Computer Science 1 — CSci 1100 — Fall 2017
Exam 3
November 20, 2017

Name: Ruoyan Wu

RCS ID: wur7 @rpi.edu

RIN#: 66184974

Honor pledge: On my honor I have neither given nor received aid on this exam.

Please sign here to indicate that you agree with the honor pledge: _____

Instructions:

- You have 90 minutes to complete this test.
- Clearly print your name, RCS ID (in all capital letters) and your RIN at the top of your exam.
- You must **have your Student ID** and you must be seated in the **correct section**. Failure may incur up to a **20 point penalty for each infraction.** If you are not sure if you are in the right section, please see a proctor before the exam starts.
- You may use only one double-sided crib sheet. Put your name and your RCS ID on it and turn it in at the end of the exam. Otherwise, put away all books, laptop computers, and electronic devices.
- Please read each question carefully several times before beginning to work.
- We generally will not answer questions except when there is a glaring mistake or ambiguity in the statement of a question.
- There are no Python syntax errors anywhere on this test.
- Unless otherwise stated, you may use any technique we have covered thus far in the semester to solve any problem.
- Please state clearly any assumptions that you have to make in interpreting a question.
- There are **five questions** on this test worth a total of **100 points**.
- When you are finished with this test please turn it into one of the proctors along with your crib sheet. After you show the proctor your student id you will be free to leave the exam room.

Write your answers entirely within the boxes below.

1. **(12 points)** Show the output of the following:

Code:	Answer:

`# Part (a)`
`d = dict()`
`u = 0`
`v = 1`
`for i in [0, 1, 2, 3]:`
`  val = u+v`
`  d[val] = u, v`
`  u = v`
`  v = val`
`for key in sorted(d):`
`  print("d[{}]: {}".format(key, d[key]))`
	d[1] : (0, 1)
	d[2] : (1, 1)
	d[3] : (1, 2)
	d[5] : (2, 3)

`# Part (b)`
`class s(object):`
`  def __init__(self, a=0, b=2, c=1):`
`  self.a = a`
`  self.b = b`
`  self.c = c`
`  def __add__(self, o):`
`  return s(self.a+o.a, \`
`  self.b+o.b, self.c+o.c)`
`  def s_print(self):`
`  print(list(range(self.a, self.b, self.c)))`
`o = s()`
`o.s_print()`
`p = o + o`
`p.s_print()`
	[0, 1]
	[0, 2]

Write your answers entirely within the boxes below.

2. **(12 points)** For each of the following write a **short section of Python code** to solve the problem. You can assume you are typing the command into the Python Shell. You must only use techniques we have covered thus far in the course. Any use of a for loop or a while loop will result in a 0 for that answer. Answers of more than **4 lines** will incur a penalty.

Code:	Answer:

(a) Given a string `s` containing only letters of the alphabet write an expression to generate a list of all the consonants in `s` in alphabetical order. Do not repeat characters. You can assume variable `s` has been defined in your program (note that the example below is just an example, write a solution for any potential value of `s`). As an example, if `s` is given as:
`s = "bumblebee"`
your code should evaluate to:
`["b", "m", "l"]`
	`sorted(set(list(s).remove('a').remove("e").\`
	`.remove("i").remove("o").remove("u"))`

(b) Given a dictionary `d` whose keys and values are both strings, write an expression that evaluates to `True` if and only if the smallest key is lexically before every value and the largest key is lexically after every value. As examples, the expression should evaluate to `True` for
`d = {'cow': 'herd', 'lion':'pride', \`
`'zebra':'dazzle'}`
and to `False` for
`d = {'cow': 'herd', 'lion':'pride', 'dog':'pack'}`
and also `False` for
`d = {'horse': 'herd', 'lion':'pride', \`
`'zebra':'dazzle'}`
	`print(list(d.keys()) == sorted(d.keys()))`

(c) Given sets `s1` and `s2` write code that evaluates to `True` if the size of the symmetric difference between the two sets is exactly twice the size of the intersection between the sets. For example, given
`s1 = {0, 1, 2, 3, 4, 5}`
`s2 = {0, 2, 4, 6, 8, 10}`
your code should evaluate to `True` and given
`s1 = {0, 1, 2, 3, 4, 5}`
`s2 = {0, 2, 4, 6, 8}`
Your code should evaluate to `False`.
	`print( S1^S2 == (S1 & S2)*2 )`

3. (18 points) We are going to revisit Homework 2. Suppose you are given two sets of words: pos and neg representing words with a positive (pos) and negative (neg) sentiment.

