Portfolio 1 Task: Psychotherapist (medium difficulty)

Overview

This task involves developing a Python function that reads and manipulates strings and lists.

One of the most famous computer programs ever written was "Eliza", developed by Prof. Joseph Weizenbaum in the mid 1960s as an experiment in human-computer interaction. Weizenbaum was one of the pioneers of natural language processing research.

Eliza engaged its user in conversation by simulating the behaviour of a non-directive psychotherapist. Weizenbaum famously became concerned by the way people treated his program as if it had real intelligence, and later became a strong critic of over-reliance on computers. In fact, Eliza was a simple program with no real intelligence at all. It just "reflected back" statements made by the user, making certain changes to their wording to give the illusion that the program understood what the user was saying.

Over the years there have been many increasingly-sophisticated implementations of Eliza-like programs. Contemporary versions are capable of carrying out lengthy conversations that cannot be distinguished from genuine human interaction. In this task you will develop a simple version of such a program, via a function that accepts a single "statement" and reflects it back as a "question" that appears to be seeking further information on the topic. For instance, given the statement

"`I think you're very attractive!`"

your function will return

"`You think I'm very attractive?`"

To do this it will replace certain words in the given string, guided by a pre-defined list of replacement words. For instance, in the example above "`I`" was replaced by "`You`" and "`you're`" was replaced by "`I'm`". Also the terminating exclamation mark was replaced by a question mark.

A problem with doing this, however, is that when the user enters a short "yes/no" style response, it's hard to say anything meaningful in return. Therefore, your function will also use a set of pre-defined statements to elicit further input from the user in this situation. For instance, if the user just types

"`No`"

your function will return the phrase

"`I'd like to hear more.`"

taken from its list of standard elicitations.

Specific steps

Enclosed with these instructions is an incomplete Python program `psycho-therapist_Q.py`. This template contains:

- Definitions of two variables, `elicitations` and `reflections`, which contain the set of standard responses and pairs of replacement words, respectively.
- A function called `talk` which you can run in IDLE's shell window to interact with the psychotherapist.
- A collection of "doctest" unit tests which your `reflect_statement` function must pass.
- A dummy "stub" of the `reflect_statement` you are required to write.

When you "run" the supplied file it will fail all of the supplied tests. Furthermore, if you call the `talk` function in the shell window, it will just repeat whatever you type because the dummy `reflect_statement` function does nothing other than return its given argument. For instance, the following is a rather unsatisfactory conversation using the dummy function. User inputs are in red.

```
>>> talk()
Hello, what seems to be your problem? I am worried.
I am worried. That's what I just said!
That's what I just said! You're just repeating me!
You're just repeating me! Goodbye.
Goodbye.
```

This uninteresting behaviour occurs because the dummy `reflect_statement` function just returns whatever it is given.

```
>>> reflect_statement("You're just repeating me!")
"You're just repeating me!"
```

Your job is to complete the `reflect_statement` function so that it produces a proper conversation, rather than just returning its input. The required extensions are as follows

1. To say something meaningful in return we need a reasonable "statement" from the user. We assume that a statement containing no blank spaces is just a single word, or perhaps no text at all. In this case the `reflect_statement` function must return one of the standard responses from the given list `elicitations`. This raises the question of which string to return. For the purposes of this task we want the response to be deterministic, to enable easy testing, so we will use the length of the user's statement as an index into the list of elicitations. For instance:

   ```
   >>> reflect_statement('ugh!')
   'Try to clarify your thoughts.'
   ```

   In this case we decided that the input was uninteresting, because it contains no spaces, and returned the string from list `elicitations` at index 4 because the length of string '`ugh!`' is 4, keeping in mind that we count from zero. (This means that if the user types nothing at all, we will return the first string in `elicitations`.)

   But what if the user's input is longer than the number of strings in `elicitations`? In this case we always return the last response in `elicitations`.

```
>>> reflect_statement('Supercalifragilisticexpialidocious')
'Pray continue.'
```

2. For "interesting" inputs, containing at least one space, we want to "reflect" the input, in effect turning a statement into a question designed to keep the user talking. The simplest part of this process is to turn a full stop (`.`) or exclamation mark (`!`) at the end into a question mark (`?`).

   ```
   >>> reflect_statement("It's yellow.")
   "It's yellow?"
   ```

   Similarly, sentences with no terminating punctuation mark should have a question mark added to the end.

