PA5

PA 5

CSE 8A Fall 2021 PA 5

Due date: Tuesday, November 9th @ 11:59PM PST

(No Late Submission is Allowed)

Do NOT import any outside libraries. Python built-in functions that we've covered in class suffice to solve the problems. If you use any function that we haven't discussed in class yet (e.g. zip, dict, etc.), you should include inline comments to explain how it works.

Provided Files

- `hiring_process.py`

File(s) to Submit

- `hiring_process.py`

Part 1: Implementation

`hiring_process.py`

Background

In this programming assignment, you will be the hiring manager of TritonTech, a tech startup based at UCSD. TritonTech is hiring current students and new graduates to fill up their Software Engineer roles. As the hiring manager, you will be responsible for implementing functions that work together to determine if a candidate can be hired, and if so, with what base salary.

To be eligible for a software engineer role, a candidate must be interviewed, and assessed with the following five criteria: **"Coding Challenge", "Data Structure", "Algorithm", "System Design", and "Behavioral"**. A raw percentage score is associated with each criterion. Each candidate has a `score_list`, a list of 5 tuples, where each tuple contains the criteria name as well as the corresponding raw percentage score in string format.

An example of `score_list` is `[("Coding Challenge", "66%"), ("Data Structure", "98%"), ("Algorithm", "74%"), ("System Design", "82%"), ("Behavioral", "60%")]`. (These criteria may not always be listed in the above order, that is, "Coding Challenge" is not necessarily the first criteria in a candidate's `score_list`.)

Each candidate's information is stored in `candidate_info`, a list of 4 elements in the following order:

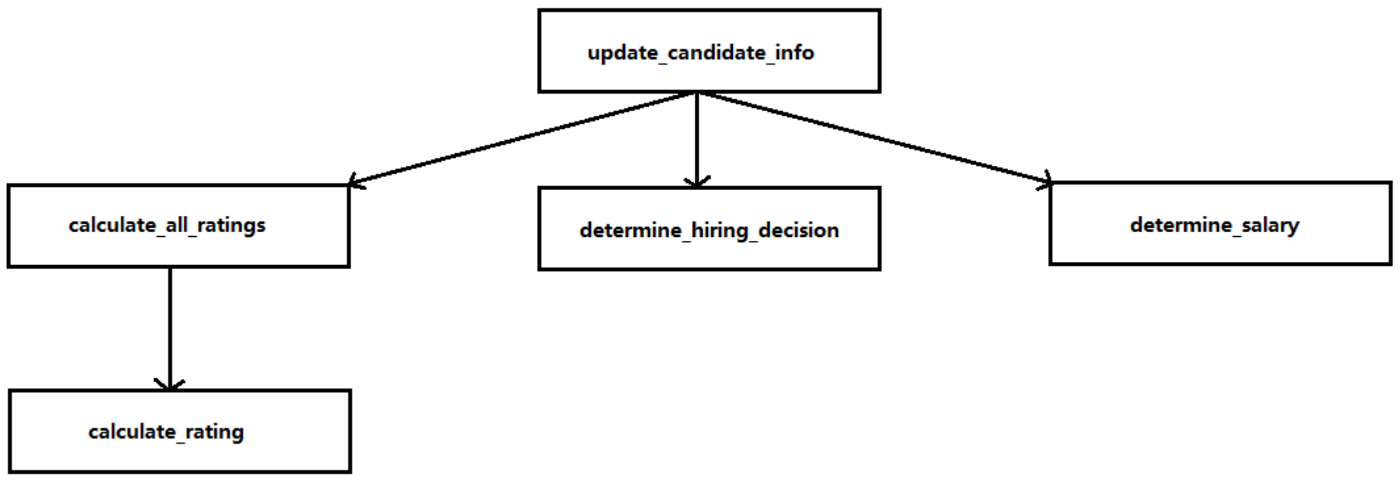
- The first element is a string that tells whether the candidate is interviewed for a full-time software engineer role or an internship, so there are only two possible values: `"Full-time"` or `"Intern"`.
- The second element is a boolean value that tells whether the candidate applied to this position with an internal referral, so a `True` value means a current employee of TritonTech has recommended this candidate.
- The third element is a string that tells the hiring decision for this candidate. This field is currently undetermined (i.e. an empty string `""`) for a new candidate. The `determine_hiring_decision` function (see below for implementation details) will update this field to `"Strong Hire"`, `"Normal Hire"`, or `"No Hire"` based on the candidate's `score_list`.
- The last element is an integer value that tells the base salary per month for this candidate. It is also currently undetermined (i.e. a zero) for a new candidate. The `determine_salary` function (see below for implementation details) will update this field to a new integer number based on the hiring decision for this candidate.

