Part 3: Large Coding Question

[P3 Q1] Quack from afar

The Problem

Unfortunately, the duck population has seen an outbreak of QUACK19, and the Duck government has placed social distancing restrictions on all pond activities. However, the Duck government is constantly changing their distancing requirements, so it is up to you to determine which ducks are allowed to swim in the pond.

You are provided with a file which contains a list of coordinates of where ducks will land in a pond. Ducks will land in the pond one after the other. The order that coordinates appear in the file corresponds to the order they will land in the pond.

When a duck attempts to land in the pond, it must first determine if the position it wants to land is far enough away from any of the current ducks in the pond. Using the `distance` that the Duck Government has given you, we can calculate whether the new duck will encroach on existing ducks in the pond. If their landing spot is too close to other ducks, they will keep flying and find another pond somewhere (we do not care about the other pond).

After all the ducks in violation of social distancing have moved on, you should write to a file the coordinates of all the ducks which are currently in the pond. The order these ducks appear in the file should preserve the order in which they arrived in the pond.

Your program should the write to a file with the same name as the `filename` parameter, although prefixed with `filtered-`.

For example, if my filename is `ducks.txt`, I would write the filtered duck coordinates to `filtered-ducks.txt`. Another example, if my filename was `pond.ducks`, I would write the filtered duck coordinates to `filtered-pond.ducks`.

The Task

Write a function `distance_ducks(filename, distance)`. The function will read a file from the given filename which contains the (x, y) coordinates of duck landing zones in a pond. The function will then filter the file so that only ducks adhering to social distancing rules can stay in the pond. The function will then write to a file containing the coordinates of ducks which are not in violation of social distancing.

If the file cannot be opened because it does not exist, you must catch the `FileNotFoundError` object and throw a new exception `FileNotFoundError` with the contents of the exception being the filename. Any other exception for failure to open a file is not caught.

If the parameter `filename` is not a string type, then a `TypeError` exception is thrown with the contents of the exception being a string `"parameter filename is not a string"`.

If the parameter `distance` is not a float or int type, then a `TypeError` exception is thrown with the contents of the exception being a string `"parameter distance is not a numeric type"`.

The file

Each line in the file will consist of two floating point values separated by a comma. Each new line of the file represents the position that a new duck will attempt to land in the pond.

An example of the file contents is shown below:

```
1.45,10.5
-3.51,0.315
1.925,-9.338
-9.283,3.190
-0.261,-10.528
3.877,-13.003
0.740,-12.094
9.066,10.707
1.099,11.568
-5.154,-6.730
1.134,-5.480
-13.035,-0.588
```

You can find sample input and output files in the `testing` folder in your workspace.

The directory names in `outputs` specify the distance value used, e.g. `1-5` indicates 1.5m of distance between ducks.

Note:

You do not have to write your own tests / test driver for this task (although you can if you'd like). The contents of the `testing` directory are sample outputs for you to check if your program is working. This is optional.

The directory structure for the `testing` folder is to let you know the distance value that was used when the sample outputs were generated, so that you can test your function with the same arguments (to hopefully get the same output). Check the README file, You can use the `diff` command to check if your output file matches the sample.

Your program does not have to mimic this directory structure. You can assume that python will output the file in the same directory as the input file (I recommend you put your input files in the current working directory).

Disclaimer: you are not guaranteed to be provided sample tests in the exam

Calculating Distance

You can use Euclidean distance to calculate the distance between any two ducks:

Consider two ducks p and q with associated coordinates x and y.

The distance between the ducks can be found with the following formula:

$$\text{distance}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Due to floating points having some precision issues, you are allowed to correct for errors less than 0.0001.

That means, `1.500000005135` would be considered as `1.5` in regards to the distance calculation.

[P3Q2] Quack Family - Siblings

The Problem

A duck can usually raise 12 ducklings, which is quite a lot. In the Kingdom of Mallard, there is a culture of recording their own family tree. Some family trees can trace back hundreds of years of lineage, while others have records of only the latest generation.

You have found some family trees online. While you were learning about their history, you had an idea to present the family tree interactively. Specifically, you would like to interactively count and identify the siblings and step-siblings of a given duck of interest.

The provided data files present the details of each duck in a separate row in the following format:

`<name>, <gender>, <year of birth>, <father>, <mother>`

Each data field is separated by commas. An example of the file contents is shown below:

```
Theo, m, 1958, ,
Ray, m, 1952, ,
Stephen, m, 1979, Theo, Mary
Daffy, m, 1976, Ray, Mary
Annabelle, F, 1975, John, Olivia
Ryan, m, 1981, Ray, Penelope
Chris, m, 1981, Theo, Mary
Deborah, F, 1996, Stephen, Annabelle
Dominic, m, 1997, Ryan, May
Brian, M, 2007, Ray, Penelope
Melvin, m, 1981, Ray, Penelope
```

The Task

Write a function `count_siblings(filename, name)`. The function will read a file from the given filename which contains the family tree of the duck of interest. It will then determine the siblings (same father and mother) and step-siblings (same father, different mother or same mother, different father) of the duck. The function will return a string in the following format:

