[CS 11 25.1] Lab 6f – I Wanna Be The Very Best 2

Cheatsheet is available here: https://oj.dcs.upd.edu.ph/cs11cheatsheet/

Submit solution [CS 11 25.1] Lab Exercise 6

Points: 260 (partial)
Time limit: 6.0s
Memory limit: 2G

Problem type

Allowed languages
py3

Problem Statement

You have several Pokémon, each with a name and a strength value. It is guaranteed that no two Pokémon have the same strength value.

You can put your Pokémon through *matches*. In a single match, you can select two different Pokémon and have them battle against each other. The Pokémon with the higher strength value wins the match.

You forgot who your strongest Pokémon are, so you decide to find this out via matches. You don't want to tire out your Pokémon too much, so you want to do this while minimizing the number of matches you carry out.

Can you identify your **strongest** and **second strongest** Pokémon in as few matches as possible?

Task Details

Your task is to implement a function named `identify_strongest`. It should have two arguments `put_in_match` and `names`.

- `put_in_match` is a function that takes in two positional arguments, both strings (`str`s) corresponding to the names of the two Pokémon to put in a match. It will output a string denoting the name of the Pokémon that wins; i.e., the Pokémon that has the higher strength value.
- `names` is a sequence of strings denoting the names of your Pokémon.

Note that you **must** pass in valid names (i.e., names that appear in `names`) to `put_in_match`, or your submission will be judged as incorrect.

The function must return a pair (`tuple` of length 2) of strings denoting the names of the strongest and second strongest Pokémon you have, respectively.

Restrictions

- The following symbols are now allowed: `map`, `filter`
- The following imports are now allowed:
  - `count`, `islice`, `chain`, `takewhile`, `starmap and zip_longest` from `itertools`.
  - `cache`, `lru_cache`, `total_ordering`, `partial`, `reduce and wraps` from `functools`.
  - `randint`, `randrange and choice` from `random`.
  - `Fraction` from `fractions`.
  - `dataclass` from `dataclasses`.
  - `contextmanager` from `contextlib`.
  - `Enum`, `auto` from `enum`.
- Anonymous functions are now allowed.
- Inner functions are allowed.
- Classes, dataclasses, and enums are allowed.
- Recursion is allowed.
- Loops are allowed.
- Generators and comprehensions are allowed.
- Your source code must have at most $1{,}000$ bytes.

Examples

Example 1 Function Call

```
identify_strongest(put_in_match, ("Magikarp", "Pikachu", "Bulbasaur", "Ditto"))
```

Copy

Example 1 Return Value

```
("Magikarp", "Bulbasaur")
```

Copy

Example 1 Explanation

Note that the inner implementation details of the `put_in_match` function are not known to you. For testing, you may make your own definition of `put_in_match`.

Constraints

- The function `identify_strongest` will be called at most 50 times.
- There are at least 2 and at most 50 Pokémon in each call to `identify_strongest`.
- Each Pokémon name is a string of uppercase and lowercase English letters with length at most 9.
- The `put_in_match` function works properly; that is, it follows the behavior given in the **Task Details**.

Scoring

Note: New tests may be added and all submissions may be rejudged at a later time. (All future tests will satisfy the constraints.)

All subtasks require that you return the correct answer in all test cases.

- You get 50 ❤️ points if you solve all test cases where:
  - The `put_in_match` function is called at most $3{,}000$ times per test case.
- You get 25 ❤️ points if you solve all test cases where:
  - The `put_in_match` function is called at most 500 times per test case.
- You get 125 🔴 points if you solve all test cases where:
  - The `put_in_match` function is called at most 100 times per test case.
- You get 60 🔴 points if you solve all test cases where:
  - The `put_in_match` function is called at most 60 times per test case.

Clarifications

Report an issue

No clarifications have been made at this time.