Answer Key - Spring 2020 Exam 2

1) **TABLE COMPLETION** [18 pts] (3 points each) Pretend you are the python interpreter. Evaluate each of the expressions below. Write down the value that each evaluates to. If your answer is a string include quotes around your answer (i.e "hello"). If your answer is a float make sure you include the decimal (i.e 5.0). Write the word error in both columns if the expression causes an error.

Table 1: Expression

Expression	Return Value of Expression (2 pts)	Data Type of Expression (1 pt)
("frappuccino",)[0][2:6:-1]	'ar' (not included)	str
[[2.5, 9.0], 13.1][0][1] + 3	12.0	float
aList.append("d")	None	NoneType
{1:2, 2:4, 3:6}[1]	2	int
("i","love","coffee") - ("coffee",)	error	error
len({"a":1, "b":4, "c":2, "a":5})	3	int

2) **MULTIPLE CHOICE** [36 pts] (3 points each) For each of the following multiple choice questions, **indicate the best answer by bubbling in the corresponding circle.** Points may be deducted otherwise.

a) Which of the following would correctly change a tuple named aTup from ("Refreshers", "Lattes") to ("Mochas", "Refreshers", "Lattes")

- ○ A: aTup.append("Mochas")
- ○ B: aTup = ("Mochas") + aTup
- ● C: aTup = ("Mochas",) + aTup
- ○ D: A and C
- ○ E. All of the above

b) Given aDict = {"Coffee" : {"Dunkin" : ("Quincy", "MA"), "Starbucks": ["Seattle", "WA"]}}, what is aDict["Coffee"]["Starbucks"][0]*2

- [x] A: "SeattleSeattle"
- [ ] B: ["Seattle", "Washington", "Seattle", "Washington"]
- [ ] C: 3900
- [ ] D: "QuincyQuincy"
- [ ] E: aDict is not a valid dictionary

c) Which of the following statements is true about closing files?

- [ ] A: Data will not be retrieved if you don't close the file that you have opened for reading.
- [x] B: Data will not be written to a file if you don't close the file that you have opened for writing.
- [ ] C: Both A and B
- [ ] D: None of the above

d) Which of the following is a valid dictionary?

- [ ] A: drink_order = {~~print("pink_drink"):~~ None 5, "americano": {}}
- [ ] B: drink_order = {"frappuccino": 1.0, 3.0: True}
- [ ] C: drink_order = {("peach_lemonade",): [1.0, ["peppermint mocha"]]}
- [ ] D. B and C
- [x] E. All of the above

e) Which of the following lines of code would remove the commas from a variable named **frap** that contains the value "ice, powder, and milk"?

- ○ A: `frap -= ","`
- ● B: `frap = frap.split(",")[0] + frap.split(",")[1] + frap.split(",")[2]`
- ○ C: `frap = frap.strip(",")` ends
- ○ D: B and C
- ○ E: None of the above

f) Which of the following lines of code correctly adds the string "cold brew" to the variable **morning_coffee** containing a list?

I. `morning_coffee.append("cold brew")`
~~II.~~ `morning_coffee += "cold brew"`
III. `morning_coffee = morning_coffee + ["cold brew"]`

- ○ A: I only
- ○ B: II only
- ○ C: III only
- ● D: I and III only
- ○ E: I, II, and III

g) How many lines of code would be printed to the shell after running the following code?

```
def func(tupList):
    try:
        for num1, num2 in tupList:
            if num1 + num2 == 0:
                print("bucks")
    except:
        print("star")
    finally:
        return("done")

tupList = [(0,0),(-2,1),(-8,8),(3,-2,9),(-5,5)]
print(func(tupList))
```

bucks
bucks
star
done

- O A: 3
- ● B: 4
- O C: 5
- O D: 6
- O E: None of the above

h) What are the possible integers that the variable **latte** could contain when the following code is executed?

```
import random as r
latte = r.randrange(4,13,3)
```

4 7 10

- O A: 4 5 6 7 8 9 10 11 12
- O B: 4 7 10 13
- ● C: 4 7 10
- O D: 4 5 6 7 8 9 10 11 12 13
- O E: None of the above

i) Which of the following ways could you import the **cosine** function from the **math** module and use it on the int **num**?

- O A: from math import * | math.cos(num)
- O B: from math import cos | cos(num)
- O C: import math as english | english.cos(num)
- ● D: B and C
- O E. All of the above

Use the following code to answer parts j, k, and l.

```
aList = ["coffee", ["Blue Donkey", "Dancing Goats"]]
bList = aList[:]
cList = aList
aList.append("tea")
cList.append(1971)
cList[3] -= 5
aList[1].append("latte")
```

j) What is the value of **aList** after the code above is run?

- O A: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea"]
- ● B: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea", 1966]
- O C: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea", 1971]
- O D: None (NoneType)
- O E: None of the above

k) What is the value of **bList** after the code above is run?

