CS 1301 Exam 2
Spring Semester 2022
Version A

Name (print clearly including your first and last name as it appears on your Buzzcard):

_____

Signature indicating you understand the Georgia Tech Honor Policy:

_____

GTID# (903000000, etc.):

_____

- Integrity: By taking this exam, you pledge that this is your work and you have neither given nor received inappropriate help during the taking of this exam in compliance with the Academic Honor Code of Georgia Tech. Do NOT sign nor take this exam if you do not agree with the honor code.
- Signing and/or taking this exam signifies you are aware of and in accordance with the **Academic Honor Code of Georgia Tech** and the **Georgia Tech Code of Conduct**.
- Academic Misconduct: Academic misconduct will not be tolerated. You are to uphold the honor and integrity bestowed upon you by the Georgia Institute of Technology.
    - Keep your eyes on your own paper.
    - Do your best to prevent anyone else from seeing your work.
    - Do NOT communicate with anyone other than a proctor for ANY reason in ANY language in ANY manner.
    - Follow directions given by the proctor(s).
    - Stop all writing when told to stop. Continuing to write after the end of the exam is an honor violation.
    - Do not use notes, books, calculators, phones, laptops etc. during the exam.
    - You are not allowed to leave the exam with your exam paper for any reason.
- Devices: If your cell phone, smartwatch, laptop, or similar item goes off during the exam, you will lose 10 points on this exam. Turn all such devices off and put them away now. You cannot have them on your desk.
- Extra paper is not allowed. If you have exhausted all space on this test, talk with your instructor.
- Pens/pencils and erasers are allowed. Do not share.
- All code must be in Python.

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"])		
("oreos",) + ("ice cream",)		
["pancakes", "bagels", "waffles", "bacon"].sort()		
["pie"] * 2		
len({"biscuits": 2, "butter": 3, "biscuits": 1})		
{"hot chocolate": "marshmallows", "cake": "frosting"}["hot chocolate"][0]		
2 in {"latte": 1, "mocha": 2, "espresso": 9, 2: "americano"}		
[9, 3, 4, (4,5)][3]+(3,)		

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()
```

- O A. "\n\tpizza time\t\n"
- O B. "\tpizza time\t"
- O C. "pizzatime"
- O D. "pizza time"
- O 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)
```

(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"}
```

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

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

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.

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))
```

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

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

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))
```

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))
```

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.

2.

3.

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"]
```

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"]}
```

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}
```