Developers Institute

Python Course

Month 2 / Week 2 / Day 3

Exercises

Exercise 1 – Autocomplete
We're all used to Autocomplete when we fill in online forms. You start typing in a text field, and the page shows a list of words that you might want to type, eg. when you start typing on a search engine page (such as Google). We will create our own version of this for the Python console.

1. Create a new database called **word-games**.
2. Write a script to import the sowpods.txt (scrabble word list) that we used earlier on in the course, into your new database.
3. You will need to create a table for the word list. What fields will it need? What data types?
4. Use test data while building your script, until you're sure it will work properly. Then run it with the real word list file.
5. Question on efficiency: which is better – a lot of small queries, or a single big query?
6. Now create a new Python project called **autocomplete-cli** (CLI = Command-Line Interface, ie. the terminal.)
7. Create a class called Autocomplete. Create a class method named get_autocomplete_list(text), which will connect to the database, fetch a list of the relevant words, and return them as a list/tuple.
8. Think about user-friendliness. How many words should you show to the user for autocompleting?
9. Now for the user interface (UI). Until the user enters an empty string, accept more input. When the user presses <Enter>, display a list of autocompletions for the text that she has typed in.
10. 'Edge cases':
    - What if the user types in a whole word?
    - Or more than one word, eg. "hello wor"?
    - What if the user types in a word for which there are no autocompletions?
    - Make sure your program functions sensibly and gracefully in all of these cases.

BONUS: Exercise 2 – Anagrams++

1. In an earlier exercise, we created an anagram finder. Our program accepted a word from the user, and showed all words which shared the exact same letters, but in a different order.
2. Rewrite your program in the following way:
    1. Instead of reading the word list from a file, perform queries on the database we created above to retrieve words.
    2. Don't only look for words that match the original word. Also look for smaller words which, together, will be an anagram of the original word. eg. "mate" → "team", "tame", "meat", "at me", "et am".

3. Bonus: Allow the user to input multiple words/phrases/sentences, not just a single word. Find anagrams for the entire phrase together, not for each individual word.