`<name> has <number> siblings: <list of siblings>, <list of step-siblings>`

Siblings will be listed first, followed by step-siblings. Step-siblings will also be identified with the string `(step)` following their names. Regular English punctuation will need to be applied. For example, from the data of the sample file shown above:

```
print(count_siblings("family.dcf", "Brian"))
# Brian has 3 siblings: Ryan, Melvin and Daffy(step).

count_siblings("family.dcf", "Daffy")
# Daffy has 5 siblings: Stephen(step), Ryan(step), Chris(step), Brian(step) and
Melvin(step).
```

If the parameter `filename` is not a string type, then a `TypeError` exception is thrown with the contents of the exception being a string `"parameter filename is not a string"`.

If the parameter `name` is not a string type, then a `TypeError` exception is thrown with the contents of the exception being a string `"parameter name is not a string"`.

If the parameter `name` is not found in the file, then a `ValueError` exception is thrown with the contents of the exception being a string `"name parameter not found in file"`.

If the file cannot be opened because it does not exist, you must catch the `FileNotFoundError` object and throw a new exception `FileNotFoundError` with the contents of the exception being the `filename`. Any other exception for failure to open a file is not caught.

[P3 Q3] Chucky's Chicken App

Complete this program that is designed to assist in Chucky's Chicken restaurant's operation. The program should allow customers to order online, choose delivery or pickup, and pay for their food using card or cash.

1) Read in Menu From File

Write a function `read_menu()` that takes one parameter (string datatype), the filename of the menu. The function will return a list of the items and their prices. You can assume the contents of the file are valid.

Example file:

```
1 piece of chicken, 3
6 pieces of chicken, 14.95
21 pieces of chicken, 34.95
Popcorn chicken, 5.95
3 wicked wings, 3.95
5 original tenders, 9.95
6 nuggets, 5.95
Original recipe burger, 5.95
```

Example result: `[["1 piece of chicken", 3], ["6 pieces of chicken", 14.95], ["21 pieces of chicken", 34.95], ["Popcorn chicken", 5.95], ["3 wicked wings", 3.95], ["5 original tenders", 9.95], ["6 nuggets", 5.95], ["Original recipe burger", 5.95]]`

2) Order Items

Write a function `order_items()` that takes one parameter (the list of menu items), displays the menu items with the price, and continuously prompts the user to enter their choice. The function then returns the a list of lists containing the items user chose (`[[<item name>, <item price>], [<item name>, <item price>], ...]`) . Assume the user will only enter integers.

Make sure the function performs in the following order:

1. print the menu in format `<item number>) <item name>: $<item price>`
    - Item number will start at 1
    - Item price should be in 2 decimal places
2. asks the user to enter their choice or enter 0 to quit. Use the message `"Please enter the number of your choice, enter 0 to quit: "`
3. if the user enters 0, return the list of lists of items chosen. Otherwise, record the choice and go back to step 1.

3) Calculate Prices

Write a function `calculate_prices()` that takes one parameter (the list of lists of chosen items) and returns the total price of the order. You do not need to round the result.

4) Charge the User

Complete the functions `charge()`, `charge_card()` and `charge_cash()`. Assume the user will only enter integers.

Part 1:

Write a function `charge()` that takes one parameter, the total price of the order, asks the user whether they would like to pay with card or cash, and call upon `charge_card()` or `charge_cash()` depending on the user's payment method of choice.

Use the message `"Would you like to pay with card or cash? Enter 0 for card or 1 for cash: "` when prompting the user to choose their payment method.

Part 2:

Write a function `charge_card()` that takes one parameter, the total price of the order. The program will ask the user to enter the following:

1. card number (if the user does not enter a 16 digit number, spaces allowed, prompt the user to reenter the card number)
2. expiration year (the user can enter 2 digits or 4 digits year. The year has to be at least 2021)
3. expiration month (prompts the user to enter the month again if the month is invalid)
4. CVC (can be 3 or 4 digits. Prompt the user to reenter CVC if the user entered invalid value)

If the user did not enter an expiration date that is later than May 2021, print a message `'Invalid expiration date'`, and ask the user to reenter all of the information.

Part 3:

Write a function `charge_cash()` that takes one parameter, the total price of the order. The program will ask the user to enter the amount paid with message `Amount paid:`. If the user entered a value that is not divisible by 5 cents, display message `'Invalid amount'` and prompt the user to reenter the amount. If the user did not enter enough amount, display the remaining amount due `"Remaining amount: $<remaining amount>"` (in 2 decimal places) and prompt the user to reenter an amount paid. The function should display the change amount and the denominations.

Scaffold

```
def read_menu(filename):
    pass

def order_items(ls):
    pass

def calculate_prices(ls):
    pass

def charge_card(total):
    pass

def charge_cash(total):
    pass

def charge(total):
   pass
```

```
def main():
    menu = read_menu("menu.txt")
    order = order_items(menu)
    total = calculate_prices(order)
    charge(total)

if __name__ == "__main__":
    main()
```