Assignment 1: Python Warm-up and Chatbot

CSE 415: Introduction to Artificial Intelligence
The University of Washington, Seattle, Autumn 2019

The reading for this assignment is *Python as a Second Language*.

Due Friday, October 4, at 23:59 PM.

Overview:
Part A of this assignment consists of several smallish Python puzzles. If you are new to Python, do the associated reading. Otherwise, this is mainly a refresher with a little problem-solving thrown in. Part B is an opportunity not only to practice your Python skills, but also to get creative and produce an agent with "character." Even more, it will offer you a chance to interact with another member of the class to create a dialog between your conversational agents. If the conversation is "good," you can get extra credit.

You should turn in four files: (1) a file named **a1.py** of function definitions, (2) a file **a1ExamplesPlus.txt** of example calls, results and answers to questions, (3) a file containing the Python code for your conversational agent (named as described in the instructions), and (4) a text file **sampleConversation.txt** containing a sample dialog between your agent and your partner's agent (see instructions for details).

For the file of examples, you can use a combination of input and output together with comment lines (beginning with a pound sign) to show that you have verified each of your answers on the computer. For each function, show a demonstration on the same example shown on this page, and two additional examples: one shorter (or smaller), and one longer (or larger) and more interesting (use your imagination).

Part A. Defining Functions (30 points).

Write Python function definitions for the following requirements (worth 5 points each, except for numbers 3 and 4, which are worth 10 points each). You should be able to infer what each function should do by a combination of reading its name and examining the relationship between its input and ouput on the given examples. Note that the functions that accept lists as arguments must be able to handle lists of any length. For this assignment your functions do not have to validate the types of their inputs. That is, you may assume that they will be called with arguments of the proper types.

Note that the name of the first function has been updated from five_x_cubed_plus_two, as of Sept. 26, to be consistent with the code that checks function signatures for you.

1. five_x_cubed_plus_2(3) -> 137

2. triple_up([2, 5, 1.5, 100, 3, 8, 7, 1, 1, 0, -2]) -> [[2, 5, 1.5], [100, 3, 8], [7, 1, 1], [0, -2]]

3. mystery_code("abc Iz th1s Secure? n0, no, 9!") -> "VWX dU OC1N nZXPMZ? I0, IJ, 9!"

   (Hint: If a character of the input is alphabetic, then it undergoes a
    change of case as well as a mapping to a different place in the alphabet.)

4 future_tense(['Yesterday', 'I', 'ate', 'pasta', 'and', 'today', 'I', 'am', 'having', 'soup']) ->
['Tomorrow', 'I', 'will', 'eat', 'pasta', 'and', 'tomorrow', 'I', 'will', 'be', 'having', 'soup']

future_tense(['Life', 'is', 'good', 'now']) ->
['Life', 'will', 'be', 'good', 'tomorrow']

Use the following rules for forming the future tense: Recognize past and present-tense forms of the verbs to be, to go, to eat, to have, and to do. Optionally handle some other verbs. Recognize the words today, yesterday, and now, and change them to tomorrow.

Note: In the future-tense function, handling of negation with the verb **do** is optional, not required. For example "We didn't eat." should map to "We will not eat.", but you don't have to handle that kind of sentence. Also, you don't have to handle the use of **do** as a helping verb, as in "We do eat meat.".

The above exercises will be graded with a combination of autograder and manual grading. Exercises 1 and 2 are worth 5 points each. Three points will be awarded for passing the instructors' autograder tests, and one point will awarded for giving each required example in your **a1examplesPlus.txt** file. Exercises 3 and 4 are worth 10 points each, with 6 of the points given for passing the autograder tests and 2 points for each required example.

A simple "sanity-check" testing program is available that might help you realize that you have a spelling error in a function name or that your functions do not return values of the proper type: a1_fn_sig_checker.py. Note that this file is not an autograder, and it does NOT test whether your functions compute the correct values.

Part B. Conversational Agents (70 points).

Create a Python program that simulates a human character in a dialog. Your program should have some definable personality, such as a widget salesman, entertainment star, political figure, etc. Besides being able to carry on a conversation with a human user, it should be able to join into conversations with the agents created by any member of the class. To do this, it will need to implement certain functions with a strict protocol.

Your solution should follow these guidelines:

- You are welcome to use rules from the Shrink3.py program, but each rule must be modified so that the response will be different.
- Incorporate a "memory" feature into your agent so that the conversation can return to some topic introduced earlier by the user. Make it work in a manner consistent with the character of your agent. (5 points).
- Incorporate a "cycle" feature into your agent so that when a given rule fires repeatedly in a session, the response is not always the same but changes in a cyclical pattern. At least 2 of your agent's rules should use this feature with at least 2 alternative responses each. (5 points)
- Make up at least one rule that uses a random-choice feature to select a response form. (5 points)
- There should be at least 15 rules in your program.
- Your program should be ready to be used as a module in another program that runs your agent with another agent in a dialog. The interface will consist of three functions that you need to write: one called **introduce()**, one called **respond(theInput)**, and one called **agentName()**.

  The **introduce** function should return a string representing a message that tells the name of the agent, what the agent represents and the name and UWNetID of you the programmer. For example, it might return a string containing:

  ```
  My name is Rusty Sales, and I sell junky cars.
  I was programmed by Jenny Chrysler. If you don't like
  the way I deal, contact her at jchrysler@u.
  How can I help you?
  ```

  The **respond** function should work almost the same way as the one in Shrink3.py. But there are two important differences. First, the function will take one argument: the input string. It should compute the wordlist and mapped_wordlist values at the beginning of the function body instead of receiving those as inputs as in Shrink3.py. Second, instead of printing out its response, your **respond** function should return it as a string. This will allow the other agent to receive it as input in the joint-dialog program.

The **agentName** function should return (as a string) a short nickname for your agent. This will be useful in printing out a prompt-like identifier when showing lines of a conversation among different agents. For example, the function might return **Rusty** for example above.

You can test out your compliance to this interface by downloading the dialog.py and minimal agent programs.

- Name your agent's Python file in the following way, so that we can keep track of the different agents: YourUWNetID_agent.py, where YourUWNetID represents your UWNetID code (i.e., your email user name within the u.washington.edu domain. For one thing, this will guarantee that each of our agents is implemented in a file with a unique name. It will also give the graders an easy way to find your agent within a group, if needed.
- Provide a comment in the code for each of your production rules.
- In addition to your Python file, turn in a text file **sampleConversation.txt** that includes at least 10 turns (and no more than 15) of the conversation for each agent, where Agent A is your agent and agent B is the agent of your partner. You can hand-edit the file with some comments -- ALL IN CAPS -- that point out places where your agent is using its MEMORY FEATURE, CYCLE FEATURE, and RANDOM-CHOICE feature. To help you create the conversation, you can use the dialog.py and Hearnone.py programs mentioned earlier.
- If there are no repeated responses in your sample conversation, and the conversation makes sense, we will award 5 points of extra credit to the author of each agent. However, if either agent repeats a response in the conversation (for example, it says "You seem quite negative" and then later says that again), then no extra credit will be awarded to either author.

Turn-In Instructions Turn in your files, named according to the instructions above, at our course's Canvas website.

Updates and Corrections

On Sept. 26, the first function to write was respelled as "five_x_cubed_plus_2". Point allocations were corrected, regarding exercises 2 and 3 on Sept. 30. If needed, additional updates and corrections will be posted here, and/or in ED.