ITEC2270
Homework 4
100 pts
Due 10/3

This assignment will exercise

- Writing, compiling and running Python programs
- Strings, Lists, Files
- Functions

Requirements

Python programs to submit:

- Part 1: lastname_firstname_hw4p1.py

The programs must be named with your last name and first name

It is required to read the text chapters related to the assignment and run all course content examples yourself, experiment with them

Part 1

Introduction/Commentary

In week 7 we discuss functions, which are a fundamental technique of application development used to break down a larger problem into manageable, well-defined and understood pieces. This assignment will make use of functions (You have seen in the text and examples some functions written already and may have written your own. This chapters takes a direct look at defining and using them), but also your knowledge of working with strings, lists and files. You will need your understanding of the numerical conversion of characters, processing string variables, accumulator patterns, and also reading and writing files.

Cryptography is a challenging area within computer science and information technology that deals with protecting information. A quick example of where cryptography is used is how computers exchange information to keep your credit card transaction secure as you buy something electronically. In your web browser, find a small ‘lock’ symbol or look for the ‘https’ in the web address of the website you loaded to see the evidence that cryptography is being used to protect your computer’s connection with the server supplying the webpage.

Encryption, or encoding in the case of this assignment, is a simple way to demonstrate the same basic techniques that application developers use to employ cryptography. The statements you write in your script will be the same tasks you have been learning (strings, characters, loop, file access), but they are applied to transform information for security purposes

Summary

Encryption is a cryptographic process that transforms data to hide or protect it from unauthorized parties.
One simple ‘encryption’ is to use substitution by adding a constant **key** number to each input character's numeric value (each input character of the original **plaintext** readable message) to produce the **ciphertext** version of the message (this means we add the same number to all characters’ numeric value in the message. After doing this the message is ‘encrypted’).
To reverse the encryption and reveal the plaintext again,
we simply subtract the same constant number (key value) from each character in the encrypted ciphertext.

The string 'hello' becomes 'ifmmp' if the key value is 1. This means the encryption is done by adding 1 to each character's numeric value. The decryption to reverse it is done by subtracting 1.

Requirements

Write a Python program with **two** functions to do the following:

- The encrypt function accepts **1 parameter** that is the name of the plaintext file to encrypt. It does the following:
  - Define your key value. The key is the length of your first name (For example, Bill Smith would use 4 as the key)
  - Open the plaintext file for **reading**
  - Read in the contents of the plaintext file into a string (then we are done with reading)
  - Open the ciphertext file for **writing** (so that we can write the encrypted characters into the ciphertext file. Opening the file gives us an object connected to the file)
  - For each character in the plaintext string,
    - convert that character to its numeric form
    - add the key value to the numeric character to do encryption
    - convert the resulting number back to character
    - call a function on the ciphertext file object to write the encrypted character to the ciphertext file

    (We will now have a complete ciphertext file containing an encrypted message. To decrypt it, the program must call the decrypt function later)

- The decrypt function accepts **1 parameter** that is the name of a file to decrypt.
  - Define your numeric key The key is the same as the encrypt function
  - Open the ciphertext file for **reading** (the same file that was encrypted with your encrypt function)
  - Read in the full contents of the encrypted file into a string
  - For each character in the encrypted string,
    - convert that character to its numeric form
    - subtract the key value to do decryption
    - convert the resulting number back to character
    - print the decrypted character to the display (to avoid printing each character on its own line, you may want to try the ‘end’ special parameter for the print function with an empty string)

- The input plaintext file is called “plaintext”, must be in your current directory, and should contain a simple sentence. Type it up with your text editor. The ciphertext file will be called ‘ciphertext’

- Finally, in your overall program (outside of the two required functions)
  call your encrypt function (passing in the name ‘plaintext’) to encrypt the plaintext file and write the results to the ciphertext file.
  Next, call the decryption function passing in the name of the ciphertext file so that it will decrypt and print the decrypted string

- *So, your program will contain only the two required function definitions, and some statements to call the two required functions (along with any associated statements you decided to add)*

Other Requirements and Details

Sample runs

Sample run 1 (the only output is the printout from the decrypt function. The encrypt function reads in the message within the plaintext file)

```
caveat emptor, dulce bellum inexpertis
```

End of Sample Runs

Requirements

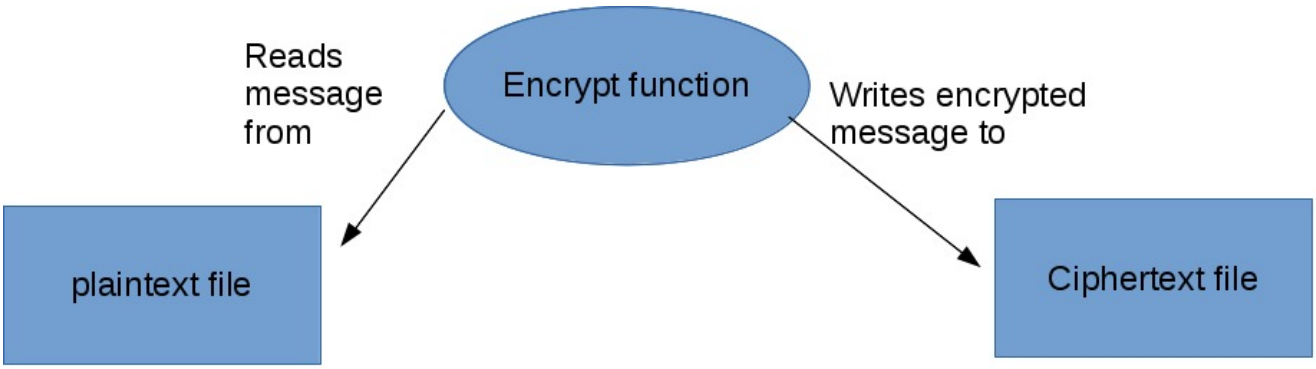
- Output must match the samples except actual values and anything excluded here
- Write your code in a readable and well-documented manner (this helps even yourself work with your code later)