An example of `candidate_info` of a new candidate is `["Full-time", True, "", 0]`, and after you implement and run `determine_hiring_decision` and `determine_salary` (these two functions will be called in `update_candidate_info`), it will be updated to something like `["Full-time", True, "Strong Hire", 10000]`.

There are a total of 5 functions to implement in this PA.

```
def calculate_rating(raw_percentage_score)
def calculate_all_ratings(score_list, candidate_info)
def determine_hiring_decision(all_ratings, candidate_info)
def determine_salary(candidate_info)
def update_candidate_info(score_list, candidate_info)
```

Write and test each function in the order presented before moving on to the next function. Since many functions depend on previous functions, this will save you some time when debugging. Here is a dependency diagram of these functions. (An arrow from function A to function B means A is dependent on / makes a call to B.)



Starter Code

We have provided a starter file `hiring_process.py`, which contains empty function definitions that return placeholder values (such as `return 0` or `return`). For functions that have specific return values, you would need to change these placeholder values.

Please see below for implementation details.

Implementation Details

1. **`def calculate_rating(raw_percentage_score)`**

In `hiring_process.py` write the function `calculate_rating(raw_percentage_score)`. In this function we want to be able to convert any given percentage score into a string rating **"Excellent"/"Good"/"Fair"/"Bad"/"Fail"**, and return that rating. The table below outlines what scores correspond to what rating.

Notation explanation: square brackets '[ ]' *mean inclusive, and parentheses* '( )' *mean exclusive. For example, [80, 90) is equivalent to 80 <= score < 90 (left bound included, right bound excluded).*

Score (%)	Rating
[90, 100]	Excellent
[80, 90)	Good
[70, 80)	Fair
[50, 70)	Bad
Else	Fail

Note that `raw_percentage_score` will initially be given as a string, representing a float number appended with a “%”. Examples: “68.90%”, “98%”, “100%”. You can assume that `raw_percentage_score` will only represent float values in the range [0, 100].

Examples:

- `calculate_rating("68%")`. Return: `"Bad"`
- `calculate_rating("80.00%")`. Return: `"Good"`

2. **`def calculate_all_ratings(score_list, candidate_info)`**

In `hiring_process.py` write the function `calculate_all_ratings(score_list, candidate_info)`. In this function, we want to convert a list of raw percentage scores to a list of ratings. The function should return **a new list** of (criterion_name, rating) tuples based on the raw percentage score the candidate received. The conversion from raw percentages score to rating is the same as the conversion in `calculate_rating(raw_percentage_score)`.

However, a candidate with internal referral is often at an advantage. If the second element of `candidate_info` is `True`, the candidate can "skip" the coding assessment of the hiring process, which means no matter what score the candidate gets on **"Coding Challenge"**, the rating of "Coding Challenge" is always set to `"Excellent"`.

Example:

```
score_list = [("Coding Challenge", "68%"), ("Data Structure", "95%"), ("Algorithm", "76%"), ("Syste
candidate_info = ["Full-time", True, "", 0]

calculate_all_ratings(score_list, candidate_info)
```

Return: `[("Coding Challenge", "Excellent"), ("Data Structure", "Excellent"), ("Algorithm", "Fair"), ("System Design", "Good"), ("Behavioral", "Bad")]`

3. **`def determine_hiring_decision(all_ratings, candidate_info)`**

In `hiring_process.py` write the function `determine_hiring_decision(all_ratings, candidate_info)`. This function takes in a list of (criterion_name, rating) tuples **in the same format as what you output from the previous function**, and a candidate's information as arguments. The function doesn't return anything, but should simply update the hiring decision field of `candidate_info` in place based on the ratings.

