**CSSE1001 Assignment 1**

**Implementing a word game in software**

For this assignment you are required to write Python code to implement a word game based on the following rules.

**Game rules**

The game involves trying to guess an unknown word. The guessing of the word is done in 11 steps.

In Step 1 of the game the player guesses a 2 letter ‘sub-word’ corresponding to the first two letters of the unknown word. They score 20 points for any of the guessed letters which are in the word (but not in the correct position) and 100 points for any letter in the correct position (See, for example, line 1 of Table 1 – the unknown word is “CASTLE”).

In Step 2 the player proposes a 3 letter sub-word to correspond to the first three letters of the unknown word, and again there are 20 points for any of the guessed letters which are in the word (but in the wrong position) and 100 points for any letter in the correct position (See, for example, line 2 of Table 1).

In Step 3 the player suggests another 3 letter sub-word to correspond to letters 2 to 4 of the unknown word, and the usual scoring system holds (See line 3 of Table 1). In Step 4 the player suggests another 3 letter sub-word to correspond to letters 3 to 5 of the unknown word (See line 4 of Table 1). In Step 5 they guess a 3 letter sub-word to correspond to letters 4 to 6 of the unknown word. (See line 5 of Table 1).

The player continues guessing sub-words according to the zig-zagging pattern shown in Table 1 until they have a final chance to guess the complete unknown word (which in the example below is “CASTLE”).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Step number | Step by step letter guesses (Unknown word is “CASTLE”) | | | | | | Score |
| 1 | A | T |  |  |  |  | 40 |
| 2 | R | A | T |  |  |  | 120 |
| 3 |  | A | L | E |  |  | 140 |
| 4 |  |  | R | I | D |  | 0 |
| 5 |  |  |  | C | A | B | 40 |
| 6 |  |  | H | U | S | K | 20 |
| 7 |  | A | N | T | S |  | 140 |
| 8 | C | A | T | S |  |  | 320 |
| 9 | C | A | S | T | E |  | 420 |
| 10 |  | S | T | A | L | E | 260 |
| 11 | C | A | S | T | L | E | 600 |

**Table 1** (Unknown word is “CASTLE”)

Code requirements

You are required to write a program to implement the kind of word game shown above. A sample output from the game is given below. In the sample printout below, the text in Italics is provided by the player, whereas the non-Italic text is printed out by the program.

Welcome to the brain teasing zig-zag word game.

What is your name? *John Smith*

Hi John! We have selected a 6 letter word for you to guess.

Let the game begin!

Now guess a 2 letters corresponding to letters 1 to 2 of the unknown word: *at*

Your guess and score were: at\_\_\_\_ : 40

Now guess 3 letters corresponding to letters 1 to 3 of the unknown word: *rat*

Your guess and score were: rat\_\_\_ : 120

Now guess 3 letters corresponding to letters 2 to 4 of the unknown word: *ale*

Your guess and score were: \_ale\_\_ : 140

Now guess 3 letters corresponding to letters 3 to 5 of the unknown word: *rid*

Your guess and score were: \_\_rid\_ : 0

Now guess 3 letters corresponding to letters 4 to 6 of the unknown word: *cab*

Your guess and score were: \_\_\_cab : 40

Now guess 4 letters corresponding to letters 3 to 6 of the unknown word: *husk*

Your guess and score were: \_\_husk : 20

Now guess 4 letters corresponding to letters 2 to 5 of the unknown word: *ants*

Your guess and score were: \_ants\_ : 140

Now guess 4 letters corresponding to letters 1 to 4 of the unknown word: *cats*

Your guess and score were: cats\_\_ : 320

Now guess 5 letters corresponding to letters 1 to 5 of the unknown word: *caste*

Your guess and score were: caste\_ : 420

Now guess 5 letters corresponding to letters 2 to 6 of the unknown word: *stale*

Your guess and score were: \_stale : 260

Now guess 6 letters corresponding to letters 1 to 6 of the unknown word: *castor*

Your guess and score were: castor : 400

You did not manage to guess the correct word. It was 'castle'. Better luck next time.

Your total score was 1900.

**Design**

You need to write an implementation of the word game using the following functions.

