Practice Week 03

Exercise 1: Lists

Write code that prompts the user for two lists of integers of equal length, and yields, as output, the element-wise multiplication of the integers.

For example, if the user enters the lists `[1,2,3]` and `[1,2,5]`, then the output should be `1 4 15`, where 1 is the product of 1x1, 4 is the product of 2x2 and 9 is the product of 3x5.

Try to make your code bulletproof, by checking whether the lists are the same length and whether they both contain only integers.

Hint: to check whether they both contain only integers, you can use the code

```
all(isinstance(item, int) for item in my_list)
```

where `my_list` is the list you want to check.

Exercise 2: Functions for guessing game

Recall your program which generates a random number between 1-10 and asks the player to guess the number. Now modify the program to:

1. Allow two players to play (one at a time, i.e. Player 1, then Player 2).
2. For each guess calculate the distance of each player's guess from the true number, and output the mean and standard deviation of the *absolute* distances of each player. For example, if the distances are `[3,-2,-1,1]` then store this as `[3,2,1,1]` before computing the mean and standard deviation. The mean and standard deviation should be implemented as functions that can be called – however we can only call them if there are at least 3 values in each data set.
3. After the two players have played, apply a statistical test for the means (see below) to compare the means.
4. Output to the screen which user was the best at guessing. You should personalise the congratulatory and commiserations message with each player's name and appropriate punctuation for the winning player. If any player guessed correctly in less than three guesses, they automatically become the winning player. If both players guessed correctly within three guess, the result is (statistically) a tie.

We can compare two means to determine whether one is statistically larger than the other. In this exercise, code the following test as a function. *Side note:* it is called a z-test and it is implemented in some packages. However, you do it "manually" for now.

Using the notation $\bar{x_1}, \bar{x_2}, s_1, s_2$ to denote the means and standard deviations for players 1 and 2 respectively, the steps are:

1. Assume there is no difference between the two means, i.e. $\bar{x_1} - \bar{x_2} = 0$

2. Compute the following test statistic z:

$$z = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

where n_1 and n_2 are the number of times each player played.

Test z as follows:

- If $z < -1.96$, then Player 1 is the better player as $\bar{x_1} - \bar{x_2} < 0$ statistically speaking, meaning Player 1 took on average less guesses.
- If $z > 1.96$ then Player 2 is the better player as $\bar{x_1} - \bar{x_2} > 0$ statistically speaking, meaning Player 2 took on average less guesses.
- Otherwise z must be in-between -1.96 and 1.96 and statistically speaking the two players were evenly matched.

Exercise 3: Dictionary – Words count

This exercise needs the file `music_review.txt` available alongside the download location of this computer practical file.

For this exercise, you need to write a program that will create a dictionary from the words contained in `music_review.txt`, a text file that contains a music review of the new Madonna album. Each entry in the dictionary should take the form `key:value`, where `key` is each unique word in the text file and `value` is a counter, containing the number of times this word appears.

Hint: instructions on opening, reading & closing files – https://www.w3schools.com/python/python_file_open.asp

Your program should:

1. Reduce storage of useless information by not storing the words "a", "an", "I", "the" and any other such words you believe will not be useful.
2. For this exercise, treat words containing punctuation as distinct words, e.g. `album` and `album;` would constitute two separate dictionary entries. We can refine this assumption as we progress (or if you're feeling adventurous, go for it now!).
3. Output to the screen the most commonly used words in the reviews. I will leave it to you to determine appropriate output information and to set the cut-off – you should try to set a meaningful cut-off for this exercise.

Make your program as a function that takes a string and cut-off value, and output most commonly used words.

Exercise 4: Pig Latin

Write a program that:

1. Reads in the text file `lotto_luck.txt`;
2. Converts the text in the file to *Pig Latin* (create a function called `pig_latin()` and test on smaller strings at first);
3. Writes the converted text to an appropriately titled output text file. *Hint:* instructions on creating new files and writing data – https://www.w3schools.com/python/python_file_write.asp

This program requires a little more work, so you may want to discuss with your programming buddy/buddies who should be responsible for what part of the code, from code design, to writing the `pig_latin()` function through to writing the main part of the code that calls the `pig_latin()` function. For this exercise, **before you code, design your program first**.

Pig Latin is the fake version of Latin, in which the first letter of a word is removed and placed at the end of the string, and then appends "ay". The only exception is if the first letter is a vowel, in which case we keep the words as it is and append "hay" to the end.

- **p**ig → ig**p***ay*
- apple → apple*hay*

Thus the sentence: "My Name is Tim" would become: "ymay amenay ishay imtay"

- You should allow the sentences in the text file to have a mixture of case. You can convert all characters to one case (e.g. lowercase) to make it easier.
- Allow as much punctuation as you can handle programming. If you want to try basic punctuation (,;.!?) you can restrict yourself to this situation. Otherwise try to avoid all punctuation (hints on how to do this are below).
- If you want to make the program more sophisticated, words that start with letter combinations such as "th", "st", "qu", "pl", or "tr" should move both of those letters to the end, and then you can append the word with "ay" e.g.
  - **st**ay → ay**st***ay*
  - **th**ree → ree**th***ay*

Hints:

- There is a string function you can use to ensure all letters are lowercase -– look up the string library and see if you can spot it.
- You might find it easier to split the input text into a list of separate words, with each word to be processed by the `pig_latin()` function. There is a string function called `split()` that would be very useful here!
- To allow for the possibility of punctuation in a sentence that you read in, you can test whether each character of the string falls within the range defined between 'a' to 'z' or 'A' to 'Z' or if the character is a space ' '. Any other character would be considered punctuation/non-alphabetic.
- Can you use isalpha()in this case? Why/why not?
- You may wish to use a container structure to store all possible vowels and then check whether a letter is in the vowels container — this will save on processing time.