ANSWER KEY

1) **TABLE COMPLETION** [16 pts] (2 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 floating point number 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 (1.5 pts)	Data Type of Expression (0.5 pt)
len([("apple", "orange"), ("banana", "grape"), "pineapple"])	3	int
("oreos",) + ("ice cream",)	("oreos", "ice cream")	tuple
["pancakes", "bagels", "waffles", "bacon"].sort()	None	NoneType
["pie"] * 2	["pie", "pie"]	list
len({~~"biscuits": 2,~~ "butter": 3, "biscuits": 1})	2	int
{"hot chocolate": "marshmallows", "cake": "frosting"}["hot chocolate"][0]	"m"	str
2 in {"latte": 1, "mocha": 2, "espresso": 9, 2: "americano"}	True	bool
[9, 3, 4, (4,5)][3]+(3,) 0 1 2 3	(4,5,3)	tuple

2. **MULTIPLE CHOICE** [9 pts] (3 pts each) For each multiple choice question below, indicate the best answer by filling in the corresponding circle.

a) Which of the following expressions would change the value of **flavors** from ["vanilla", "chocolate", "strawberry"] to [["cotton candy"], "chocolate", "strawberry"]?

○ ~~A. flavors.append(["cotton candy"])~~

○ ~~B. flavors.remove("vanilla")~~

● C. flavors = [["cotton candy"]] + flavors[1:]

○ D. flavors[0] = [["cotton candy"]]

○ E. None of the above

b) What will be printed by the following code?

```
aDict = {~~'ramen': 0,~~ 'ramen': 1, 'pho': 0}
print(aDict)
```

○ A. {'pho': 0, 'ramen': 1, ~~'ramen': 0~~}

○ B. {~~'ramen': 0,~~ 'pho': 0}

● C. {'ramen': 1, 'pho': 0} ✓

○ D. A and C

○ E. None of the above

c) After the following lines of code are run, what will be the value stored in the variable **foodStr**?

```
foodStr = "\n\tpizza time\t\n"
foodStr = foodStr.strip()
```

○ A. "\n\tpizza time\t\n"

○ B. "\tpizza time\t"

○ C. "pizzatime"

● D. "pizza time"

○ E. None of the above

2. **SHORT ANSWER** [14 pts]

(4 pts)
a) In the box below, write an import statement that allows this code to print the value of pi when run (pi is a variable in the **math** module)

```
________ #import statement
print(m.pi)
```

import math as m

(5 pts)

b) Given the dictionary **aDict**, write one line of code which will change the value mapped to the key "cereal" from its current value "raisin bran" to the value "fruit loops".

```
aDict = {"breakfast" : {"cereal" : "raisin bran"},
         "lunch" : "sandwich"}
```

```
aDict["breakfast"]["cereal"] = "fruit loops"
```

(5 pts)

c) In the box below, define a new variable, **newPieList**, that is a **clone** of pieList.

```
pieList = ["pumpkin", "pecan", "apple"]
```

```
newPieList = pieList[:]
```

4) **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.

flavor	price
blue	bluebenes, 4.50
brown	cocoa powder, 2.0
red	cherry, 7.00

```
a) def makePie(flavors):
        prices = []
        for flavor, price in flavors.items():
            if len(flavor) % 2 == 0:
                print("Yum " + flavor + "!")
            elif price[1] == 2.00:
                prices.append(price[0])
                print("Perfect!")
        return prices

    flavors = {"blue": ("blueberries",
    4.50), "brown": ("cocoa powder", 2.00),
    "red": ("cherry", 7.00)}

    print(makePie(flavors))
```

prices
cocoa powder

```
Yum blue!
Perfect!
["cocoa powder"]
```

j	name	price
~~8~~	paprika	4.0
2.0	garlic	0.0
	cumin	2.0

j
~~8~~
2.0
error

```
b) def shoppingList(items):
        j = 8
        for name, price in items:
            try:
                print("I love " + name)
                j /= price          → 2nd error!
            except:
                print("I love " + name)
            finally:
                if price == 0.0: ✓
                    break

itemList = [("paprika", 4.0),  ("garlic", 0.0), ("cumin", 2.0)]
print(shoppingList(itemList))
```

None!

```
I love paprika
I love garlic
I love garlic
None
```

a
[illegible]
[illegible]
[illegible]
[illegible] 3

x	y
0	1, eggs
1	0, sugar

a	
~~0~~	~~1~~
~~0~~	3
~~2~~	

c)
```
def recipe(foods):
    a = 0
    for x, y in enumerate(foods):
        a = x
        a += len(y)
        print(y[x])
    print(a)

foods = [(1,"eggs"), (0,"sugar")]
print(recipe(foods))
```

