CMPUT 175 - Lab 2: Caesar Cipher

Goal: Review file I/O, basic input data validation, and defining functions that return values. Learn how to include docstrings for each function you write. Learn about another encryption method, and more about the Unicode table.

Useful functions: chr(), ord()

Look up details about how to use *chr()* and *ord()* in the Python 3 online documentation: https://docs.python.org/3/index.html.

Look up the Unicode table: https://unicode-table.com/en/

Look up examples of Python docstrings: https://www.geeksforgeeks.org/python-docstrings/

Background Information:

The Caesar cipher is a simple encryption method that works by substituting each letter in a word (or message) with a letter that is a specific number of letters ahead of it in the alphabet. That specific number is called the **cipher key**, and both the encrypter and decrypter must know the value of the cipher key in order to successfully exchange secret (encoded) messages.

Example: If the cipher key is 5, then the letter 'a' in a clear message will be encoded as the letter 'f' in the encrypted message. Using the same cipher key, the letter 'v' in a clear message will be encoded as the letter 'a' (because we wrap around to the beginning of the alphabet when we reach the end).

In order to decrypt a message encoded with a Caesar cipher, simply perform the encryption process in reverse. Example: If the cipher key is 5, then the letter 'a' in the encoded message becomes the letter 'v' in the decoded message.

For more details about the Caesar cipher, please watch the optional online video "Encryption & Public Keys Explained" on eClass under Week 2:
(https://www.youtube.com/watch?v=mEFko9nf5UU)

Problem:

In this lab, you will create a new Python program (lab2.py). In that program, you will ask the user for the name of a text file. You should check that this file ends with a ".txt" extension. If it does not, print an error message on the screen and continue to ask the user for the name of a text file until a valid one is provided. This is the only validation you need to do for the file name – you can assume that the .txt file exists in the same directory as your Python file. This should all be done in a function called *getInputFile()*, which returns the valid name of the text file as a string.

When provided with a valid text file name, you can assume that the file will contain exactly two lines. The first line will consist of a positive integer: this is the cipher key. The second line will consist of a series of encrypted words, where each word is encoded using the Caesar cipher described above. You are tasked with decrypting the message, and printing the clear message (all in lower case letters) to the screen. For example, if you have a cipher key of 1, the encrypted message ***IfmmP XpsMe*** becomes ***hello world***. This should all be done in a function called *decrypt(filename)*.

Things to keep in mind:

1. Both lines in the text file may or may not have leading and/or trailing whitespace that you should strip away before processing the data.

2. There are an unknown number of encrypted words in the text file, and these words are separated by inconsistent whitespace (e.g. tabs, single space, multiple spaces). Your decrypted message should separate each word by a single space. You can include a single space after the last word in your decrypted message.

3. You can assume that the encrypted words will only contain letters (i.e. no numbers, punctuation, or symbols), but these letters could be uppercase or lowercase. Your decrypted message should only contain lowercase letters.

4. The cipher key can be greater than 26.

5. Letters should wrap around, so that if you have a cipher key of 1, the encrypted message ***qjaab*** becomes ***pizza***.

Be sure to include a docstring at the top of each of your function definitions. In Python, docstrings are located on the line directly after `def function_name(param1, …):` and is indented one level. The docstring may be one line or many lines, but must be enclosed by triple quotes (`"""`). Each docstring should describe the purpose of the function, the function parameters, and what the function returns. Test your docstrings by calling `help(getInputFile)` and `help(decrypt)` in your main function.

Partial Sample Run 1 (does not show output from help calls):

```
Enter the input filename: secretMessage1
Invalid filename extension.  Please re-enter the input filename: secretMessage1.jpg
Invalid filename extension.  Please re-enter the input filename: secretMessage1.txt
The decrypted message is:
congratulations
```

Partial Sample Run 2 (does not show output from help calls):

```
Enter the input filename: secretMessage2.txt
The decrypted message is:
i came i saw i conquered
```

Testing:
Download the files called **secretMessage1.txt** and **secretMessage2.txt** from eClass, and save it in the same directory as your lab2.py. When you test your code using these files, your output should match what is shown in the above sample runs. However, the sample runs only test some of the functionality of your *getInputFile* and *decrypt* functions. Once you are satisfied that your program meets these minimum requirements, modify these files or create your own text files to test all 5 points under "things to keep in mind." Try to be efficient in your testing (i.e. don't test the same thing multiple times unless you're testing it in a different way), and consider any "edge" cases.

Challenge Problem (optional):
Once you've created (and tested) your *getInputFile* and *decrypt* functions, try writing an *encrypt()* function. This function will ask the user for the name of a text file – this file may or may not exist yet, but it must have a ".txt" extension. If the file already exists, the contents of it will be overwritten during the encryption; otherwise, a new file will be created. The user should be asked to enter a message that s/he would like to be encrypted – this message may consist of multiple words, but you can assume that it only consists of letters and spaces (i.e. no numbers, punctuation, or special characters). The user will then be asked to provide a cipher key. Your program should validate that the user has entered an integer number for this key. (*Hint*: lookup string methods *isalpha()*, *isalnum()*, *isdecimal()*, *isdigit()*, and *isnumeric()* – can you use any of those to help?) If it is not a whole number, an error message should be printed, and the user should be re-prompted to enter a valid integer cipher key. Finally, *encrypt()* should encode the entered message using the Caesar cipher method and the entered cipher key. The cipher key should be printed to line 1 of the text file, and the encoded message should be printed in all CAPITAL letters to line 2 of the text file.

You may want to create additional functions to support your *encrypt* function. You can also rename/modify functions that you created in the non-optional decryption part of this lab, if appropriate – just be sure to test any changes you make to ensure that your decryption still works.