COSC 101, Final Exam
Fall 2018

Honor Code

I agree to comply with the spirit and the rule of the Colgate University Academic Honor Code during this exam and throughout the final exam period. I will not discuss the contents of this exam with other students until the exam period is over, and affirm that I have neither given or received inappropriate aid on this exam.

Signature: _____

Printed Name: _____

Write your name; do not open the exam until instructed to do so.

You have 120 minutes to complete this exam.

There are 8 questions and a total of 87 points available for this exam. Don't spend too much time on any one question.

Since indentation is important in Python, please be sure that your use of indentation is obvious for any code you write.

If you want partial credit, show as much of your work and thought process as possible.

If you run out of space for answering a question, you can continue your answer on one of the scrap pages at the end of the exam. If you do so, be sure to indicate this in two places: (1) below the question, indicate which scrap page contains your answer, and (2) on the scrap page, indicate which question you are answering.

Question:	1	2	3	4	5	6	7	8	Total
Points:	9	11	7	10	10	10	10	20	87
Score:									

1. (9 points) Assume that the following statements have already been executed:

```
a = ['hot', 'chocolate']
b = 'snowman snowball snowfall'
c = 'slippery'
d = {'cold': 'snow', 'freezing': 32, 'below zero': -1,
       'warm': 'fire'}
```

For each of the following expressions, evaluate the expression and write the resulting value, or identify the error in the code that would prevent it from running.

(a) `a.count('h')`

Solution:

```
0
```

(b) `'n' in d[0]`

Solution:

```
Error: 0
```

(c) `32 in d.keys() and not('cold' in d.values())`

Solution:

```
False
```

(d) `a[d['below zero']]`

Solution:

```
'chocolate'
```

(e) `d['freezing'] + b.find('man')`

Solution:

```
36
```

(f) `b.split()[1].upper()`

Solution:

```
'SNOWBALL'
```

(g) `a[0].join(a)`

Solution:

```
'hothotchocolate'
```

(h) `a[len(d) - len(a)]`

Solution:

```
Error: list index out of range
```

(i)
```
x = list(d.keys())
x.sort()
print(x[-1] + c[:-1] + c[0])
```

Solution:

```
warmslippers
```

2. For each program, state what it will print. None of the programs contains an error.

   (a) (4 points) What is printed by the following program?

```
def f1(y):
    x = []
    x.append(y[0]*2)
    x.append(y[1]*3)
    print("X:", x)
    print("Y:", y)
    return x

x = ['a', 'b']
y = f1(x)
print("x:", x)
print("y:", y)
```

Solution:

```
X: ['aa', 'bbb']
Y: ['a', 'b']
x: ['a', 'b']
y: ['aa', 'bbb']
```

   (b) (3 points) What is printed by the following program?

```
def f2(u, g, h):
    u[0] = g
    print("h:", h)
    h.remove(4)
    return h

u = [1, 3, 4]
g = 'c'
print(f2(u, g, u))
print("u:", u)
```

Solution:

```
h: ['c', 3, 4]
['c', 3]
u: ['c', 3]
```

(c) (4 points) What is printed by the following program?

```
def f3(x):
    print("x:", x)
    if sum(x) > 5:
        return sum(x)
    else:
        return f3(x+x)

rv = f3([2])
print("rv:", rv)
```

Solution:

```
x: [2]
x: [2, 2]
x: [2, 2, 2, 2]
rv: 8
```

3. (7 points) For this problem, select one line of code from each of the pairs of lines of code below and reorder them to solve the following problem:

> Write a function named `make_teams` that accepts a list of student names and a group size, and returns a list of lists, where each sublist contains student names, grouped by the given group size. If the number of students given is not evenly divisible by the group size, the last sublist will be less than the group size.

For example `make_teams(['Lin', 'John', 'Soria', 'Javier', 'Maggie'], 2)` should return the list `[['Lin', 'John'], ['Soria', 'Javier'], ['Maggie']]`.

A1 `        nextgroup = students[groupsize:]`
A2 `        nextgroup = students[:groupsize]`

B1 `    return groups`
B2 `    return students`

C1 `        students = students[:groupsize]`
C2 `        students = students[groupsize:]`

D1 `    while len(students) > 0:`
D2 `    for i in range(0, len(students)):`

E1 `def make_teams(students, groupsize):`
E2 `def make_teams([students]):`

F1 `        groups += [nextgroup]`
F2 `        groups.append(nextgroup)`

G1 `    groups = []`
G2 `    groups = list`

Select only 7 lines of code from above, and only one line from each pair. You may fill in line identifiers (*e.g.*, E2) below, or write out the code.