None

```
1
sugar
3
None
```

d)
```
def foodReview(tupA, listB):
    x,y,z = tupA
    b = listB[:]
    b.append("bread")
    x = b.sort()
    z = sorted(b)
    print(x)
    print(z)
    return(y)

tupA = ("donut", "honey", "cupcake")
listB = ["wow", "good"]
print(foodReview(tupA, listB))
```

```
None
["bread", "good", "wow"]
honey
```

x	y	z	b	
~~donut~~	honey	~~cupcake~~	wow	→ bread
None		bread	good	good
		good	bread	wow
		wow		

LONG ANSWER [5 pts]

The following function, longDrink(), should take in one parameter which is a list of drink names (*str*). It should return a dictionary, which maps the string "short" to a list of drink names which have a length less than or equal to 4, and should map the string "long" to a list of drink names which have length greater than 4. Fill in the 3 missing lines of code to complete the function.

```
def longDrink(drinkList):
    drinkDict = {"short": [], "long": []}
    for drink in drinkList:
        #write a single line of code that will check the length
        #of drink to see if it is less than or equal to 4
        1.________________________________
            key = "short"
        else:
            key = "long"
        #write a single line of code that will add drink to the
        #list mapped to the key variable
        2.____________________________
    #write a single line of code for the return statement
    #return whatever value you believe is correct
    3.________________________
```

Write answers in the boxes below:

1.

if len(drink) < = 4:

2.

drinkDict[key].append(drink)

3.

return drinkDict

CODING [40 pts]

CODING 1 [12 pts] - Write a function called **candyShop()** that takes in three parameters: a dictionary of candies (dict), your preferred candy type (str), and your allowance (float). The dictionary will contain a candy type (str) mapped to a list of tuples containing the candy name (str) and price (float).

Return a list that contains the names of candies that are your preferred type and have a price less than or equal to your allowance. You can assume that your preferred candy type will always be found in the dictionary.

Example Output #1:

```
>>> candies = {"chocolate": [("Twix", 3.00), ("KitKat", 1.75)],
               "hard candies": [("Jolly Ranchers", 1.30)]}
>>> candyShop(candies, "chocolate", 2.50)
["KitKat"]
```

Example Output #2:

```
>>> candies = {"chocolate": [("Twix", 2.75)],
          "sour candy": [("Sour Skittles", 1.10), ("Sour Patch Kids", 1.25)]}
>>> candyShop(candies, "sour candy", 1.50)
["Sour Skittles", "Sour Patch Kids"]
```

```
def candyShop(adict, preference, allowance):
  newlist = []
  for candy, price in adict[preference]:
    if price <= allowance:
      newlist.append(candy)
  return newlist
```

CODING 2 [14 pts] - Write a function called **menu()** that takes in a list of tuples in the form (foodCategory(str), foodName(str)). Your function should return a dictionary that maps food categories to a list of food names that have that category.

```
Example Output #1:
>>> foods = [("savory", "panini"), ("sweet", "crepe"), ("savory", "omelet")]
>>> menu(foods)
{"savory": ["panini", "omelet"], "sweet": ["crepe"]}
```

```
Example Output #2:
>>> foods = [("entrees", "burger"), ("sides", "salad"), ("entrees", "steak")]
>>> menu(foods)
{"entrees": ["burger", "steak"], "sides": ["salad"]}
```

```
def menu(alist):
  newdict = {}
  for category, food in alist:
    if category not in newdict:
      newdict[category] = [food]
    else:
      newdict[category].append(food)
  return newdict
```

CODING 3 [14 pts] - Write a function called **saladBar()** that takes in two parameters: a list of tuples in the form (person(str), restriction(str)) and a dictionary mapping restrictions to a list of foods corresponding to that restriction (e.g. "vegan" : ["meat"]). The function should return a dictionary that maps the name of the person to the number of foods they **cannot** eat.

```
Example Output #1:
>>> restrictionsList = [("Audrey","vegan"), ("Ramya", "dairy-free")]
>>> foodDict = {"vegan": ["chicken", "ranch", "feta cheese", "bacon", "egg"],
               "dairy-free": ["ranch", "feta cheese"]}
>>> saladBar(restrictionsList, foodDict)
{'Audrey': 5, 'Ramya': 2}
```

```
Example Output #2:
>>> restrictionsList = [("Cynthia", "nut-free"), ("Aryan", "gluten-free")]
>>> foodDict = {"nut-free": ["cashew", "peanut"], "gluten-free": ["crouton"]}
>>> saladBar(restrictionsList, foodDict)
{'Cynthia': 2, 'Aryan': 1}
```

```
def saladBar(alist, adict):
    newdict = {}
    for person, restriction in alist:
        count = len(adict[restriction])
        newdict[person] = count
    return newdict
```