159.172 Computational Thinking

Tutorial 6: Using Python Dictionaries

The Text Messaging Psychotherapist

In the early 1960's, the MIT computer scientist Joseph Weizenbaum developed a famous program called **DOCTOR** that could converse with the computer user, mimicking a non-directive style of psychotherapy. The "doctor" in this kind of therapy is essentially a good listener who responds to the patient's statements by rephrasing them, or indirectly asking for more information. In this tutorial, you will work with a (drastically) simplified version of the original **DOCTOR** program.

Go to Stream and download the file:

text_therapist.py

Set up a new project and add this file to it.

The text_therapist program performs a very simple version of *natural language processing*. If you run the program, you will be able to generate a sequence of exchanges similar to that shown here:

```
Good morning, I hope you are well today.
What can I do for you?

>>> my mother and I don't get along
Why do you say that your mother and you don't get along ?

>>> she always favours my sister
You seem to think that she always favours your sister ?

>>> my dad and I get along fine
Can you explain why your dad and you get along fine ?

>>> he helps me with my 159172 homework
Please, carry on talking.

>>> quit
Have a nice day!
```

When the user enters a statement, the program responds in one of two ways.

1. With a randomly chosen hedge, such as "Please, carry on talking."
2. By changing some key words in the user's input string and appending this string to a randomly chosen qualifier. Thus, to "my lecturer always plays favourites", the program might reply "Why do you say that your lecturer always plays favourites ?"

The program consists of a set of functions that share a common data pool. The hedges and the qualifiers are stored as tuples of strings. The other important data set is the **replacements** dictionary, which stores a mapping between first-person pronouns and second-person pronouns. For example, when the program see "I" in the patient's input it should respond with a sentence containing "you".

The **main** function displays a greeting, displays a prompt, and waits for user input. Our therapist might not be an expert, but there is no charge for his services, and he seems willing to go forever. If the patient wants to quit, she can do so by typing **quit** to end the program.

The **reply** function takes the patient's input string as an argument and returns another string as the reply. A quarter of the time, a hedge is returned. Otherwise, the function constructs a reply, by changing the pronouns in the patient's input and appending the result to a randomly selected qualifier.

The **changePerson** function performs the translation on the patient's input. This function extracts a list of words from the patient's input string. It then build a new list where any word that is present as a key in the **replacements** dictionary is replaced by its corresponding value. This list is then converted back to a string and returned. Note that the **get** method is called to get a replacement from the **replacements** dictionary. If the word is not present as a key in the dictionary, the default argument to the **get** method, the original word itself, is returned. The string method **join** glues the list of words together into a single string, with a space character as a separator.

Task 1 - basic improvements

When the patient addresses the therapist personally, the therapist's reply doesn't change the pronouns appropriately. To see an example of this problem, test the program with **"you are not a helpful therapist"**. Fix this problem by adding more items to the **replacements** dictionary.

Conversations often shift focus to earlier topics. Modify the program to support this capability. Add each patient input to a history list. Then, some small proportion of the time, choose an element from this list, apply the **changePerson** function and prepend the qualifier **"Earlier, you said that "** to this reply. Make sure that this option is triggered only after several exchanges have occurred (test the length of the history list.) You can extend the probability range to allow this option to be chosen only a small proportion of the time. You should also delete the element from the history list once it has been used in this fashion.

Task 2 - text messaging

Sadly, the swinging sixties ended a very long time ago, back when most of the current 159172 class were not yet born (and some of us were just very young.) Nowadays, language has moved on. Your main task is to get our old-fashioned therapist to *text message* his replies. This would allow our therapist to converse via a phone app, for example.

Start by adding a second dictionary to the program, **text_replacements**, that contains key-value pairs with common text abbreviations (for single words) as values and the corresponding English words as keys. For example **"you":"u"** and **"are":"r"**. You can make your new dictionary as big as you like, depending on your text messaging prowess!

Most longer English words can be easily recognized by the "average" reader without vowels included. Many people take advantage of this phenomenon when text messaging. Your next job is to define a function **removeVowels** that takes a single word as its argument, and returns a version of that word with all of the vowels removed, if the original word contains at least four characters. If the original word is shorter than four characters, then it should be returned unchanged.

Hint: make a dictionary of vowels, where each vowel key has as its value the empty string **""**. Use the **get** method on this dictionary, with an appropriate default value. You can append a single character, or another string, to a given string, using the **+** operator.

Now define a new translation function **changetoText**, that works just like the **changePerson** function, but calls the **get** method on the **text_replacements** dictionary, with appropriate default value. This function should take a string as an argument and replace all words in the string that appear as keys in the dictionary with the appropriate abbreviation, and replace all other words with their "vowels removed" version. Note that this function needs to return a new string.

In the **main** function, wherever you see a print statement, change the string argument to be the text message version of the string, using a call to **changetoText**. Now we have a text messaging psychotherapist!

Your program should start up with something like:

```
Gd mrnng, I hp tht u r wll tdy ?
Wht can I do for u ?
```

Submit your work via Stream by 12 midnight, September 20th.

Extensions

For anyone bored over the mid-semester break!

Extend the program so that the patient can "train" the therapist. Every time his reply doesn't make good sense, provide a facility for the patient to put the program into "training" mode (see how "QUIT" is handled, for example.) Then, tell the therapist what he should have said, and use this input to automatically add new key-value pairs to the **replacements** or **text_replacements** dictionaries.

Extend the dictionaries to allow phrases to be replaced, rather than just single words. For example, **"I am":"you are"** or **"see you later":"cul8r"** (not sure if that is a regular text abbreviation, but you should get the point.) This will require a rethink of how the patient's

abbreviation, but you should get the point.) This will require a rethink of how the patient's input sentence is handled. Google "Python string methods" or "Python string operations" to see the plethora of options available for processing strings in Python.