Programming Principles (CSI6208)

Assignment 1:	Individual programming assignment ("English Test" program)
Assignment Marks:	Marked out of 15, worth 15% of unit
Due Date:	8 April 2019, 9:00AM

Background Information

This assignment tests your understanding of and ability to apply the programming concepts we have covered in the unit so far, including the usage of variables, input and output, data types, selection, iteration, functions and data structures. Above all else, it tests *your ability to design and then implement a solution to a problem* using these concepts.

Assignment Overview

You are required to design and implement an "English Test" program in which the user is presented with a word and then asked to answer a simple question about the word. The user begins by specifying how many words they would like the test to contain. The words are selected at random from a list of words in the starter file (english_test.py) provided to you with this assignment brief. *Please use the starter file as the basis of your assignment code.*

For each of the words, the program randomly selects one of the following 4 questions to ask:

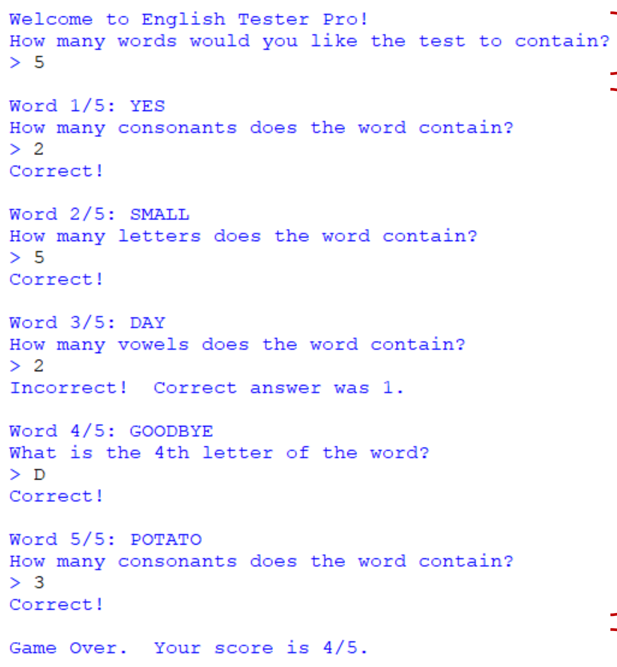
1) How many letters does the word contain?
2) How many vowels does the word contain?
3) How many consonants does the word contain?
4) What is the # letter of the word? *("#" is a number including the ordinal suffix, e.g."1st" or "3rd")*

Once the user enters their answer, the program determines if it is correct before printing an appropriate message ("Correct!" / "Incorrect!") and continuing to the next word. A correct answer gains 1 point. The program should display the user's final score at the end of the test.

The entirety of this program can be implemented in under 130 lines of code (although implementing optional additions may result in a program longer than this) – Ask your tutor for advice if you feel your program is unusually long or inefficient.

Program Output Example

To help you visualise the program, here is an annotated screenshot of the program being run:



```
Welcome to English Tester Pro!
How many words would you like the test to contain?
> 5

Word 1/5: YES
How many consonants does the word contain?
> 2
Correct!

Word 2/5: SMALL
How many letters does the word contain?
> 5
Correct!

Word 3/5: DAY
How many vowels does the word contain?
> 2
Incorrect!  Correct answer was 1.

Word 4/5: GOODBYE
What is the 4th letter of the word?
> D
Correct!

Word 5/5: POTATO
How many consonants does the word contain?
> 3
Correct!

Game Over.  Your score is 4/5.
```

The program first welcomes the user and asks them how many words they would like the test to contain – they entered 5.

The program displays the first of 5 randomly chosen words, asks a randomly chosen question about it, and awaits the user's answer.

The user enters their answer, then the program assesses it and shows an appropriate message, then continues to the next word…

Oh no, the user got question 3 wrong. As you can see, the correct answer is shown in the "Incorrect!" message.

Once the last question has been answered, the program prints a "Game Over" message including the user's score.

Pseudocode

As emphasised by the case study of Module 5, it is important to take the time to properly *design* a solution before starting to write code. Hence, this assignment requires you to *write and submit pseudocode of your program design* as well as the code for the program. Furthermore, while your tutors are happy to provide help and feedback on your assignment work throughout the semester, they will expect you to be able to show your pseudocode and explain the design of your code.

