COSC 101, Final
December 2023

Name: _____

Please write your name above. Do not start the exam until instructed to do so.

You have 2 hours to complete this exam.

There are 5 questions and a total of 85 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 blank 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 blank page contains your answer, and (2) on the blank page, indicate which question you are answering.

The last page of the exam contains documentation for string, list, and dictionary methods.

Question	Points	Score
1	16	
2	15	
3	20	
4	15	
5	19	
Total:	85	

1. (16 points) Assume that the following statements have already been executed:

```
name = 'Colgate'
rank = 1
mixed = [False, rank, 2.99, '3.14']
all = {'cheer': 'GoGate',
       'info': {'name': name, 'rank': rank},
       'misc': mixed}
```

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) `len(name)// 2 * name[len(name)% 3]`

Solution:

`'ooo'`

(b) `-rank ** int(mixed[2])`

Solution:

`-1`

(c) `rank + mixed[3]`

Solution:

`Type error`

(d) `mixed[1] != rank and round(mixed[2])<= int(mixed[2])or not mixed[0]`

Solution:

`True`

(e) `str(int(mixed[-2]))+ mixed[3] + str(int(mixed[0]))`

Solution:

`'23.140'`

(f) `mixed[-1:] + mixed[4:]`

Solution:

`['3.14']`

(g) `mixed[-2] + mixed[4]`

Solution:

`Index out-of-bounds error for list`

(h) `name in all`

Solution:

`False`

(i) `all[name]`

Solution:

`Key Error`

(j) `'gate' in all['info']['name']`

Solution:

`True`

(k) `'go' in all['cheer'].lower()and all['info']['name']`

Solution:

`True`

(l) `all['misc']['1']`

Solution:

`Type / Index error`

(m) `all['cheer'][:2] + all['info']['name'][3:].capitalize()`

Solution:

`'GoGate'`

(n)
```
name[:3] = 'Go'
len(name)
```

Solution:

```
Type error
```

(o)
```
mixed[:3] = ['Bazzinga']
len(all['misc'])
```

Solution:

```
2
```

(p)
```
all['cheer'] == 'GoGate' and mixed = all['misc']
```

Solution:

```
Syntax error
```

2. Consider the following function:

```
def mystery(month: str, day: int) -> None:
    if month not in ['June','July','Aug']:
        print("in season")
        if month in ['Sept','Oct','Nov','Dec']:
            print("$1000/month")
        elif month in ['Jan','Feb','March','April','May']:
            print("$900/month")
    else:
        if month in ['June','July']:
            print("off season")
            print("$100/week")
        elif month == 'Aug':
            if day < 15:
                print("off season")
                print("$100/week")
            else:
                print("in season")
                print("$250/week")
```

(a) (2 points) What does `mystery('June', 8)` print?

Solution:

```
off season
$100/week
```

(b) (2 points) What does `mystery('Aug', 8)` print?

Solution:

```
off season
$100/week
```

(c) (2 points) What does `mystery('Aug', 18)` print?

Solution:

```
in season
$250/week
```

(d) (2 points) What does `mystery('Oct', 8)` print?

Solution:

```
in season
$1000/month
```

(e) (2 points) What does `mystery('March', 8)` print?

Solution:

```
in season
$900/month
```

(f) (5 points) Re-write the function as one if-elif-else structure (that is, without any separate or nested if statements):

```
if ...
    ...
elif ...
    ...
elif ...
    ...
...
else
    ...
```

Solution:

```
def mystery(month: str, day: int) -> None:
    if month in ['Sept','Oct','Nov','Dec']:
        print("in season")
        print("$1000/month")
    elif month in ['Jan','Feb','March','April','May']:
        print("in season")
        print("$900/month")
    elif month in ['June','July']:
        print("off season")
        print("$100/week")
    elif month == 'Aug' and day < 15:
        print("off season")
        print("$100/week")
    elif month == 'Aug' and day >= 15:
        print("in season")
        print("$250/week")
```

3. Trace the following code snippets:

   (a) (6 points) Draw the loop tables as you trace the code and write the output. If the code results in an infinite loop, write the first two outputs followed by "infinite loop".

```
def mystery(i: int, j: int) -> None:
    while i > 5:
      if i % 2 == 0:
        i -= 2
      else:
        i += 1
      j += 1
      print(i, j)

mystery(7,1)
```

