EMAT10007 – Introduction to Computer Programming

Part 1 of this week's exercise sheet is composed of the two `function` exercises from last week, as many of you continued working through the `catch-up` exercises on `Dictionaries`, `Lists`, etc.

If you didn't manage to finish those exercises, I would suggest spending this lab moving onto functions, as these are quite an important part of the first assignment, as well as next week's syntax test.

Exercises – Week 5, Part 1: Last week's exercises

In the interest of time, you can pick **one** of the two exercises, whichever you would prefer to work on. Try to complete it, asking for help if needed. If there is time at the end, feel free to come back and work on the other one.

1. 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=...)`.

2. 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.~~
    This week you will know about returning values from functions, so instead of printing each word individually, add each word to a list and then reform the message from the list.

Exercises – Week 5, Part 2: More Functions

1. **Variable scope.** It is important to understand a little bit about **scope** – an important concept in programming. The following exercises will demonstrate how variables are treated depending on whether they are declared inside (local scope) or outside (global scope) of a function.

   - Run the following code, predict the value printed to the screen.

   ```
   def F():
       print(S)

   S = "I hate spam"
   F()
   ```

   - Run the following code, predict the values printed to the screen.

```
def F():
    S = "Me too"
    print(S)

S = "I hate spam"
F()
print(S)
```

- Run the following code, predict the values printed to the screen.

```
def F():
    global S
    S = "Me too"
    print(S)

S = "I hate spam"
F()
print(S)
```

Here we tell the interpreter that the variable `S` is `global` and so a local variable `S` is not being created by Python. Instead, S refers to the *global variable* `S`, or the variable S that was created in the main body of the program, and not in the function `F()`.
Check that you understand how the scope of the variable impacts its use.

- To further illustrate the point, try the following:

```
def F():
    S = "I am a variable"
    print(S)

F()
print(S)
```

Because `S` is only created and used inside `F()`, and is not passed back to the main body of the program where `F()` is called, then `S` is not defined in the main body of the program.

2. **Variadic functions and returning values.**

Variadic functions can take any number of arguments, just like the print function, which works when used in the following way:
`print("Hello, I am", Age, "years old, born in the month of", Month, ".")`

- Write a function that take an unknown amount of numbers (integers or floats) and then:
  - Calculates the sum of the numbers
  - Calculates the product of all of the numbers

  and **returns** these values as opposed to printing them from the function.
- Write a function that takes an unknown amount of numbers as its arguments and returns the `min`, `max` and `average` of those numbers, *without* returning these as a list.

3. **Recursive functions.**

- Write a program that calculates the Fibonacci sequence using a recursive function called `Fib(n)` where `n` is the n^{th} number in the sequence.

  The following code should print:
  ```
  1  1  2  3  5  8  13  21  34  55  89  144  233  377
  ```

  ```
  for i in range(1,15):
      print(Fib(i), end=" ")
  ```

  If you are struggling, start by researching how to calculate the Fibonacci sequence. There are lots of tutorials online, so be sure to search for implementations of the Fibonacci function using *recursion*.

  Hint: You will need to identify the "special cases" of your program which prevent the function from calling itself infinitely many times.

  Read about the relative advantages and disadvantages of using *recursive* functions i.e. what happens if `n >= 100`? Is there another version of the Fibonacci function that does not use recursion? Can this be called on large values of `n`?

- Write a recursive version of the "99 Bottles of Beer" program from earlier. This is a litte tougher than the ones above, as the initial call to your `FillVerses()` function should still return a list complete with all of the verses, in the correct order.