- ● A: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"]]
- O B: ["coffee", ["Blue Donkey", "Dancing Goats"]]
- O C: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea", 1966]
- O D: None (NoneType)
- O E: None of the above

l) What is the value of **cList** after the code above is run?

- O A: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea"]
- ● B: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea", 1966]
- O C: ["coffee", ["Blue Donkey", "Dancing Goats", "latte"], "tea", 1971]
- O D: None (NoneType)
- O E: None of the above

3) **Tracing** [16 pts] (4 points each) Show exactly what would be printed out when each of the following segments of code are executed. None of these code segments will cause an error. They all have at least partial output that would be shown.

a)

```
def traceMe(aStr):
    myList = aStr.split()
    first = open("aFile.txt","w")
    first.write(myList[1])
    first.close()
    second = open("aFile.txt", "w")
    second.write(myList[3])
    third = open("aFile.txt")
    print(third.read())


print(traceMe("let's go get coffee!"))
```

myList = ['let's', 'go', 'get', 'coffee']

coffee!
None

b)

```
def func(aDict):
    newDict = {}

    for item in aDict.items():
        a, b = item
        newDict[b] = a
        newDict[a] = b
    return newDict

dictList = {"iced":"latte",
"coffee":"iced", "mocha":"latte"}

print(func(dictList))
```

{'latte': 'mocha', 'iced': 'coffee', 'coffee': 'iced', 'mocha': 'latte'}

adict.items() = [('iced', 'latte'), ('coffee', 'iced'), ('mocha', 'latte')]

newdict: {'latte': ~~'iced'~~ 'mocha', 'iced': ~~'latte'~~ 'coffee', 'coffee': 'iced', 'mocha': 'latte'}

c)

```
def tupDrank(aTup, bTup):
    try:
        flavor, drink = aTup = ('caramel'
        calorie = aTup[0] + flavor 'Macc.')
        print(drink)          → Caramel caramel'
        print(calorie/24)
    except:
        print("too many calories!")
    finally:
        print("good night")
tupDrank(("Caramel", "Macchiato"), (800,
"tall"))
```

Macchiato
too many calories!
good night

d)

```
def popPrince():
    aDict = {"prince":["Biebs"]}
    aDict["prince"] = ["Harry Styles"]
    for value in aDict:
        print(aDict[value])
    aDict["prince"][0] = "Shawn Mendes"
    print(aDict)

popPrince()
```

["Harry Styles"]
{"prince": ["Shawn Mendes"]}

adict: {"prince": ['~~Harry Styles~~']
['Shawn Mendes']

4) **CODING** [10 pts] Write a function called **surpriseMe** that takes in two parameters. The first parameter is a list of tuples where each tuple has three ingredients (strings) and the last element of the tuple states if the drink is hot or cold (string). The second parameter is your preference for a hot or cold drink (string). The function should create and return a list of tuples from the original list that matches your hot or cold drink preference. You may assume that the tuples will always contain four strings.

Example Output #1:

```
>>> recipes = [("milk", "mocha syrup", "espresso", "hot"), ("chai",
"cinnamon", "almond milk", "hot"), ("whipped cream", "espresso",
"chocolate", "cold")]
>>> preference = "cold"
>>> print(surpriseMe(recipes, preference))
[("whipped cream", "espresso", "chocolate", "cold")]
```

```
def surpriseMe(tuplist, pref):
    newlist = []
    for recipe in tuplist:
        if recipe[3] == pref:
            newlist.append(recipe)
    return newlist
```

5) **CODING** [10 pts] Write a function called **bulkOrder** that takes in one parameter, a list of drinks. Return a dictionary that maps each drink to the number of times that the drink appears in the list. You can assume all drinks will be in lowercase.

Example Output #1:

```
>>> drinks = ["frap", "americano", "frap", "espresso", "espresso"]
>>> print(bulkOrder(drinks))
{"frap": 2, "americano": 1, "espresso": 2}
```

```
def bulkOrder(drinks):
    result = {}
    for drink in drinks:
        if drink in result.keys():
            result[drink] += 1
        else:
            result[drink] = 1
    return result
```

6) **CODING** [10 pts] Write a function named **sbux_menu** that takes in the name of a file. This file will contain one drink on each line. Your function should return a tuple of the drinks, where each drink is one element of the tuple. You may assume that the file name will always be valid and that the file is in the same directory as the code.

Hint: Don't forget to remove newline characters!

Example Output #1:

```
>>> print(sbux_menu("sbux.txt"))
("Pink Drink", "Espresso", "Americano", "Cortado")
```

Example sbux.txt (actual file could have many more lines)

```
Pink Drink
Espresso
Americano
Cortado
```

```
def sbux_menu(filename):
    myfile = open(filename, "r")
    alist = myfile.readlines()
    myfile.close()
    result = ()
    for drink in alist:
        drink = drink.strip()
        result += (drink,)
    return result
```