CS 1301 – Homework 5

Tuple and Dictionary Manipulation
Due: Monday October 3rd, 11:55pm
File to submit to T-Square: HW5.py

You should work individually on this assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 1301, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. **Collaboration at a reasonable level will not result in substantially similar code.**

For Help:

- TA Helpdesk – Schedule posted on class website.
- Email TA's or use Piazza Forums

Notes:

- **Do not forget to include the required collaboration statement** (outlined on the syllabus).
- **Do not wait until last minute** to do this assignment in case you run into problems.
- **Read the entire specifications document before starting this assignment.**

Part 1 – Tuple Manipulation

Function Name: defineTAs (15 points)

Description: Define a function that accepts a tuple of tuples as the parameter. Each tuple contains information regarding the TAs. Every index in the tuples corresponds to a TA, and a TA is only associated with **one index**. The function should concatenate the company that a TA worked for, the name of the TA, and the age of the TA into a TA bio **(represented as the following: Company: Name, Age)** and return a tuple of all of the concatenated TA bios.

Parameter(s):

1. A tuple of tuples. All of the tuples are equal in length.
    a. Each element of the first tuple is a name (as a **string**).
    b. Each element of the second tuple is the age (as an **integer**).
    c. Each element of the third tuple is the company that the corresponding person worked at (as a **string**).

Return Value: A new tuple with the correct values. Each element of the new tuple is a string of the concatenated company, name, and age (**in that order)**.

Assumption(s):

- Each of the tuples are in a specific order. Therefore, you only need to worry about the index to create the TA bios.

Example(s):
```
>>> information_tuple = (("Nico", "Daniel", "Rachel"), (21, 20, 19), ("Google",
"Workday", "Ultimate Software"))
>>> print(defineTAs(information_tuple))
('Google: Nico, 21', 'Workday: Daniel, 20', 'Ultimate Software: Rachel, 19')
```

Function Name: offset (20 points)

Description: Define a function that accepts two tuples of integers and two integers as parameters. For a tuple, **each of the integers** in that tuple corresponds to a valid **index** in the **other tuple**. The first integer parameter will only be used to index into the first tuple. The second integer parameter will only be used to index into the second tuple. You will be creating a repetitive process of updating the first integer parameter by indexing into tuple 2 with the second integer parameter, and updating the second integer parameter by indexing into tuple 1 with the first integer parameter. You should do this as long as neither of the parameters is 0 (i.e. if **either one** of the numbers is a 0 then stop).

Your function should return the path that it took jumping between the two tuples until it finds the first 0 in either of the tuples. The path is a **tuple** with each of the numbers you retrieved until you reached 0.

To better represent this say we have tuple_1: (3,2,1,0) and tuple_2: (0, 1, 2, 3). The element at index 0 for tuple_1 is 3. This 3 now represents the number which you will use to index into tuple_2. tuple_2 at index 3 is 3. This 3 now represents the number which you will use to index into tuple_1. tuple_1 at index 3 is 0. So now you will stop.

Parameter(s):

1. A tuple of integers. Each integer is a valid index for the **other** tuple.
2. A tuple of integers. Each integer is a valid index for the **other** tuple.
3. A single integer. The index of where to begin the path in the **first** tuple.
4. A single integer. The index of where to begin the path in the **second** tuple.

Return Value: A new tuple with the path to 0.

Assumption(s):

- The integers in both of the tuples are unique.
- The integers correspond to actual indexes of the other tuple.
- The integer indexes will not result in an endless loop.
- Once one of the indexes is a 0, you can stop.
- The first index (starting index) should only index into the first tuple and the second index should only index into the second tuple.

Example(s):

```
>>> number_tuple_1 = (6,4,2,7,1,9,3,5,8,0)
>>> number_tuple_2 = (3,8,1,0,9,6,5,4,7,2)
>>> index_1 = 4
>>> index_2 = number_tuple_1[index_1]
>>> print(offset(number_tuple_1, number_tuple_2, index_1, index_2))
(4, 1, 8, 8, 7, 5, 6, 3, 0)
```

Part 2 – Dictionary Manipulation

Function Name: heroEgos (15 points)

Description: Define a function that accepts a dictionary as a parameter. The dictionary maps heroes to a list of movies they appeared in. Your function should return a new list containing only the movies in which the heroes' name appears in the title.