   (a) Write a function score that takes as input a string and two sets pos and neg. The function returns a "sentiment" score as follows: Compute a total score for each word in the sentence, +5 per word in the positive set and -5 per word in the negative set. Then, divide the total score by the number of unique words in the sentence that have no positive or negative sentiment (i.e. do not appear either in pos or neg) and return this value.

   For example if you were given the following sets in your program:

```
pos = {'great', 'happiness', 'happy'}
neg = {'bad', 'sad', 'terrible'}
```

   We would expect the following:

```
>>> score(pos, neg, "Here comes bad news Because I'm happy happy")
1.0
```

   There are two positive words in this sentence, happy and happy and one negative word bad, giving a total score of 5. There are exactly five words that are neutral: 'here', 'comes', 'news', 'because', "I'm", hence the score is 5/5 = 1. You can assume there will always be at least one neutral word (do not be concerned about division by 0) and do not worry about any punctuation symbols in the input word (do not write any special code to remove them, leave them as is).

```
def score ( pos, neg, string ):
    positive_score = 0
    negative_score = 0
    neutral_words = 0
    for word in string:
        word = word.lower()
        if word in pos:
            positive_score += 5
        elif word in neg:
            negative_score += 5
        else:
            neutral_words += 1
    return float( (positive_score - negative_score) / neutral_words ).
```

(b) Now write the rest of the program. First read in the sets pos and neg from a file named `sentiment.txt`. Positive words are given on the first line of the file and negative words are on the second line; both lines are separated by spaces. The file will be all lower case.

Your program should then read in a sentence from the user and print out a message about how happy the person is based on the score returned from the `score` function and the following:

i. `score > 0.5` then print `thumbs up`

ii. `score between 0.5 and -0.5` then print `meh`

iii. `score < -0.5` then print `thumbs down`

For example, if the file `sentiment.txt` contains the following (the file may contain many more words, this is only given as an example):

```
great happiness happy
bad sad terrible
```

we would expect the contents of the sets to be

```
pos = {'great', 'happiness', 'happy'}
neg = {'bad', 'sad', 'terrible'}
```

and a full run of the program in this case will be:

```
Sentence: Here comes bad news Because I'm happy Happy
thumbs up
```

```
pos = set()
neg = set()
f = open('sentiment.txt')
s = f.read().strip().split('\n')
positive = s[0].split(' ')
negative = s[1].split(' ')
for word in positive:
    pos.add(word)
for word in negative:
    neg.add(word)
sentence = input("Sentence: ")
if score(pos, neg, sentence) > 0.5:
    print("thumbs up")
elif score(pos, neg, sentence) > -0.5 and score(pos, neg, sentence) < 0.5:
    print("meh")
elif score(pos, neg, sentence) < 0.5:
    print("thumbs down")
```

4. **(28 points)**

Assume you have 2 dictionaries **children** and **chores**. Children is keyed by the ~~child's~~ name and contains a ~~list~~ set of the chores the child completed that week. Chores is keyed by the name of the chore and contains the allowance dollars given for completing that chore. You can assume that **children** and **chores** are given to you. As an example, the following could be valid values for the dictionaries:

```
children = {'Sam': {'dishes', 'pets'}, 'Chris': {'lawn', 'dishes', 'garbage'}, \
            'Pat': {'sweeping', 'dishes'}}
chores = {'lawn': 10, 'dusting': 2, 'sweeping': 2, 'dishes': 4, 'pets': 1, 'garbage': 2}
```

You can assume that both dictionaries have been defined elsewhere in your program. Note that this is just a possible example of the dictionaries. The dictionaries may have more or fewer entries and different values.

(a) A chore is **not completed** if it appears as a key in **chores**, but is not claimed by any child. Write a function **find_undone** that takes the dictionaries **children** and **chores** and returns the **set** of all chores that have not been completed. For the above dictionary definitions, we would have:

```
>>> print(find_undone(children, chores))
{'dusting'}
```

```
def find_undone (children, chores):
    done = set ( )
    for child in children:
        for item in children[child]:
            done. add (item)

    need_done = set( )
    for chore in chores:
        need_done. add (chore)
    return need_done - done
```

(b) Now write a function count that uses only the children dictionary to derive and return a new dictionary that has chores as the key and the number of children who helped with the chore as the value. Since this only uses the children dictionary, there will be no entry for chores that were left undone (such as 'dusting' in our example above). For our example data, we would expect:

```
>>> print(count(children))
{'lawn': 1, 'sweeping': 1, 'garbage': 1, 'dishes': 3, 'pets': 1}
```

```
def count(children):
    chores = dict()
    for child in children:
        for chore in children[child]:
            if chore in chores:
                chores[chore] += 1
            elif chore not in chores:
                chores[chore] = 1
    return chores.
```

(c) When a child does a chore by themselves, they receive the full allowance for the chore. When they work together, each receives an equal share of the total chore value. Write a program that prints out the chores that were left undone and how much each child was paid. Undone chores and children must be printed in lexicographic order.

i. You **must** use `find_undone` and `count` in your program.

ii. You **can assume** `children` and `chores` are already defined.

iii. You **can assume** that any chore a child completes is found as a key in `chores`

iv. You **cannot assume** that `children` and `chores` are identical to the values below.