*select\_unknown\_word(filename)*. This function:

1. reads the words in the file, *filename*,
2. repeatedly selects a word at a random position in the file, and checks to see if it is 6 letters long, and
3. if it finds that the word is 6 letters long that word is returned.

*make\_guess\_and\_get\_score(step\_number,unknown\_word)*. This function takes in a guess of a suitable sub-word at Step, *step\_number,* and compares that guess to the corresponding letters in the unknown word. Based on this comparison a score for that guess is generated and that score is returned. If the player enters a sub-word of the wrong length, the function must print out an error message and request another sub-word to be entered.

*check\_final\_guess\_and\_compute\_overall\_score(scores, guess\_for\_unknown\_word).* This function takes in a list of scores for all the guesses and sums all these scores to produce an overall score: this score is then printed out. The function also takes in a guess for the unknown word and informs the player if they have correctly guessed the correct word. If the player has not guessed the correct word the function prints out the word.

*interact()*. This function is a high level routine which interacts with the user and prompts him/her for their name and invokes and manages the above functions.

Your program should not have any global variables, though it may have global constants. Your program should also have a main function that drives your program. Failing to achieve this will result in losing 1 or 2 marks for the programming constructs criteria.

When reading input from a user you should perform some simple checks to make sure the input is valid. Tutors will not perform esoteric input testing but if your program does not correctly catch guesses of the wrong length you will lose some marks. It is good design practice, and a good assessment strategy, to implement a program in stages. Doing this means that you are much more likely to have a working program to submit even if you cannot implement all of the requirements.

**Submission**

You must submit your completed assignment electronically through Blackboard. For information on submitting through Blackboard, please read: https://web.library.uq.edu.au/library-services/it/learnuq-blackboard-help/learnuqassessment/blackboard-assignments You should submit your assignment as a single Python file called pyword\_game.py (use this name – all lower case). You may submit your assignment multiple times before the deadline – only the last submission will be marked.

Late submission of the assignment will not be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on time, you may submit a request for an extension. See the course profile for details of how to apply for an extension: http://www.courses.uq.edu.au/student\_section\_loader.php?section=5&profileId=85405 Requests for extensions must be made no later than 48 hours prior to the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted to the ITEE Coursework Studies office (78-425) or by email to enquiries@itee.uq.edu.au. If submitted electronically, you must retain the original documentation for a minimum period of six months to provide as verification should you be requested to do so.

**Assessment and Marking Criteria**

This assignment assesses course learning objectives:

1. apply program constructs such as variables, selection, iteration and sub-routines,

3. read and analyse code written by others,

5. read and analyse a design and be able to translate the design into a working program,

6. apply techniques for testing and debugging

|  |  |
| --- | --- |
| **Criteria** | **Mark** |
| **Programming constructs**  • Program is well structured and readable  • Variable and function names are meaningful  • Functions correspond to design expectations  • Algorithmic logic is appropriate | 1  1  1  1  2 |
| **Sub-Total** | **5** |
| **Functionality**  • Implements correct functionality with no serious errors  • Implements correct functionality with small number of errors  • Implements part of the functionality with large number of errors | 3  2  1 |
| **Sub-Total** | **3** |
| **Documentation**  • Entire program is documented clearly and concisely, without excessive or extraneous comments  • Program is documented clearly, with all functions having meaningful docstring comments  • Some parts of the program have adequate comments 2 1.5 0.5 |  |
| **Sub-Total** | **2** |
| **Bonus**  • Improved on the provided design. This bonus will only be awarded if the total mark for the previous sections is at least 7.5. Changing the design, without improving it, will result in a low mark for the Programming Constructs section.  • Criteria that will be used for assessing if the design has been improved are:  o simplification of logic (i.e. something which makes the flow of the solution easier to follow); and  o improving the cohesion of functions (cohesion means that all parts of the function relate to a single purpose).  • The maximum possible mark for this assignment is 10, even if the total for the assignment and the bonus is greater than 10. | 1 |
| **Total / 10** |  |

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in the week following assignment hand-in deadline, in the practical session you have signed up to. You must attend that session in order to obtain marks for the assignment.