In this project, you will be modifying the *Caesar Cipher* program you completed earlier in the semester. We will use multiple shift amounts in an attempt to make our "encryption" harder to crack and also make our program more general by reading the message from a file.

Caesar Cipher Encryption Review
The Caesar Cipher is simple way to "encrypt" **alphabetic letters** (aside: don't try encrypt anything important with the Caesar Cipher). **All** remaining punctuation symbols, numeric digits, or other characters (spaceing) remain unchanged.

Encrypting a message using the Caesar Cipher involves replacing each letter in a message with the letter *k places* further down the alphabet, wrapping around at the end of the alphabet when necessary. With *k = 0*, each letter is replaced by itself. With *k = 20*, each letter is shifted 20 places down the alphabet. To decode or decrypt the text you simply shift the encrypted letter in the opposite direction by *k* places and wrap around as necessary.

The letters in the Caesar Cipher alphabet start at 0 and continue through 25, with the letter "A" being 0 and "Z" being 25. If the user were to choose *k = 3*, the letter "A" (0) would be replaced by the letter "D" (3), while the letter "B" (1) would be replaced by the letter "E" (4). If a letter appears towards the end of the alphabet, the alphabet simply wraps around and starts again. So the letter "Z" (25) would be replaced by the letter "C" (25 + 3 = 28), which is 2 after wrapping around. The "wrap around" math is accomplished simply using the "modulo" operator in Python (%) which returns the remainder after integer division.

If *x* is the position of some letter we are trying to encrypt, it should be replaced by the letter at the position denoted by *(x + k) % 26*, which will be a number between 0 and 25. For the above example, 25 + 3 = 28, and 28 % 26 = 2, or "C". To decrypt you simply subtract *k* instead of adding: *(x - k) % 26*. To reverse the previous example we take the result "C" = 2, with a shift of 3, so (2-3) % 26 = 25 giving us back the letter "Z".

Shifting the Letters in Python
Recall (from Chapter 4) that our alphabet in Python is encoded using UTF-8 (see the table in the slides or book), from Python's perspective "A" is not 0. Instead, an upper case "A" is represented by the number 65, "B" by 66, and so on, with capital "Z" represented by 90. While a lower case "a" is represented by the number 97, "b" by 98, and so on, with "z" represented by 122. Your program will have to account for this when shifting the letters using the formula above. **Note: your program should not change the case of the original text or message.**

To obtain the UTF-8 values, you will need to use the built-in `ord(char)` function, which accepts a string containing a single letter and returns the integer value of the letter. You will also need to use the built-in `chr(int)` function which accepts an integer and returns the corresponding Unicode/ASCII letter. The methods `isupper()` and `islower()` may also be useful.
Further, any remaining punctuation symbols, numeric digits, or other characters (spaces) should be unmodified and remain unchanged in the message. Chapter 4 contains examples of how to identify and skip these characters while encoding, as a hint you will need to `import string`.

Implementation and Differences from Homework 5

Your program should begin by prompting the user for an input file (which contains the message), 5 shift amounts (all integers), and whether they would like to encrypt or decrypt, then our implementation will add some variation to the basic algorithm to make it a little harder to decode. When encrypting:

(1) We will read the text data from the file line-by-line, and all of the transformations described below be should applied line-by-line. **Before** doing the transformations below you should use `strip()` to remove the whitespace at the ends of a line.
(2) Like before, you'll replace each `"e"` character in the original message with the letters `"zw"`. If the file contains `"Hello World!"`, the resulting message will be `"Hzwllo World!"`.
(3) Also like before, after replacing the `"e"` characters, you'll add the word `"hokie"` to the beginning, middle (the middle is `length//2`), and the end of the message. Continuing with example above the message that's actually encrypted using the Caesar Cipher is `"hokieHzwllohokie World!hokie"`. This should be done **for each** line in the file.

Once you have the altered message then we will perform a slightly modified Caesar Cipher encoding as described below and print the result. Like above this transformation is applied line-by-line and shouldn't "cross" lines.

Rather than have a single shift, instead we will have 5 shift amounts entered by the user. The shifts are applied character by character. So if the user enters shift amounts: `3 4 5 6 7` and the first line of the file contains "Hello World!", your program will first transform it into `"hokieHzwllohokie World!hokie"` then:

`"h"` would be shifted by 3 characters.
`"o"` would be shifted by 4 characters.
`"k"` would be shifted by 5 characters.
`"i"` would be shifted by 6 characters.
`"e"` would be shift by 7 characters.

We then start back at the first shift amount:

`"H"` would be shifted by 3 characters.
`"z"` would be shifted by 4 characters.
`"w"` would be shifted by 5 characters.
`"l"` would be shifted by 6 characters.
`"l"` would be shift by 7 characters.