You will gain a lot more benefit from pseudocode if you actually attempt it *before* trying to code your program – even if you just start with a rough draft to establish the overall program structure, and then revise and refine it as you work on the code. This back and forth cycle of designing and coding is completely normal and expected, particularly when you are new to programming. The requirements detailed on the following pages should give you a good idea of the structure of the program, allowing you to make a start on designing your solution in pseudocode.

See Reading 3.3 for further information and tips regarding writing good pseudocode.

Write a *separate section of pseudocode for each function* you define in your program so that the pseudocode for the main part of your program is not cluttered with function definitions. Ensure that the pseudocode for each of your functions clearly describes the parameters that the function receives and what the function returns back to the program.

It may help to think of the pseudocode of your program as the content of a book, and the pseudocode of functions as its appendices: It should be possible to read and understand a book without necessarily reading the appendices, however they are there for further reference if needed.

Only one function is required in this assignment (detailed later in the assignment brief).

Program Requirements

These requirements begin where the starter file ends – after defining the "`candidateWords`" list. In the following information, numbered points describe a *core requirement* of the program, and bullet points (in italics) are *additional details, notes and hints* regarding the requirement. Ask your tutor if you do not understand the requirements or would like further information.

1. Print a welcome message, and then ask the user how many words they would like the test to contain. If the user enters a number that is less than 1 or more than the number of words in `candidateWords`, keep re-prompting them until they enter a valid number.

- *This assignment brief will assume a variable name of "`numWords`" for this value.*

2. Create a list of `numWords` randomly chosen words from `candidateWords`. These will be the words in the test. Also create an integer variable for the user's score (set it to 0).

- *This assignment brief will assume variable names of "`wordList`" and "`score`" for the variables mentioned above. Individual words in `wordList` will be referred to as "`word`".*
- *The "`random.sample()`" function can be used to randomly select a number of different items from a list and return them as a new list.*

3. Enter a loop that repeats once for each `word` in `wordList`.

- *Using a "`for`" loop and the "`enumerate()`" function is an effective way to loop through the words in `wordList` and have access to their index numbers as well. See Lecture 3 for an example of this.*

The body of this loop must…

3.1. Print the current word as well as the number of the word, e.g. if `numWords` was 5 and the current word was "HELLO", it would print "Word 1/5: HELLO" for the first word.

3.2. Generate a random number between 1 and 4, and use it to choose which question is asked.

- *Use the "`random.randint()`" function to generate a random number.*

3.3. If the random number is 1, ask the user "How many letters does the word contain?" and prompt them for input. Assess their answer, and then print an appropriate message.

- *Remember to convert the user's response to an integer so that you can assess it correctly.*
- *If the user answers correctly, remember to add 1 to the* `score` *variable. If they answer incorrectly, remember to show the correct answer as part of the "Incorrect!" message.*

3.4. If the random number is 2, ask the user "How many vowels does the word contain?" and prompt them for input. Assess their answer, and then print an appropriate message.

- *Remember to convert the user's response to an integer so that you can assess it correctly.*
- *Use the "*`countVowels`*" function (detailed below) to determine the correct answer.*
- *If the user answers correctly, remember to add 1 to the* `score` *variable. If they answer incorrectly, remember to show the correct answer as part of the "Incorrect!" message.*

3.5. If the random number is 3, ask the user "How many consonants does the word contain?" and prompt them for input. Assess their answer, and then print an appropriate message.

- *Remember to convert the user's response to an integer so that you can assess it correctly.*
- *Use the "*`countVowels`*" function here as well – all letters that are not vowels are consonants.*
- *If the user answers correctly, remember to add 1 to the* `score` *variable. If they answer incorrectly, remember to show the correct answer as part of the "Incorrect!" message.*

3.6. If the random number is 4, randomly select a number within the length of the word, (e.g. if the word is HOUSE, a randomly select a number between 1 and 5) and use it to ask the user "What is the # letter of the word?", where "#" is the selected number with the appropriate ordinal suffix, e.g. "1st", "3rd", etc. Prompt the user for input, assess their answer, and then print an appropriate message.

- *Convert the user's response to uppercase so that you can assess it regardless of case.*
- *If the user answers correctly, remember to add 1 to the* `score` *variable. If they answer incorrectly, remember to show the correct answer as part of the "Incorrect!" message.*

4. The last thing the program should do (after the body of the loop described in Requirement 3) is print a "Game Over" message that includes the user's score out of `numWords`.

