Password Encryption/Decryption Program

The following program will encrypt and decrypt passwords containing uppercase/lowercase characters, digits, and special characters. This program utilizes the following programming features:

➤ for loop ➤ nested sequences (tuples)

Example program execution is given below:

```
Program Execution ...

This program will encrypt and decrypt user passwords

Enter (e) to encrypt a password, and (d) to decrypt: e
Enter password: Pizza2Day!
Your encrypted password is: Njaam2Fmc!
>>>

Program Execution ...

This program will encrypt and decrypt user passwords

Enter (e) to encrypt a password, and (d) to decrypt: d
Enter password: Njaam2Fmc!
Your decrypted password is: Pizza2Day!
```

Task: In IDLE, open a new project and save at as Encryption_yourLastName. Copy the code from the sample on the next page. Test and revise the program, as needed.

```
1  # Password Encryption/Decryption Program
2
3  # init
4  password_out = ''
5  case_changer = ord('a') - ord('A')
6  encryption_key = (('a','m'), ('b','h'), ('c','t'), ('d','f'), ('e','g'),
7    ('f','k'), ('g','b'), ('h','p'), ('i','j'), ('j','w'), ('k','e'),('l','r'),
8    ('m','q'), ('n','s'), ('o','l'), ('p','n'), ('q','i'), ('r','u'), ('s','o'),
9    ('t','x'), ('u','z'), ('v','y'), ('w','v'), ('x','d'), ('y','c'), ('z','a'))
10
11 # program greeting
12 print('This program will encrypt and decrypt user passwords\n')
13
14 # get selection (encrypt/decrypt)
15 which = input('Enter (e) to encrypt a password, and (d) to decrypt: ')
16
17 while which != 'e' and which != 'd':
18     which = input("\nINVALID - Enter 'e' to encrypt, 'd' to decrypt: ")
19
20 encrypting = (which == 'e')  # assigns True or False
21
22 # get password
23 password_in = input('Enter password: ')
24
25 # perform encryption / decryption
26 if encrypting:
27     from_index = 0
28     to_index = 1
29 else:
30     from_index = 1
31     to_index = 0
32
33 case_changer = ord('a') - ord('A')
34
35 for ch in password_in:
36     letter_found = False
37
38     for t in encryption_key:
39         if ('a' <= ch and ch <= 'z') and ch == t[from_index]:
40             password_out = password_out + t[to_index]
41             letter_found = True
42         elif ('A' <= ch and ch <= 'Z') and chr(ord(ch) + 32) == t[from_index]:
43             password_out = password_out + chr(ord(t[to_index]) - case_changer)
44             letter_found = True
45
46     if not letter_found:
47          password_out = password_out + ch
48
49 # output
50 if encrypting:
51     print('Your encrypted password is:', password_out)
52 else:
53     print('Your decrypted password is:', password_out)
```

Notes:

Lines 4–9 perform the initialization needed for the program. Variable `password_out` is used to hold the encrypted or decrypted output of the program. Since the output string is created by appending to it each translated character one at a time, it is initialized to the empty string.

Variable `encryption_key` holds the tuple (of tuples) used to encrypt/decrypt passwords. This tuple contains as elements tuples of length two,

```
encryption_key = (('a', 'm'), ('b', 'h'), etc.
```

The first tuple, ('a', 'm'), for example, is used to encode the letter 'a'. Thus, when encrypting a given file, each occurrence of 'a' is replaced by the letter 'm'. When decrypting, the reverse is done - all occurrences of letter 'm' are replaced by the letter 'a'.

Line 12 contains the program greeting. Line 15 inputs from the user whether they wish to encrypt or decrypt a password. Based on the response, variable `encrypting` is set to either `True` or `False` (line 20).

The program section in lines 26–47 performs the encryption and decryption. If variable `encrypting` is equal to `True`, then `from_index` is set to 0 and `to_index` is set to 1, causing the "direction" of the substitution of letters to go from the first in the pair to the second ('a' re- placed by 'm'). When encrypting is `False` (and thus decryption should be performed), the direction of the substitution is from the second of the pair to the first ('m' replaced by 'a').

Variable `case_changer` (line 33) is set to the difference between the encoding of the lowercase and the uppercase letters (recall that the encoding of the lowercase letters is greater than that of the uppercase letters). The for loop at line 38 performs the iteration over the pairs of letters in the encryption key. The first time through the loop, `t = ('a', 'm')`. Thus, `t[from_index]` and `t[to_index]` refer to each of the characters in the pair. Since all characters in the encryption key are in lowercase, when uppercase letters are found in the password, they are converted to lowercase by use of variable `case_changer` (line 43) before being compared to the (lowercase) letters in the encryption key. This works because the character encoding of all lowercase letters is greater than the corresponding uppercase version,

```
>>> ord ('A')          >>> ord ('a')          >>> ord ('a') - ord ('A')
65                     97                     32
```

A similar approach is used for converting from lowercase back to uppercase. Finally, on lines 50–53, the encrypted and decrypted versions of the password are displayed to the user.

The substitution occurs in the nested for loops in lines 35–47. The outer for loop iterates variable `ch` over each character in the entered password (to be encrypted or decrypted). The first step of the outer for loop is to initialize `letter_found` to `False`. This variable is used to indicate if each character is a (uppercase or lowercase) letter. If so, it is replaced by its corresponding encoding character. If not, it must be a digit or special character, and thus appended as is (line 47). The code on lines 39–41 and lines 42–46 is similar to each other. The only difference is that since the letters in

the encryption key are all lowercase, any uppercase letters in the password need to be converted to lowercase before being compared to the letters in the key.