CSE101 – Spring 2021
Programming Assignment #4

Due April 15, 2021 by 11:59pm, KST. The assignment is worth 14 points.

Instructions

For each of the following problems, create an error-free Python program.

- Each program should be submitted in a separate Python file that follows a particular naming convention: Submit the answer for problem 1 as "Assign4Answer1.py" and for problem 2 as "Assign4Answer2.py" and so on.
- These programs should execute properly in VS Code using the setup we created in lab.
- At the top of every file add your name and Stony Brook email address in a comment.
- Include all the provided test cases in your solutions – for test cases that just return a value, make sure to add a print() statement so you can see the result.

Regarding working in pairs:

- You are welcome to work with a partner on the homework assignment, but you **MUST write both your names and email addresses in each file** in a comment. Only one person needs to submit the homework on Blackboard.
- You are only allowed to work together with one other person – larger group submissions or collaborations (beyond high-level discussions of problems, as stated on the syllabus) are not allowed.

Problem 1: (3 points)

Complete the function `bestStudent`, which computes and returns the name and age of the student who had the highest score on a recent exam. Unfortunately, the data to process is not provided to the function in a particularly convenient format. The first argument is a list containing the students' names, such as `['Qi', 'Jack', 'Connor', 'Romin']`. The second argument is a list containing the students' ages and exam scores, interleaved. For instance, the list `[21, 92, 25, 95, 24, 90, 26, 98]` indicates that Qi is 21 years old and scored a 92, Jack is 25 years old and scored a 95, and so on.

Using list methods and functions (e.g., `max` and `index`), as well as slicing, determine which student had the highest score and then return that student's name and age.

Hint: remember that with slicing we can provide a third value that tells Python to select every k-th value from a list. For instance, `costs[1::2]` would select elements 1, 3, 5, 7, ... from the list called `costs`.

Examples:

Function Call	Return Values
`bestStudent(['Qi', 'Jack', 'Connor', 'Romin'], [21, 92, 25, 95, 24, 90, 26, 98])`	`('Romin', 26)`
`bestStudent(['Albert', 'Cris', 'Danny'], [22, 100, 24, 90, 23, 91])`	`('Albert', 22)`
`bestStudent(['Albert', 'Erin', 'Yang', 'Cris', 'Danny'], [25, 80, 24, 90, 27, 88, 23, 91, 24, 98])`	`('Danny', 24)`
`bestStudent(['Toni', 'Kim'], [25, 95, 27, 88])`	`('Toni', 25)`

Code to get you started is below:

```
def bestStudent(names, records):
    pass  # delete this line and start writing your code here


# Test cases
print(bestStudent(['Qi', 'Jack', 'Connor', 'Romin'], [21, 92, 25, 95, 24, 90, 26, 98]))
print(bestStudent(['Albert', 'Cris', 'Danny'], [22, 100, 24, 90, 23, 91]))
print(bestStudent(['Albert', 'Erin', 'Yang', 'Cris', 'Danny'], [25, 80, 24, 94, 27, 88, 23, 91, 24, 98]))
print(bestStudent(['Toni', 'Kim'], [25, 95, 27, 88]))
```

Problem 2 (2 points)

Complete the function `tomorrowsDate`, which returns a string containing tomorrow's date. Assume that there are 12 months in a year, and 30 days in each month. Also, assume leap years don't exist. The function's input is of the format `'MM/DD/YYYY'`, where each letter corresponds to one digit. The output will be of the same format, but leading zeroes don't matter. For example, `'01/01/2021'` is just as correct as `'1/1/2021'` or even `'01/1/2021'`.

First, separate the month, day, and year into three variables. Then, follow this logic:

```
if month is 12 and day is 30: # so we advance the month, day, and year
    next day is '1/1/(year + 1)'
elif day is 30:  # therefore the month is NOT 12, so we advance the month and day
    next day is '(month + 1)/1/(year)'
else:  #  we advance only the day
    next day is '(month)/(day + 1)/(year)'
```

Examples:

Function Call	Return Value
`tomorrowsDate('12/30/1999')`	`'1/1/2000'` or `'01/01/2000'`
`tomorrowsDate('01/30/1996')`	`'2/1/1996'` or `'02/01/1996'`
`tomorrowsDate('11/20/2021')`	`'11/21/2021'`

Code to get you started is below:

```
def tomorrowsDate(today):
    pass  # delete this line and start writing your code here


# Test cases
print(tomorrowsDate('12/30/1999'))
print(tomorrowsDate('01/30/1996'))
print(tomorrowsDate('11/20/2021'))
```

Problem 3 (3 points)

A local pizza restaurant has hired you to write a function to price the pizzas they sell every day. Small pies cost $12, medium pies cost $14, and large pies cost $16. Each topping costs $2. However, a customer can order extra toppings by prepending one or more copies of the word "Extra" in front of the string. Each "Extra" topping costs an additional $1.50. For example, `'Pepperoni'` would add $2 to the cost, `'Extra Pepperoni'` would add $3.50, `'Extra Extra Pepperoni'` would add $5, and so on.