Solution:
E1
G1
D1

A2
C2
F2
B1

4. (10 points) For this problem, select one line of code from each of the pairs of lines of code shown on the next page and reorder them to solve the following problem:

   President Casey, delighted with the hard work that the University's professors put into the semester has decided to buy gifts for the best professors in each department. To keep track of all of the departments and gifts he plans to give, he is using the most obvious natural choice: a Python dictionary. It looks like this:

```
faculty_gifts = { 'English':  {'DuComb': 'iPad',
                               'Maurer': 'The Golden Mean Book'},
                  'Chemistry': {'Nolen': 'iPad'},
                  'Computer Science': {'Mulry': 'iPad',
                        'Hay': 'Breaking Bad DVD',
                        'Fourquet': 'The Golden Mean Book'}
                 }
```

   (Each inner dictionary maps a person name to the gift they will receive from Casey. Shh…it's a surprise!) As Casey heads out to the mall, he realizes that this dictionary is not the most efficient representation. What he needs is a Python dictionary that looks like this:

```
shopping_list = { 'iPad': 3,
                  'Breaking Bad DVD': 1,
                  'The Golden Mean Book': 2
                }
```

   This dictionary maps each gift to the number of times it will be given. Write a function that takes in a dictionary of the form of `faculty_gifts` and returns a dictionary of the form of `shopping_list`. You should not assume that the gifts shown in the example above are the only possible gifts.

(problem continues on the next page)

A1
```
            shopping[gift] += 1
```
A2
```
            shopping[gift].append(1)
```

B1
```
            shopping[gift] = 1
```
B2
```
            shopping[gift] = [1]
```

C1
```
    return shopping.items()
```
C2
```
    return shopping
```

D1
```
        else:
```
D2
```
        elif gift in shopping:
```

E1
```
        if shopping.contains(gift):
```
E2
```
        if gift in shopping:
```

F1
```
      for person in holidays.keys(holiday):
```
F2
```
      for person in holidays[holiday]:
```

G1
```
def shopping_list(holidays):
```
G2
```
def shopping_list({holidays}):
```

H1
```
        gift = holidays[person][holiday]
```
H2
```
        gift = holidays[holiday][person]
```

I1
```
    shopping = []
```
I2
```
    shopping = {}
```

J1
```
    for holiday in holidays:
```
J2
```
    for holiday in holidays.values():
```

Select only 10 lines of code from above, and only one line from each pair. You may fill in line identifiers (*e.g.*, E2) below, or write out the code.

Solution:
G1
I2

J1
F2
H2
E2
A1
D1
B1
C2

5. (10 points) Your friend is designing a campus scavenger hunt for her sorority, where teams will have to solve clues and go to a place on campus to find the next clue. She's organized her clues and where they're placed into two separate Python dictionaries, as shown in the example below.

```
location_to_clue_given = { "CS Lounge" : 1, "Cooley" : 3,
    "Donovan's" : 5, "University Chapel" : 4, "START" : 2}

clue_to_next_location = { 1 : "Cooley", 2 : "CS Lounge",
    4 : "FINISH", 3 : "Donovan's", 5 : "University Chapel"}
```

Write a function `print_route` that takes two dictionaries in the format above. The first parameter is a dictionary like `location_to_clue_given` that indicates, for each location, what clue a team would find there. In this dictionary, there will be one key `"START"` that indicates the clue the teams are given to start. The second parameter is a dictionary like `clue_to_next_location`, and indicates where each clue leads when solved. In this dictionary, there will be one clue that maps to `"FINISH"`, indicating that the scavenger hunt is over when this clue is solved.

Your function should print out the entire route traversed by the scavenger hunt team in the format below.

```
>>> print_route(location_to_clue_given, clue_to_next_location)
START -> CS Lounge
CS Lounge -> Cooley
Cooley -> Donovan's
Donovan's -> University Chapel
University Chapel -> FINISH
```

Solution:

```
def print_route(clues_given, answers):
    from_location = 'START'
    to_location = answers[clues_given[from_location]]

    while to_location != 'FINISH':
        print(from_location, "->", to_location)
        from_location = to_location
        to_location = answers[clues_given[from_location]]

    print(from_location, "->", to_location)
```

6. (10 points) Write a **recursive** function `cold_count(L, count)` that takes a list of temperatures and an integer `count` and returns `True` if the list contains at least `count` temperatures that are below freezing (less than 32) and `False` otherwise.

   You may assume that `count` is ≥ 0 initially.

   Examples:

   - `cold_count([34, 15, 14, 35], 3)` returns `False`.
   - `cold_count([34, 15, 14, 35], 2)` returns `True`.
   - `cold_count([80, 71], 0)` returns `True`.
   - `cold_count([80, 75], 1)` returns `False`.

   Requirements: your solution must **not** use any built-in functions or methods. Your solution **must** be recursive. Solutions that do not meet these requirements earn **zero** points.

   Partial credit option (7 points): If you write a non-recursive `cold_count` that calls a recursive helper function, you can receive partial credit.

