Project #5 – Matchmaking

1 Objective

Implement varied selection logic in a realistic project. Create and use tuples in simple contexts.

2 Problem Overview

2.1 Intro

A friend of yours approaches you with an idea to create a new online matchmaking service, helping users find other users with whom they are compatible. Your friend wants to work on the website; they are asking you to implement the logic implementing the matchmaking algorithm.

2.2 Matching Algorithm

The algorithm is based on matching users based on athleticism, eating out, entertainment out, seriousness of relationship desired, religiosity and religion, and political zeal and leaning.

Many categories will be matched as follows[1]: if their ratings (1-10) are within 1 of each other, that categories match probability is 100%[2]; if within 3, 70%; if within 6, 40%; otherwise, 0%. For example, if person 1 has a rating of 7 in athleticism, and person 2 has a rating of 4 in that same category, the difference is 3 (7-4), so these two have a probability match of 70% in that category.

Politics and religion are special cases. For religion, if either person is rated at 8 or above on religiosity, then a match on their religions yields a 100% match in that category. If their religions don't match, then it's 50% for that category. But if neither is that religious, the normal rating scale is used. For example, if person 1 rates themselves at 8 for religiosity, and person 2 rates themselves 6, and both say they are Buddhists, then their category is matched at 100%. But if person 2 is an Atheist, the match would be 50%.

For politics, if either person is rated at 9 or above for politics (zeal), then the category is determined by the rating of the political *leaning*. If neither person is that political, then the normal rating scale is applied to political *zeal*. For example, if person 1 has a politics rating of 4 (not too political), and person 2 a rating of 5, then the normal rating applies, which is a difference of 1, so the category would match at 100%.

Each category has a weighting toward the overall match: athleticism 15%, food out 10%, entertainment out 15%, relationship seriousness 20%, religion 20%, politics 20%. Multiply the category result by the category weight to find the weighted contribution to the overall match. The overall match is the sum of the contributions of all the individual categories.

Remember that while sharing *code* is never permitted, sharing general *approaches* and broad *test cases* are fine. Compare with classmates your match results; if multiple classmates agree, that's usually a good sign that you're on track.

See the Appendix for a complete example.

[1] Referred to in future wording as "normal rating scale."

[2] Remember how percentages are represented in math: 100% is 1.0, 70% is 0.70. Use math number for calculations; you can always make it prettier for output.

3 Tuples

To simplify data passing, create tuples representing each person's ratings. These can then be passed in a single variable. Here's an example of such a tuple, along with helpful comments:

```
# athletic = a, foodOut = f, entertainmentOut = e, seriousRelationship = s
# religious = r,  religion = specific religion
# politics = p, political leaning = l
#          a   f  e  s   r   religion    p  l
person1 = (7, 10, 6, 8,  10, "buddhism", 6, 8)
```

4 Functions

Write these functions, implementing the scenarios described. Functions other than main should contain no user input and no display of information to the user. Please use the function names shown; my test code depends on exact naming.

Function	Purpose/Tasks	Parameters	Returns
main	Create tuples for two people, call match() to determine probability, output results	none	none
match	Matches two people to determine the probability of their compatibility, calling rate() as necessary and determining probabilities for religion and politics	tuples representing person1 and person2	Probability of their overall match, in range 0.0 to 1.0
rate	Contains logic for the "normal rating scale", returning the category probability	rating1, rating2	Probability of match, in range 0.0 (no chance) to 1.0 (certain)

5 Function Details

5.1 Global Variables

Create global constants for category weightings like athleticism at 15%. Follow class style for naming.

5.2 main

In main, create two person tuples. Pass them to the *match* function to determine the overall match probability, then display the results, with probability shown to the nearest tenth. For example:

```
The probability of a compatible match is 80.5%
```

6 Code Specifications

- Place all functions in the same file; there is only one .PY file to submit for this project.
- You should need no additional functions; these should be enough.
- At the bottom of the program you should have a call to main().
- Include header comments at the top of the file. Include your name, the date, and a brief description of what the program does.
- Include comments for each section saying what's going on in the lines of code below.