For the dictionaries:

```
children = {'Sam': {'dishes', 'pets'}, 'Chris': {'lawn', 'dishes', 'garbage'}, \
        'Pat': {'sweeping', 'dishes'}}
chores = {'lawn': 10, 'dusting': 2, 'sweeping': 2, 'dishes': 4, 'pets': 1, 'garbage': 2}
```

Your program would print

```
Undone:
dusting
Chris: $13.33
Sam: $2.33
Pat: $3.33
```

```
children_paid = [ ]
print ("Undone:")
for work in sorted ( find_undone(children, chores)):
    print work

for child in children:
    paid = 0
    for item in children[child]:
        paid += chores[item] / count(children)[item]

    children_paid.append ((child, paid))
children_paid.sort ( )
for child in children_paid:
    print (child[0] + ": ${:.2f}".format( child[1] ))
```

5. **(30 points)**

You have data about houses for sale stored in a file `houses.txt`. The data is stored one house per line with commas separating fields. The fields are (in order):

```
id_num, bedrooms, bathrooms, offers
```

These refer to, in turn, the listing number (an integer), the number of bedrooms, the number of bathrooms, and a list of floats representing all the current offers on the house. An example file is:

```
7008961, 2, 4, 300000, 200000, 350000
7008969, 1, 2
7868961, 2, 5, 325000, 1200000, 350000, 400000
7869961, 4, 1, 315000, 100000
```

Note that this is just representative of the lines in the file. The file may have more lines, more or fewer offers, and different values. The fields `bedrooms`, `bathrooms` and `id_num` field will always be there, but there may not be any offers. The file has no header line.

(a) Write the definition for the class `House` with `Attributes` to hold the data from the file and a single `method` that performs initialization. For full points, use the best practices we illustrated in class. You should assume you create the class in a file name `House.py`.

```
class House (object):
    def __init__(self, id_num, bedrooms, bathrooms, offers)
        self.id_num = id_num
        self.bedroom = bedrooms
        self.bathroom = bathrooms
        self.offers = list(offers)
```

(b) Now add 2 additional methods to your class. The first should be an implementation of the `__str__` function and the second should be a method named `filter` that takes a target number of bathrooms and bedrooms and returns `True` if the house has at least as many bathrooms and bedrooms as desired. For example, assuming you use a list to hold the offers, the following code

```
h = House(7008961, 2, 4, [300000, 200000, 350000])
print(h.filter(1, 5))
print(h.filter(1, 4))
print(h)
```

should output

```
False
True
House 7008961 with 2 bathrooms and 4 bedrooms, has 3 offers.
```

```
def __str__(self):
    return "House" + str(self.idnum) + "with" + str(self.bathrooms) + "bathrooms" \
        + "and" + str(self.bedrooms) + "bedrooms, has" + str(len(self.offers)) + \
        "offers."

def filter(self, bathrooms, bedrooms):
    return self.bathrooms >= bathrooms and self.bedrooms >= bedrooms
```

(c) Assume the class House exists and works correctly. In a **SEPARATE** file, `main.py`, write code to read the lines from `houses.txt`, create a house from each line, and write out to a second file `valid.txt` those houses that have at least 2 bathrooms and 3 bedrooms. Your solution must use the `filter` and `__str__` methods you created.

For the `houses.txt` above, the file `valid.txt` would contain:

```
House 7008961 with 2 bathrooms and 4 bedrooms, has 3 offers.
House 7868961 with 2 bathrooms and 5 bedrooms, has 4 offers.
```

```
from House.py import class House.
f = open('houses.txt')
S = f.read().strip().split('\n')
f.close()
f_write = open('valid.txt', 'w')
for house in S:
    house = house.split(',')
    id_num = house[0]
    bedrooms = house[1]
    bathrooms = house[2]
    offers = int[3:]
    house = House(id_num, bedrooms, bathrooms, offers)
    if house.filter(2,3):
        f_write.write(__str__(house))
f_write.close()
```

This page is left blank for scratch work.