Solution:

```
def cold_count(L, count):
    if count == 0:
        return True
    if len(L) == 0:
        return False
    if L[0] < 32:
        count -= 1
    return cold_count(L[1:], count)
```

Partial credit: 7/8 out of 10:

```
def cold_count2(L, count):
  return sum_cold(L) >= count

def sum_cold(L):
  if len(L) == 0:
    return 0
  if L[0] < 32:
    return 1 + sum_cold(L[1:])
  else:
    return sum_cold(L[1:])
```

7. (10 points) Suppose you are given a file called `scores.txt`, which contains information about players' scores on a variety of games. Your task is to write a function, `maxscores` that creates a new file called `best_scores.txt`, containing the best player for each game. The format for each file is as follows: on the left is an example of `scores.txt` for which the file on the right, `best_scores.txt`, is produced.

`scores.txt` content:

```
# Angry Birds
Andre 150
Keisha 175
# Happy Birds
Andre 325
Keisha 120
Vijay 275
# Flappy Birds
Elodie 120
Nguyen 5000
```

`best_scores.txt` content:

```
# Angry Birds
175
# Happy Birds
325
# Flappy Birds
5000
```

In the input file, `scores.txt`, if a line starts with `'# '`, then the line records the name of a game. The lines that follow consist of player-score pairs, one per line, representing a player performance on that game. This pattern repeats with each occurrence of a game name. Note that the scores are not ordered and the number of players is not constant.

Write a program to solve this problem. The `maxscores` function should read a file called `scores.txt` and summarize the data to produce the file `best_scores.txt` with only the highest score for each game. The games in `best_scores.txt` do not have to be presented in the same order than in `scores.txt`. You can assume that no two players will tie for highest score.

Hint: be sure your program handles the last high score in `scores.txt`!

Solution:

```
def write(fileObj, game, score):
       fileObj.write(game+"\n")
       fileObj.write(str(score)+"\n")

def maxscores(filename):
       f = open(filename, 'r')
       f_out = open('bestscores.txt', 'w')

       game_name = ""
       for line in f:
              line = line.strip()
              if (line.startswith("#")):
                     old_game = game_name
                     if old_game != "":
                            write(f_out, old_game, maxscore)
```

```
                    game_name = line
                    maxscore = 0
            else:
                    fields = line.split()
                    score = int(fields[1])
                    if (score > maxscore):
                            maxscore = score
      write(f_out, old_game, maxscore)
      f.close()
      f_out.close()
maxscores('scores.txt')
```

(work space for previous problem)

8. (20 points) *This problem is challenging and worth twice as much as other questions.*

   Write a function named `read_phone_number` that accepts a string in the form of a United States-format telephone number (e.g., `'315-778-8000'`), and returns a new string with the telephone translated to English words. Note that with US-format telephone numbers, they are divided into three sections separated by hyphens.

   For repeated digits within a section, the digit should only be "said" once, with a word preceding it that indicates how many times the digit is repeated. For example, the number `'315-778-8000'`, should be read: "three one five double seven eight eight triple zero." (It does not read "double eight" because the 8s appear in different sections.)

   The rules for reading digits are as follows:

   - Single numbers should just be read separately
   - 2 successive numbers (within a section) use double.
   - 3 successive numbers (within a section) use triple.
   - 4 successive numbers (within a section) use quadruple.

   Additional examples are as follows:

   - `'212-333-4455'` should be read as "two one two triple three double four double five"
   - `'212-344-1555'` should be read as "two one two three double four one triple five"
   - `'111-111-1111'` should be read as "triple one triple one quadruple one"

   Program Design: This question is graded both on correctness and program design. You **must** write at least one helper function.

   In your solution, you can use `DIGITS` to refer to this list:

```
DIGITS = ['zero','one','two','three','four',
          'five','six','seven','eight','nine']
```

Solution:

```
DUPLICATE = ['','','double ',' triple ','quadruple ' ]

def translate(digit, count):
    rv = DUPLICATE[count] + DIGITS[int(digit)]
    return rv

def read_phone_section(phonestr):
    digitlist = list(phonestr)
    translation = []
    count = 1
    digit = digitlist[0]
    digitlist.pop(0)
    while len(digitlist) > 0:
```

```
        if digitlist[0] == digit:
            count += 1
        else:
            translation.append(translate(digit, count))
            digit = digitlist[0]
            count = 1
        digitlist.pop(0)
    translation.append(translate(digit, count))
    return translation


def read_phone_number(phonestr):
    words = []
    for section in phonestr.split('-'):
        words += read_phone_section(section)
    return ' '.join(words)

print(read_phone_number('315-778-8000'))
print(read_phone_number('212-344-4555'))
print(read_phone_number('111-111-1111'))
```

(This page is intentionally blank. Label any work with the corresponding problem number.)

(This page is intentionally blank. Label any work with the corresponding problem number.)

(This page is intentionally blank. Label any work with the corresponding problem number.)