Solution:

```
8 2
6 3
4 4
```

(b) (7 points) Draw the loop tables as you trace the code and write the output. If the code results in an infinite loop, write the first two outputs followed by "infinite loop".

```
for i in range(1, 6, 2):
  print(i)
  for j in 'go':
    if i < 5:
      print(j)
```

Solution:

```
1
g
o
3
g
o
5
```

(c) (2 points) Draw the state as you trace the code and write the output:

```
def funcOne(a):
  a['nickName'] = 'Bill'
  print(a['nickName'])

x = {'nickName': 'Brian', 'lastName': 'Casey'}
funcOne(x)
print(x['nickName'])
```

Solution:

```
'Bill'
'Bill'
```

(d) (2 points) Draw the state as you trace the code and write the output:

```
def funcTwo(a):
  b = a[:]
  b[0] = 1
  print(b)

x = [2, 2]
funcTwo(x)
print(x)
```

Solution:

```
[1, 2]
[2, 2]
```

(e) (3 points) Draw the state as you trace the code and write the output:

```
def funcThree(a):
  a = a.upper()
  print(a)

x = 'colgate'
x.capitalize()
print(x)
funcThree(x)
print(x)
```

Solution:

```
'colgate'
'COLGATE'
'colgate'
```

4. Consider the following function:

```
def mysteryA(lst: list) -> list:
    ret_lst = []
    for i in range(0, len(lst),2):
        ret_lst += [lst[i]]
        ret_lst += ['$']*lst[i+1]
    return ret_lst
```

(a) (2 points) What does `mysteryA([1,1,1,1])` return?

Solution:

```
[1, '$', 1, '$']
```

(b) (2 points) What does `mysteryA([0,2,1,0])` return?

Solution:

```
[0, '$', '$', 1]
```

(c) (3 points) What input to `mysteryA` returns `[3,'$','$','$',1,'$','$']`?

Solution:

```
mysteryA([3,3,1,2])
```

Consider the following function:

```
def mysteryB(lst: list) -> list:
    ret_lst = [lst[0]]
    count = 0
    for i in range(1, len(lst)):
        if lst[i] != '$':
            ret_lst += [count]
            ret_lst += [lst[i]]
            count = 0
        else:
            count += 1
    ret_lst += [count]
    return ret_lst
```

(d) (2 points) What does `mysteryB([1,'$','$',3])` return?

Solution:

```
[1, 2, 3, 0]
```

(e) (3 points) What does `mysteryB([0,'$','$',1,2,'$',3])` return?

Solution:

```
[0, 2, 1, 0, 2, 1, 3, 0]
```

(f) (3 points) What input to `mysteryB` returns `[1,0,2,3]`?

Solution:

```
mysteryB([1,2,'$','$','$'])
```

5. **This problem has four parts**. The end goal is to write a well-structured program that reads a sentiment model from a file (like Homework 8) and produces a file with randomly generated positive reviews.

   For example, consider the following model:

   ```
   this 3.0
   product 3.0
   is 3.5
   good 5.0
   well 4.5
   made 5.0
   functions 4.0
   mediocre 3.0
   broke 2.0
   junk 1.0
   ```

   The file produced by the program may contain:

   ```
   is well made
   functions good
   well made is functions good
   ```

   You must write three functions for this program: `read_positive_model`, `save_positive_reviews` and `generate_positive_reviews`. The rest of the program looks as follows:

   ```
   def main() -> None:
       generate_positive_reviews("model.txt",3)

   main()
   ```

(a) (5 points) Write a function `read_positive_model` that accepts the name of the file containing the sentiment model and returns a dictionary with the words that have a value greater or equal to 3.5. For example, for the model provided, the expected dictionary is:

```
{'functions': 4.0,
    'good': 5.0,
    'is': 3.5,
    'made': 5.0,
    'well': 4.5}
```

Solution:

```
def read_positive_model(filename: str) -> dict:
    positive_model = {}
    with open(filename) as fobj:
        for line in fobj:
            parts = line.strip().split()
            if float(parts[1]) >= 3.5:
                positive_model[parts[0]] = float(parts[1])
    return positive_model
```

(b) (4 points) Write a function `save_positive_reviews` that accepts a list of reviews and save them to a file called `positive_reviews.txt`. For example,

```
save_positive_reviews(['is well made',
 'functions good',
 'well made is functions good'])
```

will produce the expected file content shown previously.

Solution:

```
def save_positive_reviews(reviews: list) -> None:
    fobj = open('positive_reviews.txt', 'w')
    for review in reviews:
        fobj.write(review + '\n')
    fobj.close()
```

(c) (8 points) Write a function `generate_positive_reviews` that accepts the name of the file containing the sentiment model and a count and produces a file with count number of randomly generated reviews.