To determine the hiring decision, the HR department has designed a metric called "hiring score". Each candidate's **total hiring score** is computed from the ratings that the candidate has on the five criteria, and then the hiring decision (**"Strong Hire"/"Normal Hire"/"No Hire"**) is determined based on the total hiring score.

- Each criterion gives an individual hiring score from 0 to 5 depending on the rating, so the total hiring score ranges from 0 to 25.
- The table below outlines what rating corresponds to what hiring score.

Rating	**Hiring Score**
Excellent	5
Good	4
Fair	3
Bad	1
Fail	0

- You should loop over each criterion in the rating list, and add up the corresponding individual hiring score to compute the total hiring score.
- For both full-time and internship positions, a total hiring score >= 21 means a "Strong Hire", a total hiring score from 16 to 20 (both inclusive) means a "Normal Hire", and a total hiring score <= 15 means a "No Hire" decision. See the table below.

Total Hiring Score	**Hiring Decision**
[21, 25]	Strong Hire
[16, 20]	Normal Hire
[0, 15]	No Hire

- **Special Rule**: If the candidate applies for a **full-time** position, **any "Fail" will result in an**

immediate "No Hire" decision, no matter how well the candidate performed on other parts.

After calculating the hiring score for the candidate, the function replaces the hiring decision field (currently an empty string "") of `candidate_info` with the corresponding hiring decision.

Example:

```
all_ratings = [("Coding Challenge", "Excellent"), ("Data Structure", "Excellent"), ("Algorithm", "F
candidate_info = ["Full-time", True, "", 0]
```

After calling `determine_hiring_decision` with these arguments, `candidate_info` becomes

```
["Full-time", True, "Normal Hire", 0]
```

Explanation:

The total hiring score is **18** = 5 (Excellent) + 5 (Excellent) + 3 (Fair) + 4 (Good) + 1 (Bad), and there isn't any "Fail" rating for this full-time job applicant.

4. **`def determine_salary(candidate_info)`**

In `hiring_process.py` write the function `determine_salary(candidate_info)`. This function takes in a candidate's information as the argument, and it doesn't return anything. It should simply update the salary field of `candidate_info` in place based on the hiring decision for the candidate.

- If TritonTech decides not to hire the candidate ("No Hire"), we should not even determine the salary. You can assume that a hire is the prerequisite for calling this function.
- No matter whether it is a strong hire or a normal hire, an internship position always has a **fixed** salary of $5,000 per month.
- If the candidate applies for a full-time position, a normal hire gives the candidate a monthly base salary of $9,000, and a strong hire would increase this value by $1,000.

Examples:

```
candidate_info_A = ["Full-time", True, "Normal Hire", 0]
candidate_info_B = ["Intern", False, "Strong Hire", 0]
```

After calling `determine_salary` with the above examples as the argument, they become:

```
candidate_info_A = ["Full-time", True, "Normal Hire", 9000]
candidate_info_B = ["Intern", False, "Strong Hire", 5000]
```

5. **`def update_candidate_info(score_list, candidate_info)`**

In `hiring_process.py` write the function `update_candidate_info(score_list, candidate_info)`. This function first calculates a list of ratings based on the candidate's scores on the five assessment criteria, and then determines (updates) the hiring decision for the candidate using the rating list. If TritonTech decides to hire the candidate in the end, the function will also determine (update) the base salary for the candidate. You should make calls to the functions you specified earlier to do the above tasks. The function returns nothing.

