscure looking strings, which he will use as passwords. Now when Nick forgets a password, he just remembers his simple version, and puts it through his function to get his password.

Nick's function starts with an alphabet of symbols that are used, both for the simple and scrambled passwords. For your convenience, he also included his alphabet in reverse. His function is on the previous page.

Write the output of the following code in the space provided:

```
print(scramble('XXXYYYZZZ'))
print(scramble('ABC123'))
print(scramble('PASSWORD'))
```

[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.]

Question 4. [5 marks]

Alfred taught a fourth year course on Languages and Automata Theory, for which he had 5 assignments, each with weight 20%, and no tests or exams. He noticed that some students tended to improve over the term, and some others tended to worsen. Alfred was most interested in these "trendy" students. For each student he had 5 marks for the 5 assignments,
$M1, M2, M3, M4, M5$
Each mark is an integer between 0 and 20. Alfred defined two kinds of trendy students. An upward trendy student is one whose marks satisfy the conditions
$M1 < M2 < M3 < M4 < M5$
A downward trendy student is one whose marks satisfy the conditions
$M1 > M2 > M3 > M4 > M5$
Nick wrote a function for Alfred. Its arguments are 5 integers,
$M1, M2, M3, M4, M5$
which are a student's marks. Each mark must be in the interval [0,20].
The function returns
+1 if the student is upward trendy
−1 if the student is downward trendy
0 if the student is not trendy
However, Alfred does not really know programming and he distrusts Nick's coding ability. So he wants to test Nick's function thoroughly. Please help Alfred by providing a good set of test cases. (We've filled in an example line to show you the format. Don't count this line as a test case)

M1	M2	M3	M4	M5	Return	Explanation
1	7	3	4	8	0	Nicks' marks (just an example)

[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.]

Question 5. [10 MARKS]

At UTSC, there are many clubs. Each club may have any number of members, who in turn may be members of other clubs. A good way to represent these clubs and their members is to use a dictionary whose keys are the names of the clubs, and whose values are lists of club members.

Nick wrote a function that, given a dictionary that represents clubs and their members, returns another dictionary whose keys are member names, and whose values are lists of clubs to which the members belong. But guess what? The CODE STEALER (a distant cousin of the CODE MANGLER) got to it, deleting every line of code, but leaving the comments. Please help Nick to complete his code again.

```
def get_membership(clubs_to_members):
    """ (dict of str to list of str) -> dict of str to list of str

    Return a dictionary whose keys are the members of all clubs in
    clubs_to_members, and whose values are the lists of clubs to which
    that member belongs.

    >>> clubs_to_members = {'programming': ['Brian', 'Nick', 'Paco'],
                           'games': ['Paco', 'Brian'], 'food': ['Paco'], 'homework': []}
    >>> members_to_clubs = get_membership(clubs_to_members)
    >>> members_to_clubs == {'Paco': ['food', 'games', 'programming'],
         'Brian': ['games', 'programming' ], 'Nick': ['programming']}
    True
    """
    # create an empty dictionary to use as result



    # loop through every club



        # loop through each member in the club



            # 2 cases to deal with:
                # if the member is already there; add to end of existing list
                # if the member is not in result; create new entry







    # return the result
```

Question 6. [10 MARKS]

In a standard deck of 52 cards, each card has a rank and a suit.

The ranks are: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.
The suits are: Clubs, Diamonds, Hearts, Spades.

For this question, we represent each rank by a string of length one:

'2' to '9' for 2 to 9 respectively, 'T' for 10,
'J' for Jack, 'Q' for Queen, 'K' for King, 'A' for Ace.

We also represent each suit by a string of length one:

'C' for Clubs, 'D' for Diamonds, 'H' for Hearts, 'S' for Spades.

Finally, we represent a card by a string of length 2, the first character being the rank and the second being the suit. For example, Queen of Hearts is represented by 'QH', and 3 of Clubs is represented by '3C'.

In the game of bridge, a hand consists of 13 cards, which we represent by a list of cards. When given a hand, here is the common way to count points.

High rank points:

- For each Ace, we count 4 points.
- For each King, we count 3 points.
- For each Queen, we count 2 points.
- For each Jack, we count 1 point.

Short suit points:

- For each suit, we count 1 point if the hand contains exactly two cards of that suit.
- For each suit, we count 2 points if the hand contains exactly one card of that suit.
- For each suit, we count 3 points if the hand contains no cards of that suit.

On the next page, write a function which takes a hand as input and returns the number of points in that hand.

