Getting Started - A Word Game | Problem Set 4 | Contenu du cours 6.00.1x

Getting Started

1. Download and save Problem Set 4, a zip file of all the skeleton code you'll be filling in. Extract the files from the zip folder and make sure to save *all* the files - `ps4a.py`, `ps4b.py`, `test_ps4a.py` and `words.txt` - in the **same folder**. We recommend creating a folder in your Documents folder called 6001x, and inside the 6001x folder, creating a separate folder for each problem set. If you don't follow this instruction, you may end up with issues because the files for this problem set depend on one another.

2. Run the file `ps4a.py`, without making any modifications to it, in order to ensure that everything is set up correctly (this means, open the file in IDLE, and use the Run command to load the file into the interpreter). The code we have given you loads a list of valid words from a file and then calls the `playGame` function. You will implement the functions it needs in order to work. If everything is okay, after a small delay, you should see the following printed out:

   ```
   Loading word list from file...
         83667 words loaded.
   playGame not yet implemented.
   ```

   If you see an `IOError` instead (e.g., "No such file or directory"), you should change the value of the `WORDLIST_FILENAME` constant (defined near the top of the file) to the **complete pathname** for the file `words.txt` (This will vary based on where you saved the files).

   For example, if you saved all the files including this `words.txt` in the directory "C:/Users/Ana/6001x/PS4" change the line:

   `WORDLIST_FILENAME = "words.txt"` to something like

   `WORDLIST_FILENAME = "C:/Users/Ana/6001x/PS4/words.txt"`

   `Windows users, if you are copying the file path from Windows Explorer, you will have to change the backslashes to forward slashes.`

3. The file `ps4a.py` has a number of already implemented functions you can use while writing up your solution. You can ignore the code between the following comments, though you should read and understand how to use each helper function by reading the docstrings:

```
# -----------------------------------
# Helper code
# You don't need to understand this helper code,
# but you will have to know how to use the functions
# (so be sure to read the docstrings!)
    .
    .
    .
# (end of helper code)
# -----------------------------------
```

   Canopy specific instructions: Every time you modify code in `ps4a.py` go to

```
Run -> Restart Kernel (or hit the CTRL with the dot on your keyboard)
```

   before running `test_ps4a.py`. **You have to do this every time you modify the file** `ps4a.py` **and want to run the file** `test_ps4a.py`, otherwise changes to the former will not be incorporated in the latter.

4. This problem set is structured so that you will write a number of modular functions and then glue them together to form the complete word playing game. Instead of waiting until the entire game is *ready*, you should test each function you write, individually, before moving on. This approach is known as *unit testing*, and it will help you debug your code.

We have provided several test functions to get you started. After you've written each new function, unit test by running the file `test_ps4a.py` to check your work.

If your code passes the unit tests you will see a `SUCCESS` message; otherwise you will see a `FAILURE` message. These tests aren't exhaustive. You will want to test your code in other ways too.

Try running `test_ps4a.py` now (before you modify the `ps4a.py` skeleton). You should see that all the tests fail, because nothing has been implemented yet.

These are the provided test functions:

`test_getWordScore()`

Test the `getWordScore()` implementation.

`test_updateHand()`

Test the `updateHand()` implementation.

`test_isValidWord()`

Test the `isValidWord()` implementation.