The following page includes details of the "`countVowels`" function, and some optional additions and enhancements that can be added to the program.

The "countVowels" function

To assess answers to question 2 (Requirement 2.4), the program must determine the number of vowels in the word. This is also helpful when assessing answers to question 3 (Requirement 2.5), since the number of consonants in a word can be determined by deducting the number of vowels from the length of the word (hence, you do *not* need a function to count consonants).

Your program must define a function named "`countVowels`" that *receives a word* (the current `word` from `wordList`) as a parameter. It must *determine the number of vowels in the word*, and *return this number back to the program as an int*eger. There are several approaches that can be used to determine the number of vowels in a word – try to find an elegant and efficient way to do it.

The definition of the function should be at the start of the program (as it is in the starter file provided), and it should be called where needed in the program. Revise Module 4 if you are uncertain about defining and using functions, and be sure to implement it so that it receives and returns values exactly as described above. Ensure that the function does exactly what is specified above and nothing more – it is important to adhere to the stated specifications of a function.

Programming Tip: Do not attempt to implement the entire program at once. Work on one small part (a single requirement or even just *part* of a requirement) and only continue to the next part once you have made sure that the previous part is working as intended *and that you understand it*.

It can also be useful to create separate little programs to work on or test small sections of complex code, allowing you to focus on just that part without the rest of the program getting in the way.

Optional Additions and Enhancements

Below are some suggestions for minor additions and enhancements that you can make to the program to further test and demonstrate your programming ability. They are *not required* and you can earn full marks in the assignment without implementing them.

- Put everything after the creation of `candidateWords` into a loop so that the test can be run repeatedly without having to re-run the program each time. At the end of the test, ask the user whether they wish to start again and exit the loop if they don't.

- Ensure that your program does not crash if the user enters something that is not an integer when prompted for an answer to question 1, 2 or 3. Instead, your program should show an "invalid input" message and prompt the user again until they enter something valid. This is best done using exception handling (Module 6), however what you need to know is covered in Workshop 4 – create the "`inputInt`" function and use it to prompt the user for input.

- Tweak your program to ensure that each of the 4 types of questions is asked at least once in the test. Ideally, the order in which the questions appear should remain random.

Submission of Deliverables

Once your assignment is complete, submit both your **pseudocode** (in PDF format) and **source code** (".py" file) to the appropriate location in the Assessments area of Blackboard. An assignment cover sheet is not required, but be sure to **include your name and student number at the top of both files (not just in the filenames)**.

Referencing, Plagiarism and Collusion

The **entirety of your assignment must be your own work** (unless otherwise referenced) and produced for the current instance of the unit. Any use of unreferenced content you did not create constitutes plagiarism, and is deemed an act of academic misconduct. All assignments will be submitted to plagiarism checking software which includes previous copies of the assignment, and the work submitted by all other students in the unit.

Remember that this is an **individual** assignment. *Never give anyone any part of your assignment – even after the due date or after results have been released. Do not work together with other students on individual assignments – helping someone by explaining a concept or directing them to the relevant resources is fine, but doing the assignment for them or alongside them, or showing them your code is not appropriate.* An unacceptable level of cooperation between students on an assignment is collusion, and is deemed an act of academic misconduct. If you are uncertain about plagiarism, collusion or referencing, simply contact your tutor, lecturer or unit coordinator and ask.

You may be asked to **explain and demonstrate your understanding** of the work you have submitted. Your submission should accurately reflect your understanding and ability to apply the unit content.

Marking Key

Criteria	Marks
Pseudocode These marks are awarded for submitting pseudocode which suitably represents the design of your source code. Pseudocode will be assessed on the basis of whether it clearly describes the steps of the program in English, and whether the program is well structured.	**4**
Functionality These marks are awarded for submitting source code that implements the requirements specified in this brief, in Python 3. Code which is not functional or contains syntax errors will lose marks, as will failing to implement requirements as specified.	**7**
Code Quality These marks are awarded for submitting well-written source code that is efficient, well-formatted and demonstrates a solid understanding of the concepts involved. This includes appropriate use of commenting and adhering to best practise.	**4**

Total:	**15**