Each review must:

- **contain at least two words**
- **contain at most `len(words_dict)` words**
- **not contain repeated words from the words dictionary**

Note: `generate_positive_reviews` must utilize `read_positive_model` and `save_positive_reviews`. If you wish, you may write additional helper functions.

Solution:

```
import random
def generate_positive_reviews(filename:str,count:int)->None:
    model = read_positive_model(filename)
    reviews = []
    for turn in range(count):
        review_words_count = random.randint(2, len(model))
        mkeys = list(model.keys())
        review = ''
        for step in range(review_words_count):
            to_remove = random.choice(mkeys)
            review += to_remove + ' '
            mkeys.remove(to_remove)

        reviews.append(review[:-1])

    save_positive_reviews(reviews)
```

(d) (2 points) If you did not write an additional helper function in the previous part, explain what parts of the code could be moved to a helper function and how you would re-write `generate_positive_reviews`.

If you did write an additional helper function in the previous part, explain your reasoning for doing so.

Solution: Generating a single review can be a separate helper function:

```
import random
def generate_positive_review(model: dict) -> str:
    review_words_count = random.randint(2, len(model))
    mkeys = list(model.keys())
    review = ''
    for step in range(review_words_count):
        to_remove = random.choice(mkeys)
        review += to_remove + ' '
        mkeys.remove(to_remove)

    return review[:-1]
```

`random` module methods

- `random()` — Returns a random decimal N such that `0.0 <= N < 1.0`
- `randint(a,b)` — Returns a random integer N such that `a <= N <= b`
- `randrange(a,b)` — Returns a random integer N such that `a <= N < b`
- `choice(seq)` — Returns a random element from the non-empty sequence `seq`
- `shuffle(seq)` — Shuffles the sequence `seq` in place

String methods

- `upper()` — Returns a string in all uppercase
- `lower()` — Returns a string in all lowercase
- `capitalize()` — Returns a string with first character capitalized, the rest lower
- `strip()` — Returns a string with the leading and trailing whitespace removed
- `count(item)` — Returns the number of occurrences of item
- `replace(old, new)` — Replaces all occurrences of old substring with new
- `find(item)` — Returns the leftmost index where the substring item is found, or -1 if not found
- `index(item)` — Like find except causes a runtime error if item is not found
- `split(separator)` — Return a list of the words in the string, using separator as the delimiter string
- `join(lst)` — Return a string which is the concatenation of the strings in lst
- `isalpha()` — Return True if all characters in the string are alphabetic and there is at least one character
- `isdigit()` — Return True if all characters in the string are decimal characters and there is at least one character
- `islower()` — Return True if all cased characters in the string are lowercase and there is at least one cased character
- `isspace()` — Return True if there are only whitespace characters in the string and there is at least one character
- `isupper()` — Return True if all cased characters in the string are uppercase and there is at least one cased character

List methods

- `append(item)` — Adds a new item to the end of a list
- `insert(position, item)` — Inserts a new item at the position given
- `extend(lst)` — Extend the list by appending all the items from lst
- `pop()` — Removes and returns the last item
- `pop(position)` — Removes and returns the item at position
- `sort()` — Modifies a list to be sorted
- `reverse()` — Modifies a list to be in reverse order
- `index(item)` — Returns the position of first occurrence of item
- `count(item)` — Returns the number of occurrences of item
- `remove(item)` — Removes the first occurrence of item
- `copy()` — Return a clone of the list
- `clear()` — Remove all items from the list

File methods

- `read(n)` — Reads and returns a string of n characters, or the entire file as a single string if n is not provided.
- `readline(n)` — Returns the next line of the file with all text up to and including the newline character. If n is provided as a parameter than only n characters will be returned if the line is longer than n.
- `readlines(n)` — Returns a list of strings, each representing a single line of the file. If n is not provided then all lines of the file are returned. If n is provided then n characters are read but n is rounded up so that an entire line is returned.
- `write(astring)` — Add `astring` to the end of the file.

Dictionary methods

- `keys()` — Returns a view of the keys in the dictionary
- `values()` — Returns a view of the values in the dictionary
- `items()` — Returns a view of the key-value pairs in the dictionary
- `get(key)` — Returns the value associated with key; None otherwise
- `get(key, alt)` — Returns the value associated with key; alt otherwise