Complete the following function given the description on the previous page

```
def num_points(hand):
```

Question 7. [10 MARKS]

Brian has list of A08 marks. Each item in the list is a 2-tuple, `(name, mark)` where name belongs to a student (a string) and mark is that student's A08 mark (an integer).

He wrote a function that, given his list and a grade `('A', 'B', 'C', 'D' or 'F')`, returns a list of students whose marks earn them that grade[1]. Brian had several versions of the function written in the file. Then (yeah, you guessed it) the CODE MANGLER got to it, and deleted all comments and indentation. He (she?) even got rid of duplicate lines (i.e., you can use lines more than once). Please help Brian to restore his code.

Mangled Code

```
elif(grade == 'A'):
elif(grade == 'B'):
elif(grade == 'C'):
elif(grade == 'D'):
elif(grade == 'F'):
else:
for (name, mark) in markslist:
for i in range(len(markslist)):
grade = maxmark_to_grade[i]
grade = minmark_to_grade[i]
grade_to_minmax = {'A':(80,100), 'B':(70,79), 'C':(60,69), 'D':(50,59), 'F':(0,49)}
if(grade == 'A'):
if(grade == 'B'):
if(grade == 'C'):
if(grade == 'D'):
if(grade == 'F'):
if((min_mark <= mark) and (mark <= max_mark)):
if((0 <= mark) and (mark <= 49))
if((50 <= mark) and (mark <= 59))
if((60 <= mark) and (mark <= 69))
if((70 <= mark) and (mark <= 79))
if((80 <= mark) and (mark <= 100))
if(0 <= mark):
if(50 <= mark):
if(60 <= mark):
if(70 <= mark):
if(80 <= mark):
if(mark <= 49):
if(mark <= 59):
if(mark <= 69):
if(mark <= 79):
if(mark <= 100):
max_mark = grade_to_minmax[grade][1]
maxmark_to_grade = {49:'F', 59:'D', 69:'C', 79:'B', 100:'A'}
min_mark = grade_to_minmax[grade][0]
minmark_to_grade = {0:'F', 50:'D', 60:'C', 70:'B', 80:'A'}
result = []
result.append(name)
return result
```

[1]For reference: 'A' = 80-100, 'B' = 70-79, 'C' = 60-69, 'D' = 50-59, 'F' = 0-49

Complete the following function given the description on the previous page

```
def get_graded_students(markslist, grade):
```

[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.]

Short Python function/method descriptions:
You may tear this page off, but if you do so, you must not include any work on it (front or back) that you wish to have marked.

```
__builtins__:
  abs(number) -> number
    Return the absolute value of the given number.
  max(a, b, c, ...) -> value
    With two or more arguments, return the largest argument.
  min(a, b, c, ...) -> value
    With two or more arguments, return the smallest argument.
  isinstance(object, class-or-type-or-tuple) -> bool
    Return whether an object is an instance of a class or of a subclass thereof.
    With a type as second argument, return whether that is the object's type.
  int(x) -> int
    Convert a string or number to an integer, if possible.  A floating point argument
    will be truncated towards zero.
  str(x) -> str
    Convert an object into a string representation.

str:
  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]) -> int
    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.isalpha() --> bool
    Return True if and only if all characters in S are alphabetic
    and there is at least one character in S.
  S.isdigit() --> bool
    Return True if and only if all characters in S are digits
    and there is at least one character in S.
  S.islower() --> bool
    Return True if and only if all cased characters in S are lowercase
    and there is at least one cased character in S.
  S.isupper() --> bool
    Return True if and only if all cased characters in S are uppercase
    and there is at least one cased character in S.
  S.lower() --> str
    Return a copy of S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.split([sep]) -> list of str
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.startswith(prefix) -> bool
    Return True if S starts with the specified prefix and False otherwise.
  S.strip() --> str
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() --> str
    Return a copy of S converted to uppercase.
```

```
list:
  append(...)
    L.append(object) -- append object to end
  count(...)
    L.count(value) -> integer -- return number of occurrences of value
  index(...)
    L.index(value, [start, [stop]]) -> integer -- return first index of value.
    Raises ValueError if the value is not present.
  insert(...)
    L.insert(index, object) -- insert object before index
  pop(...)
    L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.
  remove(...)
    L.remove(value) -- remove first occurrence of value.
    Raises ValueError if the value is not present.

set:
  pop(...)
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.

dict:
  keys(...)
    D.keys() -> a set-like object containing all of D's keys
  get(...)
     D.get(k[,d]) -> returns D[k] if k is in D, otherwise returns d.  d defaults to None.

object:
  __init__(...)
    x.__init__(...) initializes x; called automatically when a new object is created
  __str__(...)
    x.__str__() <==> str(x)
```