Lab Session 03: Decomposition, Abstraction and Functions and Tuples, Lists, Aliasing, Mutability, Cloning

In this lab we will take a closer look at decomposition, abstraction and functions together with some list operations. We will try to build on the intuition gained from the previous sessions to clarify the concepts here.

Steps:

1 - Create a new file in your Spyder window and save it as lab004.py. **Note the folder where you stored the files.**

2 – Consider the following code in your file:

```
print("*" * 14)
print("*"," " * 10, "*")
print("*"," " * 10, "*")
print("*"," " * 10, "*")
print("*" * 14)
```

We have used this code to draw a square before. Write a program packaging this code as a function and later on invoking the function so that it actually runs. Recall that after defining a function, it has to be invoked for it to actually perform its task.

4 – Modify the function to accept two arguments. The first argument is the character which should be used to draw the square. The second is the length/side of the square e.g if it is 4, the square should have 4 lines. Check that the first argument is a single character and the second argument is a valid number.

5 – Write a function which accepts a person's full name and returns only their first name and last name e.g when given the name Mubarak Ibrahim Abdallah Abdallah, it returns "Mubarak Abdallah" only. Return both names as elements in a tuple.

6 – Map operations are very common in computer programs, where every element of some collection/list is operated upon in a certain way to yield some result. Using .append(), write a function that accepts a list of numbers and returns another list containing the square of the numbers.

7 – A common counterpart to map operations are reduce operations, where a source list is filtered based on some criterion. Consider Problem 2 in the previous problem set. Write a function that reduces the list of filenames to yield either the names of the cat files or the dog files. You should parameterize the function appropriately.

8 – Processing pipelines are a common way of dealing with tasks in a sequential manner. Think of an assembly line in a car plant. Design and implement a text processing pipeline program. The program should offer the following processing steps:

a. First-Letter capitalization (“flc”) e.g apple -> Apple
b. Lower-Casing (“lc”) e.g BrexIT -> brexit
c. Full Capitalization (“fc”) e.g apple -> APPLE
d. Alternate Capitalization (“ac”) e.g apple -> ApPlE
e. Stop-word filtering (“swf”) e.g the quick brown fox is lazy -> quick brown fox lazy (removes stop words: is, the, and, a,an,if, etc. Provide your own stop words)
f. One-shift encryption (“ose”): apple -> bqqmf (Hint: The ord() function may be helpful with this)

When run, the accept a single line of text, and should accept a second line with the desired list of pipeline operations separated by commas e.g “swf,ac,ose”. You should check to ensure that only valid pipeline operations are specified. For simplicity, if using One-shift encryption, always convert the string into lowercase first and then perform the encryption.

Afterwards, the program should perform the desired operations in sequence and print the result.

Good hunting