Parameter(s):

1. A dictionary. The keys are strings and the values are lists of the.
   a. The keys are the heroes.
   b. The values are the list of movies the hero has appeared in.

Return Value: A new list containing the correct values.

Example(s):

```
>>> heroes = {"Iron Man":["Iron Man", "Iron Man 2", "Iron Man 3"],
"Batman":["Batman Begins", "The Dark Knight", "The Dark Knight Rises"],
"Captain America": ["The First Avenger", "The Winter Soldier", "Civil War"],
"Superman": ["Man of Steel", "Batman v Superman", "Superman Returns"]}
>>> print(heroEgos(heroes))
['Iron Man', 'Iron Man 2', 'Iron Man 3', 'Batman v Superman', 'Superman Returns',
'Batman Begins']
```

Function Name: createActors (15 points)

Description: Define a function that accepts two dictionaries as parameters. The first dictionary maps directors to a movie they directed. The second dictionary maps movies to an actor that starred in the movie. Your function should return a new dictionary mapping actors to a tuple containing the movie they starred in and the director of that movie. Note: Directors may have directed multiple movies, and actors may have acted in multiple movies, therefore **only match actors and directors who have the same movie in common**.

Parameter(s):

1. A dictionary of the directors. The keys are strings and the values are strings.
   a. The keys are the directors.
   b. The values are the corresponding directors' movie.

2. A dictionary of the movies. The keys are strings and the values are strings.
    a. The keys are movies.
    b. The values are the corresponding movies' actor.

Return Value: A new dictionary mapping the actor (as a **string**) to a tuple. The **first element** of the tuple is the actors' matching movie (as a **string**) and the **second element** of the tuple is the director (as a **string**) of the matching movie.

Example(s):

```
>>> directors = {"Steven Spielberg": "Raiders of the Lost Ark",
          "George Lucas": "Star Wars",
          "Christopher Nolan": "Inception",
          "James Cameron": "Terminator 2: Judgement Day",
          "Peter Jackson":"Lord of the Rings: The Two Towers",
          "Quentin Tarantino":"Pulp Fiction",
          "Joss Whedon": "The Avengers"}
>>> movies = {"Raiders of the Lost Ark": "Harrison Ford",
        "The Terminator": "Arnold Schwarzenegger",
        "Lord of the Rings: The Two Towers":"Ian McKellen",
        "Star Wars: Episode VII": "Daisy Ridley",
        "Pulp Fiction":"Samuel L Jackson",
         "The Dark Knight": "Christian Bale",
         "The Avengers":"Chris Evans"} (4, 1, 8, 8, 7, 5, 6, 3, 0)
>>> print(createActors(directors, movies))
{'Harrison Ford': ('Raiders of the Lost Ark', 'Steven Spielberg'), 'Chris Evans':
('The Avengers', 'Joss Whedon'), 'Samuel L Jackson': ('Pulp Fiction', 'Quentin
Tarantino'), 'Ian McKellen': ('Lord of the Rings: The Two Towers', 'Peter
Jackson')}
```

Function Name: updateNetworks (20 points)

Description: Define a function that takes in a dictionary and a list of tuples as parameters. The dictionary maps networks to a list of TV shows on that network. Each of the tuples contains a network and a TV show. Your function should update the networks dictionary using the tuples according to the following guidelines. If the network in the tuple is **not** in the dictionary, then you should add it to your dictionary with the value being a **list** of that networks' TV show. If your network is in the dictionary, then if the TV show is **not** in that networks' list of TV shows, then it needs to add the TV show to the list of TV shows for that corresponding network. Otherwise, if your network is in your dictionary and the TV show is in that networks' list of TV shows, then you should delete that TV show from the list of networks. You will then return the updated dictionary.

Parameter(s):

1. A dictionary. The keys are strings and the values are lists of strings. The list may be empty.
    a. The keys are the networks.
    b. The values are the corresponding networks' TV shows.
2. A list of tuples. Each tuple has two elements, both strings.
    a. The first element of each tuple is a network
    b. The second element of each tuple is a TV show.

Return Value: The original dictionary with the updated values.