   ```
   >>> reflect_statement('The car is blue')
   'The car is blue?'
   ```

   If the user's input ends with a question mark we just leave it unchanged.

3. However, since the psychotherapist usually asks questions, it is common for the user's sentences to start with "yes" or "no", and these words are not helpful to us in keeping the conversation going. Therefore, any sentence that starts with either of these words, whether capitalised or not, and possibly followed by a comma, should have that word removed from the response.

   ```
   >>> reflect_statement('No, the car is red!')
   'The car is red?'
   ```

4. Our psychotherapist is very pedantic and insists on punctuating and capitalising sentences correctly. Therefore, any punctuation in the middle of the user's input is preserved and the response must always begin with an upper case letter.

   ```
   >>> reflect_statement('when, if ever, will it?')
   'When, if ever, will it?'
   ```

   For our purposes we assume that the only punctuation marks appearing in the middle of a sentence are commas (`,`), colons (`:`) and semi-colons (`;`). We treat apostrophes (`'`) appearing within words as just part of the word itself, so they need no special handling.

5. By far the most important transformation, however, is to use the `reflections` list to replace words in the original sentence to turn it from a statement into a question. The list is effectively a table with two columns of words, forming pairs. The first word of each pair represents a word from the user's statement and the second word is its replacement in the computer's response. For instance, two pairs in `reflections` are:

`I`	`you`
`you're`	`I'm`

   Using these pairs we can replace the word on the left with the one on the right to achieve the following transformation.

```
>>> reflect_statement("I think you're very
                      attractive!")
"You think I'm very attractive?"
```

Thus your `reflect_statement` function needs to go through each word in the given statement and see if it appears on the left of one of the pairs in `reflections`. If so, it must replace it with the word on the right.

Making this slightly harder is the possibility that a word to be reflected may have a punctuation mark next to it, so you will need to find a way to separate the word from the punctuation.

```
>>> reflect_statement("Will you, could
                      you, love me?")
"Will I, could I, love you?"
```

Putting all these transformations together will allow your `reflect_statement` function to be used by the given `talk` function to conduct realistic sounding dialogues (up to a point):

```
>>> talk()
Hello, what seems to be your problem? You mustn't know.
I mustn't know? Yes, because it's so shameful.
Because it's so shameful? yes im embarrassed
You're embarrassed? Will you keep it a secret?
Will I keep it a secret? Precisely!
I'd like you to give me more detail. no i mustn't
You mustn't? It would ruin me!
It would ruin you? Yes, my family would be shocked.
Your family would be shocked? You're not helping me!
I'm not helping you? Stop!
Tell me what you're really thinking of. I told you, it's
                                        too shameful.
You told I, it's too shameful? Your grammar is poor.
My grammar is poor? I think you're just a machine!
You think I'm just a machine? Yes!!!!
You seem to be having trouble expressing your feelings.
                                Argh!
Tell me what you're really thinking of. Goodbye.
Goodbye.
```

See the unit tests in the program template file for many more examples.

Development hints

This task involves implementing several distinct transformations. For some you can use string operations but for many it will be more convenient to split the user's statement into a list of separate strings, so that you can operate on one word at a time. One of the trickier parts is dealing with words that have a punctuation mark immediately adjacent to them. A good strategy here is to devise a way to first separate the words from the punctuation, and then rejoin them when you've finished all the other steps.

Deliverables

The deliverables for this task are as follows.

1. A Python program called `psychotherapist.py` containing an extended version of the function `reflect_statement` that passes the tests in the original `psychotherapist_Q.py` file. It must contain the original set of unit tests **without modification** and must be ready to run. Your program must work under Python Version 2.7. Create your program from the supplied `psychotherapist_Q.py` template, with the '`_Q`' suffix removed from the file name. Note that when we test your program we will use additional unit tests not supplied to you, so your code must work in general, not just for the given tests.
2. A completed "statement" at the beginning of the Python file to identify your programming pair. (We can't award you marks if we don't know who you are!) Also you must indicate the percentage contribution made to the whole portfolio by both students. We do *not* expect a precise 50/50 split; an accurate and honest assessment is appreciated. This percentage will not affect your marks *except* in cases of extreme imbalances or disagreements.

Your program must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names.