The function takes two arguments: a string containing the size (`'Small'`, `'Medium'` or `'Large'`) and a list of strings that provide the toppings.

Examples:

Function Call	Return Value
pizza_cost('Small', ['Sausage', 'Pineapple'])	`16.0` or `16`
pizza_cost('Large', ['Onions', 'Peppers', 'Chocolate', 'Extra Extra Extra Extra Bacon', 'Mushrooms'])	`32.0` or `32`
pizza_cost('Medium', ['Olives', 'Extra Extra Extra Sausage', 'Extra Extra Cheese', 'Mushrooms'])	`29.5`
pizza_cost('Medium', ['Extra Bacon', 'Chicken', 'Pepperoni'])	`21.5`

Code to get you started is below:

```
def pizza_cost(pie_size, toppings):
    pass  # delete this line and start coding here



# Test cases
print(pizza_cost('Small', ['Sausage', 'Pineapple']))
print(pizza_cost('Large', ['Onions', 'Peppers', 'Chocolate', 'Extra Extra Extra Extra Bacon', 'Mushrooms']))
print(pizza_cost('Medium', ['Olives', 'Extra Extra Extra Sausage', 'Extra Extra Cheese', 'Mushrooms']))
print(pizza_cost('Medium', ['Extra Bacon', 'Chicken', 'Pepperoni']))
```

Problem 4 (3 points)

Complete the function `moveUp`, which takes a list of strings where each string represents a student and their "U-status" and returns the updated status of all the students. Each string is of the form `name, U#` (where # is the digit 1, 2, 3, 4). The function returns a new list containing the same students after advancing each a year. However, if a student was initially U4, they will graduate, and so the function excludes them from the new list. Otherwise, the function adds 1 to their year (i.e., U1 becomes U2; U2 becomes U3; and U3 becomes U4).

Examples:

Function Call	Return Value
`moveUp(['Moe, U2', 'Homer, U3'])`	`['Moe, U3', 'Homer, U4']`
`moveUp(['Dan, U4', 'Sydney, U3', 'Qi, U1', 'Jason, U4', 'Cassey, U2'])`	`['Sydney, U4', 'Qi, U2', 'Cassey, U3']`
`moveUp(['Kate, U1', 'Dan, U4', 'Sydney, U4', 'Susan, U2', 'Becky, U3', 'Qi, U4', 'Bob, U1])`	`['Kate, U2', 'Susan, U3', 'Becky, U4', 'Bob, U2']`
`moveUp(['Dan, U4', 'Sydney, U4', 'Qi, U4'])`	`[]`

Code to get you started is below:

```
def moveUp(students):
    pass  # delete this line and start coding here
```

```
# Test cases
print('Expected list:', "['Moe, U3', 'Homer, U4']")
print('  Actual list:', moveUp(['Moe, U2', 'Homer, U3']))
print('Expected list:', "['Sydney, U4', 'Qi, U2', 'Cassey, U3']")
print('  Actual list:', moveUp(['Dan, U4', 'Sydney, U3', 'Qi, U1', 'Jason, U4', 'Cassey, U2']))
print('Expected list:', ['Kate, U2', 'Susan, U3', 'Becky, U4', 'Bob, U2'])
print('  Actual list:', moveUp(['Kate, U1', 'Dan, U4', 'Sydney, U4', 'Susan, U2', 'Becky, U3',
'Qi, U4', 'Bob, U1']))
print('Expected list:', '[]')
print('  Actual list:', moveUp(['Dan, U4', 'Sydney, U4', 'Qi, U4']))
```

Problem 5 (3 points)

Write a function called `calculateHighAverageLow` that takes a list of lists of numbers (a 2-dimensional list) and calculates the highest number, the average, and the lowest number for each list within the list. The function should then return a new 2-dimensional list with the results of those calculations for each list in the format:

```
[[high1, average1, low1], [high2, average2, low2]]
```

You can assume every list will have at least one number. You should account for both positive and negative numbers, as seen in the 3rd example below.

Examples:

Function Call	Return Value
`calculateHighAverageLow([[10, 15], [5]])`	`[[15, 12.5, 10], [5, 5.0, 5]]`
`calculateHighAverageLow([[20,30,10,5,5], [36,25,20], [17,2,12,14]])`	`[[30, 14.0, 5], [36, 27.0, 20], [17, 11.25, 2]]`
`calculateHighAverageLow([[20,-25,-30,35,-15], [15,10,-5,25,20,17.5]])`	`[[35, -3.0, -30], [25, 13.75, -5]]`

Code to get you started is below:

```
def calculateHighAverageLow(data)
    pass  # delete this line and start coding here

# Test cases
print(calculateHighAverageLow([[10, 15], [5]]))
print(calculateHighAverageLow([[20,30,10,5,5], [36,25,20], [17,2,12,14]]))
print(calculateHighAverageLow([[20,-25,-30,35,-15], [15,10,-5,25,20,17.5]]))
```