Example(s):

```
>>> networks = {"Netflix": ["Daredevil"],
        "ABC": [],
        "CBS": ["The Big Bang Theory", "Two and a Half Men"],
        "NBC": ["30 Rock", "Parks and Recreation", "The Office"],
        "HBO": ["Last Week Tonight"]}
>>> tv_shows = [("Netflix", "Luke Cage"), ("Netflix", "Stranger Things"),
        ("The CW", "Jane the Virgin"), ("CBS", "Two and a Half Men"),
        ("ABC", "Fresh Off the Boat"), ("NBC", "Parks and Recreation"),
        ("NBC", "The Office"), ("NBC", "30 Rock"),
        ("The CW", "Pretty Little Liars")]
>>> print(updateNetworks(networks, tv_shows))
{'The CW': ['Jane the Virgin', 'Pretty Little Liars'], 'ABC': ['Fresh Off the Boat'],
'HBO': ['Last Week Tonight'], 'CBS': ['The Big Bang Theory'], 'Netflix':
['Daredevil', 'Luke Cage', 'Stranger Things'], 'NBC': []}
```

Function Name: artistMishap (15 points)

Description: Define a function that takes in a dictionary as a parameter. The dictionary maps artists to the number of solo records they have (not actually correct). Your function should update each of the solo records according to the following criteria:

- If the number is odd:
  - If the number is a multiple of 7 then increment the number by 7
  - If the number is a multiple of 5 then decrease the number by a factor of 5 (keep it as an integer)
  - Else if the number is a multiple of 3 then decrement the number by 3
- If the number is even:
  - If the number is a multiple of 7 then decrement the number by 7
  - Else if the number is a multiple of 5 then increment the number by 5
  - If a number is a multiple of 3 then increment the number by a factor of 3
  - **Always** square the number

Note: Some number may be a multiple of any combination of 3, 5, or 7. Also, the numbers in your dictionary should and will change as you go through the different cases. It is your job to make sure that number is only changed according to the correct criteria (i.e. if the number is 105 that means it is odd and a multiple of 3, 5 and 7. However we will **only** increment the number by 7 because the updated value in the dictionary after we check to see if it is a multiple of 7 is now **112,** which is not a multiple of either 5 or 3).

After updating the values in the dictionary, your function should return a tuple of tuples. The first tuple contains the artist with the minimum number of solo records, and the second tuple contains the artist with the maximum number of solo records. Each of the tuples should be represented as (artist, number_of_records), and both of these tuples should be placed within one tuple.

Parameter(s):

1. A dictionary. The keys are strings and the values are integers.
   a. The keys are the artists.
   b. The values are the corresponding artists' solo records.

Return Value: A tuple containing two tuples. The **first tuple** is the artist with the **minimum** number of solo records. The **second tuple** is the artist with the **maximum** number of solo records. Each of the tuples has the **artist** (as a **string**) as the **first element** and **the number of solo records** (as an **integer**) as the **second element.**

Example(s):

```
>>> artists = {"Shakira": 65, "Carlos Vivez": 42,
        "Juanes": 38, "Neon Trees": 30,
        "Walk the Moon": 15, "Kacey Musgraves": 7,
        "Justin Timberlake": 63, "Michael Jackson": 70,
        "Michael Buble": 35, "Papa Roach": 26,
        "Disturbed": 21, "Lil Jon": 30}
>>> print(artistMishap(artists))
(('Walk the Moon', 3), ('Michael Jackson', 35721))
```

Rubric

defineTAs	**15**
correct function header	3
returns result of correct type	2
result is correct	10
offset	**20**
correct function header	3
returns result of correct type	2
result is correct	15
heroEgos	**15**
correct function header	3
returns result of correct type	2
result is correct	10
createActors	**15**
correct function header	3
returns result of correct type	2
result is correct	10
updateNetworks	**20**
correct function header	3
returns result of correct type	2
result is correct	15
artistMishap	**15**
correct function header	3
returns result of correct type	2
result is correct	10

(-10) For relying on code outside of the functions, excluding import statements (In other words, only write your code inside your functions! Don't have global variables and don't call your functions in your code)

(-100) No credit will be given for any syntax errors in your code!