CMPUT 274 - Caesar Cipher

Goal: Practice solving problems involving file I/O, basic input data validation, and defining functions that return values. Learn about a simple encryption method, and learn 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/

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 see https://www.youtube.com/watch?v=mEFko9nf5UU

Problem:

Write a Python program that asks 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. You should also check that the .txt file exists in the same directory as your Python file, printing a different error message and re-prompting if it doesn't. 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. Write a function to read in and return the cipher key (int) and cipher message (string).

Using the returned cipher key and cipher message, write a function called *decrypt(key,message)* that decrypts the message, and prints 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***.

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. This is good file processing practice.

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***.

Sample Run 1:

```
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
```

Sample Run 2:

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

Testing 1:

Download the files called **secretMessage1.txt** and **secretMessage2.txt** from eClass, and save it in the same directory as your program. 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 functions. Once you are satisfied that your program meets these minimum requirements, 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.