ICS3U Case Study: Cryptography

At its most basic level, cryptography is about encoding and decoding messages. Typically, each character of a message to be encrypted (called the *plaintext*) is combined with a character in a secret phrase (the *key*) to produce an encrypted message (the *ciphertext*). The manner in which the key and plaintext are combined determines the complexity of the encryption.

Imagine a simple encryption scheme where each letter of the alphabet is assigned a value based on its position in the alphabet (e.g. A=1, B=2, etc.). Reading down, each letter of the alphabet would have the value shown below:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
00000000011111111112222222
12345678901234567890123456
```

For this encryption scheme, the plaintext is encrypted by adding the value of each character with the value of each key character. The resulting sum will be a value between 2 (1+1) and 52 (26+26). If the sum is greater than 26, the new value has 26 subtracted from it so that it corresponds to a position in the scheme. Non-alphabetic characters, such as spaces or numbers, are not encrypted.

If the key is shorter than the plaintext, then the characters are repeated until it is the same length. If the key is longer than the plaintext, then it is truncated (cut off) instead. As a minimum security requirement, the key must consist of three or more characters, all of which are letters.

In the following example, the plaintext is SECRET MESSAGE, and the key is CODE. Since the key is shorter than the plaintext, its characters are repeated until it is the same length. The first character of the plaintext is S (19) while the first character of the key is C (3). Thus, the first encrypted character of the ciphertext is $19+3=22$, or V. The sixth character of the plaintext is T (20) while the sixth character of the key is O (15), so the sixth character of the ciphertext is $20+15=35$. Since this value exceeds 26, it is encoded as $35-26=9$, or I, instead. The entire message is encoded as follows:

```
 plaintext:  SECRET MESSAGE
       key:  CODECODECODECO
             100102 1011000
             953850 3599175
             ++++++ +++++++
             010001 0010001
             354535 5354535
             ====== =======
             220203 1032012
             207385 8843600
   35-26 = 09 ─────┘    └───── 34-26 = 08
ciphertext:  VTGWHI RHHWFJT
```

Since each encrypted letter depends on both the plaintext and the key, two identical letters may be encrypted differently. The two Es in SECRET are encoded as T and H, making it more secure.

Below is code that will encrypt a plaintext message according to the rules above.

```
# Get the plaintext (any characters, converted to uppercase)
plaintext = input("Plaintext: ").upper()
ptlen = len(plaintext)

# Get a valid key (all letters, at least 3 characters long)
key = ""
while not key.isalpha() or len(key) < 3:
    key = input("Key: ").upper()
keylen = len(key)

# Build the keytext (make it larger than needed, then trim it)
keytext = key * (ptlen // keylen + 1)
keytext = keytext[:ptlen]

# Set some values for encryption
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
ciphertext = ""

# loop through each character in the plaintext and keytext
for idx in range(ptlen):
    # character is a letter -- encrypt it
    if plaintext[idx] in alphabet:
        # add the plaintext and key values together
        # (offshift 1 for index)
        ptval = alphabet.find(plaintext[idx]) + 1
        ktval = alphabet.find(keytext[idx]) + 1
        newval = ptval + ktval
        # subtract 26 if resulting value is too large
        if newval > 26:
            newval -= 26
        # set the new character (offshift 1 for index)
        ciphertext += alphabet[newval - 1]
    # character is not a letter -- skip it
    else:
        ciphertext += plaintext[idx]

# display the encrypted message
print("The encrypted message is '{}'.".format(ciphertext))
```

The code begins by obtaining the plaintext from the user and converting it to uppercase using the `upper` string method. This cuts down on the number of characters that we need to encrypt (26 rather than 52). We note the length of the plaintext and assign it to `ptlen`.

Next, we obtain the key. We check that it consists entirely of letters using `isalpha`, and verify that at least three characters were entered using `len`. If the key is invalid, the user is prompted to enter the key again. Once a valid key is entered, we note its length and assign it to `keylen`.

To ensure that the key is repeated enough times to equal the length of the plaintext, we check how many times its length, `keylen`, will divide evenly into the length of the plaintext, `ptlen`. Integer division (quotient) handles this. For example, if the length of the plaintext was 7 and the length of the key was 3, then `7 // 3` produces a value of 2. Since $2 \times 3 = 6$, a repeated key of this length would be slightly too short. To compensate for this, we repeat the key one additional time and then trim the repeated key to be the exact length of the plaintext using slicing. A loop could have also been used to monitor the length of the repeated key, adding one character at a time until it was the right length.

In order to avoid the need to use multiple `if` statements to convert each numeric value to a letter, we will be using the index of each letter in `alphabet`. For example, the letter A has index 0, B has index 1, and so on. In general, the index is exactly one lower than the letter's value in the encoding scheme. We will need to adjust for this in our program either by subtracting or, later, adding one to our values.

Once we have set the alphabet and initialized the ciphertext as an empty string, we iterate over all characters in the plaintext. This is done by index, since we will need to reference the same position in both the plaintext and the repeated key string. As we encounter each character in the plaintext, we check if it is a letter. If so, we find the indices of both the plaintext character and the key character in `alphabet`. Recall that the indices are off-shifted by one, and so we must add one to each value. The sum of these values is assigned to `newval` and, if required, 26 is subtracted to ensure that the final value is between 1 and 26. This value is the index of the encrypted character (off-shifted by one, so we subtract one from the value), which we append to the ciphertext. If the character is not a letter, it is appended as-is, with no conversion. When all characters have been processed and the ciphertext has been built, it is displayed to the screen.

Exercises

1. Walk through the entire conversion of SECRET from the example to verify the output VTGWHI.
2. Walk through the code to predict the output for the phrase I AM SAM, using the key DAY, then check your prediction by running the program.
3. Write a similar program that will *decode* a message that has been encoded using this scheme.