Module 12 - String Processing: Activities

In order to process strings, we often are using a pattern:

1. Create an empty list results
2. Use a for loop to go through all of the letters in the string
3. Based on the letter, add to the results list
4. When done, make the results list, which can contain single or multiple character strings, into a string using “”.join(results)

(1) Implement a function that takes a string and then returns the same string without white spaces and punctuation symbols. For example:

```
This is a long, convoluted sentence, isn't it? —>
Thisisalongconvolutedsentenceisntit
```

(2) Implement a function that replaces the vowel ‘a’ (capital or minor) with ‘1’, ‘e’ with ‘2’, ‘i’ with ‘3’, ‘o’ with ‘4’, and ‘u’ with ‘5’ in a string.

(3) Implement a function that return a Boolean, according to whether all characters in a string are vowels.

(4) Implement a function that replaces all letters with the next one in the alphabet and ‘z’ with ‘a’. We are going to learn dictionaries soon, whose use would make this exercise less tedious. We are also going to learn about ASCII code, which makes this exercise even easier. But for now, you just need to use a huge if-elif-else statement. This type of change is known as a Caesar cipher, but as cryptography, it no longer provides confidentiality protection when used against adversaries more sophisticated (or less drunk[1]) than the Gauls.

(5) Write a function that checks whether a string ends with “`.txt`”. When we are going to process directories with files in them, such functions are very handy. Avoid using the `endswith()` method for strings.

[1] Before Caesar’s invasion of Gaul, Roman merchants were selling enormous quantities of wine towards the North.