Example:

```
score_list = [("Coding Challenge", "68%"), ("Data Structure", "95%"), ("Algorithm", "76%"), ("Syste
candidate_info = ["Full-time", True, "", 0]
```

After calling `update_candidate_info` with these arguments, `candidate_info` becomes

```
["Full-time", True, "Normal Hire", 9000]
```

Part 2: Style

Coding style is an important part of ensuring readability and maintainability of your code. We will grade your coding style in all submitted code files according to the style guidelines. You can keep these guidelines in mind as you write your code or go back and fix your code at the end. **If you fix your code at the end, be sure to click the "Mark" button to submit your code. We will grade the last submission, not your workspace.** For now, we will mainly be focusing on the following:

- **Use of descriptive variable names**: The names of your variables should describe the data they hold. Your variable names should be words (or abbreviations), not single letters.
    - e.g. `a` --> `index_of_apple`; `letter1` and `letter2` --> `lower_case_letter` and `upper_case_letter`
    - Exception: If it is a loop index like `i`, `j`, `k`, one char is OK and sometimes preferred.
- **Inline Comments**: If there is a length of code that is left unexplained, take the time to type a non-redundant line summarizing this length of code (e.g. `#initialize an int` is redundant, vs. `#set initial length to 10 inches`). It will let others who look at your code understand what's going on without having to spend time understanding your logic first. But don't be too descriptive, as too many comments reduces readability.

Part 3: Conceptual Questions

You are required to complete the Conceptual Questions in PA lesson. There is no time limit but you must submit by the PA deadline. You can submit multiple times before the deadline. Your latest submission will be graded.

Submission

Turning in your code

Submit all of the following files to PA Lesson on EdStem via the “Mark" Button by **Tuesday, November 9th @ 11:59PM PST (No Late Submission is Allowed. Any late submission will not be graded.)**:

- `hiring_process.py`

Evaluation

- **Correctness** (80 points) You will earn points based on the autograder tests that your code passes. If the autograder tests are not able to run (e.g., your code does not compile or it does not match the specifications in this writeup), you may not earn credit. Your latest submission will be graded.
- **Style** (5 points)
- **Conceptual Questions** (15 points)

PA5 conceptual questions

Question 1 *Submitted Nov 9th 2021 at 2:13:40 pm*

What will be printed?

```
chars = "h;e;l;l;o;".split(";")
print(len(chars))
```

- ○ 4
- ○ 5
- [✓ 6]
- ○ 7
- ○ Error: Cannot split using a semicolon (;)

Question 2 *Submitted Nov 9th 2021 at 2:15:05 pm*

How many stack frames (from function calls) are created when the following codes are executed? **Include the global frame as well.**

```
def bar(num):
 if num > 0:
   return 1
 else:
   return -1

bar(3)
val = -3
bar(val)
```

- ○ 0
- ○ 1

- ○ 2
- [✓ 3]
- ○ 4

Question 3 *Submitted Nov 9th 2021 at 2:26:11 pm*

How many reference variables are created by the following code?

```
name = 'cse8a'
age = 42
grades = [90.0, 65.5, 100]
courses = ['cse8a', 'cse8b', 'cse11']
```

- ○ 0
- ○ 1
- ○ 2
- ○ 3
- [✓ 4]
- ○ 6
- ○ 7
- ○ 8

Question 4 *Submitted Nov 9th 2021 at 2:23:22 pm*

The following code will print out 4, 4, 4. Pick all the correct statements about this code.

```
def count():
  x = 3
  x += 1
  print(x)

count()
count()
count()
```

- [✓] x is a local variable and is re-initialized to be 3 in the stack frame
- [ ] x is a variable that is both in the global frame and the count function's stack frame.
- [ ] We can print x in the count function because x is a parameter of the function
- [ ] if we move x to be in front of the count function (as below), the code's output will stay the same

```
x = 3
def count():
  x += 1
  print(x)

count()
count()
count()
```

Question 5 *Submitted Nov 9th 2021 at 2:19:27 pm*

Please select all the statements that are true about memory models in Python.

- [✓] Python doesn't evaluate a function until it is called
- [ ] A function terminates only by executing a return statement
- [✓] A function always returns to the code using the global frame.
- [✓] A function stack frame is always created when a function is called, and is destroyed when the function returns.
- [ ] A function can only access variables in its own stack frame