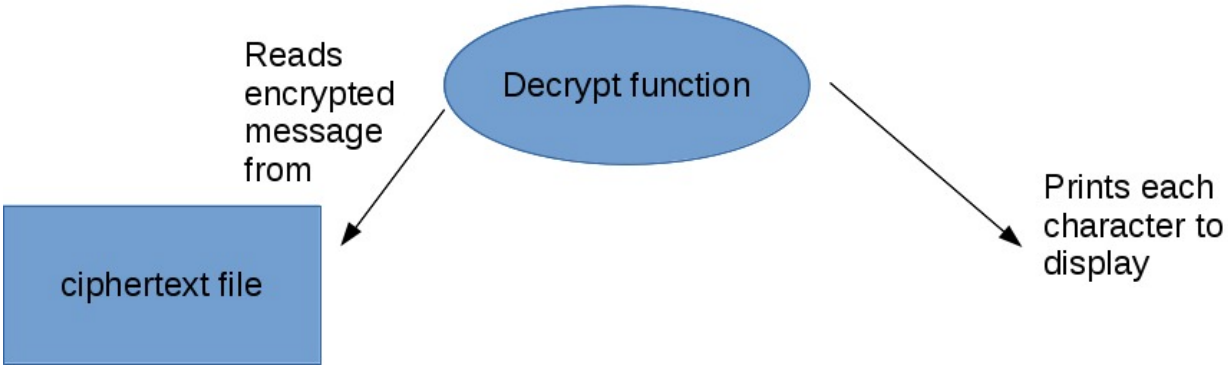
Submission

Submit in the assignments submission folder

- You are required to download or retrieve your submission after you submit it in order to check that your files are as you intend to be graded
- Resubmission is possible before deadlines
- Latest gradeable submission is graded

References





General Notes for Software Development

- We write a python script (like an actor's script, except that we are directing the computer processor what to do with the contents of memory spaces), which the python interpreter software 'reads' and executes. In the python script, we write modules, each of which is a group of statements, and statements, each of which does some particular processing of data

- Notice that in python scripts we make statements and also sometimes 'call' modules by their name to cause them to do the actions in their group of statements. Some modules/functions are defined in Python's libraries itself (like the **print** function), so we can call them without defining their statements ourselves. When we call a module/function, we must use its name and write in the data which we are sending into the function for use by the function's statements

- To write a python script we
  - open a text editor program
  - type in the statements (any kind of statements) and save the file. You must follow the rules of Python language syntax, python must 'know what everything you type is' in some way. Everything must be explicit
  - at your terminal screen (a window at which you can type commands), run the script by typing **python3 scriptname.py** where **scriptname.py** is the name of the script you wrote. Assuming no problems with your use of python, any output from the script would be printed out to the screen. Any action the script took would be taken.

- Use the documentation/tutorials and all course resources (see the syllabus) on python.org to find specifics on language features or look up functions, etc.
- Draw diagrams to represent the parts of the program you are implementing
- Until you have gained some skill and experience, expect to possibly spend significant time fixing small errors in syntax etc. and do not be discouraged. It may sometimes be necessary to restart from an earlier version of the code to recover from errors
- Continuing from the above note, when you make a significant/risky change to the code, consider making a new, differently named copy of the program to change. If the change turns out to be a bad idea, the new 'branch' is discarded. This helps to avoid losing the benefit of past work
- Write pseudocode to explain what steps are needed to solve the problem. Pseudocode is not real code (thus, you can conveniently ignore language requirements and focus on the logic), but explicit, specific English statements that detail what actions to take and what items are worked on (such as 'add 1 to the value in variable x'
- Expect to make small errors and to be spending some time developing instincts and the ability to write error-free code. Do not be surprised by making initial errors of syntax (how the code is laid out, etc), but be patient while your skill improves
- When modifying source code, make changes to the smallest region possible, then test that the change works as expected. As you become more skilled, you will be able to change more of the code at one time while still avoiding errors.

Debugging (if python prints an error about the script, we must find the problem and fix it)

- Printing
  - Printing out important values is a good way to reveal problems
  - Is the value what you expected? Is the value what it should be based on the code that affects that variable? If not, suspect a problem with the code that manipulated that variable

- Commenting
  - Temporarily commenting code statements (to force the compiler to ignore them)
  - If the problem disappears, suspect the commented code

- Disabling functions
  - When working on functions or procedures, consider disabling a function to help determine whether a problem exists with the code within the function or outside of the function.
  - This can be achieved by commenting calls to the function
  - This can also be achieved by writing a stub version of the function to call instead of the original. The stub version does nothing useful, or as little as possible and avoids affecting any important variables

- Going back to the assumptions
  - If the code does not perform as required, consider going back to question your assumptions about how the statements work (does your assumption match what the text and course materials indicate?) The previous printing and commenting techniques can help reveal when one of your assumptions has been violated. If one of your assumptions was incorrect, adjust the assumption

- Isolating the bug

- Try to reduce the size of the area of code you are testing as much as possible before debugging (commenting can help)
- This is an application of the divide and conquer strategy: dividing a difficult task into smaller, more manageable tasks Exclude/hide other areas until the current area's problems are resolved
- The goal: to isolate the source of the problem to as small a region of code as possible. This allows the programmer to blame a specific region and resolve it with confidence that no other regions are contributing to the problem

Reference material