EMAT10007 – Introduction to Computer Programming

Exercises – Week 4, Part 1: Reviewing concepts

1. **Dictionaries.** Last week's exercise sheet contains a few exercises covering dictionaries. Please make sure you work through these after the lecture content has been presented.

   To ensure that everyone is up to speed on everything we have covered in the course so far, you should review any material you have not yet covered, and then attempt to complete the following exercises before moving onto Part 2's exercises on **functions**.

2. **Catch-up exercises.**

   (a) We use dictionaries to store pieces of data which are associated with one another. For example, a shop might store its products and their associated prices together. An example of a dictionary containing this information might:
   `Products = {"Apple" : 0.4, "Bread" : 1.8, "Milk" : 1.1, ... }`
   which is useful for lots of common operations we might need to perform on our products "list" (yes, technically it's a *dictionary* here).

   - In a new program, create a **dictionary** and fill it with lots of products that you might find in your local convenience store (or any kind of store), where the `key : value` pair should be the `name : price`. You can leave out the currency symbols for now, as we have done above.
   - Now, make your program ask the user what they would like to purchase, creating a "shopping **list**" for the customer which remembers each product and how many of each product they would like to purchase.
   - If the product is not available (not `in` the `Product` dictionary) then apologise that you do not stock the product and print the list of items you do stock (you can get just the product names by using the `keys()` function).
   - Once the customer enters "STOP", you should print out the customer's shopping list: product – price – quantity. Then, the final line of the output should be the total amount owed.

   (b) Hangman – In the game of Hangman, the user is presented with a number of blank space or, in our case, underscores: _ _ _ _ _ where each line represents a letter in a randomly selected word. The user gets 6 guesses to fill in the tiles. If the user guesses a letter correctly, then no the number of guesses remaining does not decrease. If the user guesses a letter incorrectly, then the number of guesses remaining decreases until none are left.

   (c)
   - Fill a list with multiple words of different lengths.
   - Select a word at random, and present the user with a number of underscores representing the number of letters they must fill. For example, for the word "cat", the user should be presented with:

     _ _ _
   - Ask the user to guess a letter. If it is correct, fill in the blank. For example, if the user guesses "a" for the word above, they should see:

     _ `a` _

     Multiple occurrences of the same letter should be filled with a single guess.

- If the user guesses incorrectly, tell them how many guesses are remaining, and ask to guess again.
- **Optional:** Allow the user to attempt to guess the whole word, rather than just one character at a time, should they be confident they know the word.
- **Optional:** Allow the user to select either "easy", "medium", or "hard" as a difficulty, where each difficulty corresponds to words lengths within a specific range. For example, "hard" may correspond to words at least 8 characters long. The ranges are left to you to decide.
  Note: You should create additional lists for each difficulty using list comprehensions, and the original word list as the range.

Exercises – Week 4, Part 2: Functions

1. **FUNctions.**

   - 99 bottles of beer
     - "99 Bottles of Beer" is a traditional song in the United States and Canada. It is popular to sing on long trips, as it has a very repetitive format which is easy to memorise, and can take a long time to sing. The song's simple lyrics are as follows:

       ```
       99 bottles of beer on the wall, 99 bottles of beer.
       Take one down, pass it around, 98 bottles of beer on the wall.
       ```

       The same verse is repeated, each time with one fewer bottle. The song is completed when the singer or singers reach zero.
       Your task here is to write a Python program capable of generating all the verses of the song. Define a function `PrintVerse(NumberOfBottles)` to print all of the verses of the song up to and including `NumberOfBottles`.
     - **Extensional exercise.** Write an alternate function that takes a number of total bottles e.g. `FillVerses(Total)`, but instead of printing the verses, returns a list filled with each verse of the song. Use a list comprehension to fill the list, rather than for-loops. After calling the function, you should then be able to loop through the list and print the verses *in the correct order*.
       Hint: There are several ways in which to ensure that the verses are printed in order e.g. `reversed()`, `sorted(...,reverse=...)`.
   - Wrod Scarbmler
     - A rather old internet meme once claimed that if you scrambled the words of a sentence in such a way that the first and last letters remained in place, but the rest of the letters were shuffled, then humans could still understand the sentence. It was claimed that this research was produced by the Universty of Cambridge.

       This later turned out to be incorrect, and an article written by a University of Cambridge researcher disproved this claim. However, it makes for an interesting programming assignment, so we will implement it here! The following sentence was

used to promote the original claim:

"Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe."

- Your task is to read in a sentence from the terminal, using `input()`, and then print out the scrambled text.
- You should use a function that takes in a word (or a list of letters) and returns the scrambled word. Then you should loop through the words in the sentence and call the scramble function on each word.
- **Remember**, you need to keep the first and last letters in the same positions, but the rest can be shuffled as you wish.
- Hints:
  * The sentence should be split into a list of words, and then your function should scramble one word at a time.
  * Within the `random` module, there is a function called `shuffle()`. This can be used to easily shuffle the letters in each word. However, `shuffle` rearranges the characters *in-place*, so it cannot work on strings – you will need to use a list here instead.
  * As we may not have taught how **return**ing values from a function works yet, then you may wish to have the function print each word. In order to print words onto the same line, you can modify the print function to look like the following:
    `print(word, end=" ")`
    which will print the word followed by a space, instead of a new line.