- Use comments elsewhere as you think they help guide the reader. Don't overdo, though! Not every line needs a comment; think about describing a block of related code.
- Use blank lines to separate sections and provide visual "breathing room."
- Use descriptive variable names.
- For formatting, use the format *function* only, not the format *method*, "f strings," etc.
- Do not use lists or other tech we haven't covered; what has been presented in class is sufficient.
- Do not use square brackets ([]) in this project. We already know how to receive multiple items from a function return; we did it in examples when we were learning about and practicing with functions.
- Functions should have only **one return** statement, the last statement in the function[3].

7 Hints

- While turning the problem description into code, remember that your first attempt will probably not achieve the **simple elegance** the problem deserves. Think carefully about finding simpler/shorter code paths without sacrificing readability or making it hard to map the problem to the code. Also remember that just because something is mentioned first in the problem description, that doesn't mean it's the first thing in your code; you can often simplify by reorganizing.
- **Code duplication** is undesirable; watch for situations where it can be eliminated or minimized.

8 Testing

Develop an appropriate number of test cases. Think about boundaries, places where behavior switches from one result to another, wrong data types (which you may not know how to fix—true—but document them and note what you need to learn for the future). Document test cases in your summary write-up.

9 Extra Credit (2+3 points)

Create a PyUnit test script called TestMatchmaker.py. In it, create automated test functions (methods) with cases for functions rating (easy, 2 points) and match (hard, 3 more points). Break test methods down by the category they are attempting to isolate and test (e.g., politics). Use the materials in the Canvas Resource module (at the top), including the Introduction to Software Testing document, and the two linked videos that demonstrate how to construct automated tests.

10 Summary

At the bottom of your program, add comments that answer these questions:

- How did you approach this assignment? Where did you get stuck, and how did you get unstuck?
- What doesn't work as you'd like, perhaps things that you'd like to fix as you learn more?
- What did you learn from this assignment? What will you do differently on the next project?

[3] This is not a hard and fast rule in the real world, but it is a good general habit. Remind me in class if you would like to have a discussion regarding recommendations for real-world use.

11 Grading Matrix

Area	Pct
main	0
tuple creation	5
display of results	5
match	0
tuple handling	5
function calls and returns	10
calculation correctness	20
general coding	5
rate	0
calculation correctness	15
general coding	5
Test cases	20
Comments, variable names, white space	5
Summary report	5
Extra credit – PyUnit tests for rating	2
Extra credit – PyUnit tests for match	3
Total	**105**

12 Appendix – Complete Example

Using these two people, the match is 69%. See calculations below.

```
# athletic = a, foodOut = f, entertainmentOut = e, seriousRelationship = s
# religious = r,  religion = specific religion
# politics = p, political leaning = l
#          a   f  e   s   r   religion       p  l
person1 = (7, 10, 8,  9, 10, "buddhism",     6, 8)
person2 = (4,  5, 6, 10,  8, "judaism",      3, 2)
```

- Athleticism: diff = 7 – 4 = 3 → 70% for category, weighting 15% → 10.5% contribution
- Food Out: diff = 10 – 5 = 5 → 40% for category, weighting 10% → 4% contribution
- Entertainment out: diff = 8 – 6 = 2 → 70% for category, weighting 15% → 10.5% contribution
- Serious relationship: diff = 10 – 9 = 1 → 100% for category, weighting 20% → 20% contribution
- Religion: at least one is 8+, but religions don't match, so 50% for category, weighting is 20% → 10% contribution
- Politics: neither is 9+ political, so we rate politics in general; diff = 6 – 3 = 3 → 70% for category, weighting is 20% → 14% contribution

Overall is sum of weighted contributions: 10.5% + 4% + 10.5% + 20% + 10% + 8% → **69%**.