The space, exclamation point, and anything else that's not a letter isn't shifted, but it does "take up" a shift amount. So if the space character would have been shifted by 3 characters, the next character would be shifted by 4. This continues until the end of the line is reached. The whole process starts over with shift amount 3 on the next line in the file.

When decrypting, perform each step in reverse order. Decrypt the message using the Caesar Cipher above, then remove the `"hokie"` strings, then change any occurrence of `"zw"` to `"e"`. You may assume the all occurrences of `"zw"` in the result are really `"e"` characters, and further you may assume the `"hokie"` strings are in the encrypted message at the appropriate place, **however there may be other occurrences of `"hokie"` in the original message.**

Finally, your program must be able to encrypt/decrypt an arbitrary number of messages. When you've completed the encrypting/decrypting a message the user should be asked if they want to enter another message. Your program should continue until the user types "N".

While it's not a requirement for this project, I'd strongly recommend writing function(s) to help break down this project. For example, you might write a function that shifts a single letter and use in your main program.

Input Files and Sample Execution
Here's a sample input file, `quote.txt`:

```
Though Birnam Wood be come to Dunsinane,
And thou opposed, being of no woman born,
Yet I will try the last.  Before my body
I throw my warlike shield.  Lay on, Macduff;
And damned be him that first cries "Hold, enough!"
-- MacBeth
```

The user input is green:

```
Enter input file: quote.txt
Enter shift amounts: 3 4 5 6 7
Encode (E) or Decode (D)? E
Result:
kspolWltank Goyqer Drsi icltqpha ivpdb ar Iauvmsguca,nvnmj
kspolDri aksz vsttygzh, icltqphantn sk ur butdr hvur,nvnmj
kspolBdbz L boso yxf xmfd pmurlifya.  Hgzjtxgz re esieoronk
kspolL ynyra sf afxsloec vlnnvnmjfdoh.  Oed vq, Shfhzlm;ltqph
kspolDri kdqsfdg gfd lns wlfz immurliwya gwogzw "Orpi, casubjl!"oronk
kspol-- ShkspolfFecakltqph

Go again? (Y/N): Y
Enter input file: encoded-quote.txt
Enter shift amounts: 3 4 5 6 7
Encode (E) or Decode (D)? D
Result:
Though Birnam Wood be come to Dunsinane,
And thou opposed, being of no woman born,
Yet I will try the last.  Before my body
I throw my warlike shield.  Lay on, Macduff;
And damned be him that first cries "Hold, enough!"
-- MacBeth

Go again? (Y/N): N
```

What to Submit
For this assignment you should submit your **p2.py** file. Your file must be named **p2.py**.

This assignment will be graded automatically. Test your programs thoroughly before submitting them. Make sure that your programs produce correct results for every logically valid test case you can think of. Do not waste submissions on untested code, or on code that does not compile with the supplied code from the course website.

Web-CAT will assign a score based on runtime testing of your submission; your best score will be counted; the TAs will later verify that your best submission meets the stated restrictions, and assess penalties if not.

To submit this assignment:

1. Visit http://web-cat.cs.vt.edu in your web browser.
2. Enter your Virginia Tech PID and password in the appropriate fields on the log-in screen, and make sure that **Virginia Tech** is selected as the institution. Click **Login**.
3. The Web-CAT home screen will display useful announcements and assignments that are currently accepting submissions. Find the assignment that you want to submit in the table, and click the "Submit" button next to it.
4. Click the **Browse...** button and select the file you want to upload. The homework assignments and programming projects for this course should be self-contained in a single **.py** file, so you can simply select that one file.
5. Click the **Upload Submission** button. The next page will ask you to review your selection to ensure that you have chosen the right file. If everything looks correct, click **Confirm**.

The next page will show that your assignment is currently queued for grading, with an estimated wait time. This page will refresh itself automatically, and when grading is complete you will be taken to a page with your results.

When your results are ready, make sure that you have **80%** on the assignment. If you have anything less, read the hints that Web-CAT gave you and make any corrections to your code that you need to make, then submit again. Remember that for the programming projects in this class (as opposed to the homework assignments), you can submit up to 5 days after the due date, with a 10% penalty per day late.

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the submitted file:

```
#  <include a description of the purpose of this file/project/package>
#
#  @author <name and surname> (your VT PID)
#  @date   <the date>
#
# Virginia Tech Honor Code Pledge
#  On my honor:
#
#  - I have not discussed the Python language code in my program with
#    anyone other than my instructor or the teaching assistants
#    assigned to this course.
#  - I have not used Python language code obtained from another student,
#    or any other unauthorized source, either modified or unmodified.
#  - If any Python language code or documentation used in my program
#    was obtained from another source, such as a text book of course
#    notes, that has been clearly noted with a proper citation in
#    the comments of my program.
#  - I have not designed this program in such a way as to defeat or
#    interfere with the normal operation of the Web-Cat Server.
#
#  <your name>
```

Failure to include this pledge in a submission will result in the submission being disallowed during code review.