Kevin Augustine 10/6/18 Comp Sci HW06

## Test debug1:

Example one:

Encoded word: WNLMY Original word: RIGHT

The reason why this example runs through the faulty program without any errors is because every single shift between the original word and encoded word is the same shift greater than zero. There isn't a shift that isn't less than zero, so every single shift is the same.

Example two:

Encoded word: ABCD Original word: WXYZ

The reason why this example runs through the faulty program without any errors is because every single shift between the original word and encoded word is the same shift less than zero. There isn't a shift that isn't greater than zero, so every single shift is the same.

The bug is that when a shift that is less than zero could be equal to a shift greater than zero, but the two shifts can't be compared to each other. The way to fix this bug is that every single time there is a shift less than zero, add 26 to the shift.

```
test_debug1.py - HW06 - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
        DEBUG ▶ Python: C ▼ 🌣 📭
                                       🍨 test_debug1.py 🗴 🦸 test_de 🗓 🕩 😤 🚦 🛟
                                        ov :param original: The original string.

■ VARIABLES

                                                  :param codeword: The encoded string.
       ▲ Locals
 Q
                                                  :return: Integer in the range 0-25 if the codeword
          codeword: 'ANALYTICAL'
                                                 represents a valid encoding of the original
                                                  string using a Caesar cipher.
                                                  Returns -1 otherwise.
         original: 'FSFQDYNHFQ'
                                                  # Use built-in ord() function to get ASCII
          shift_per_idx: 21
                                                 # integer value associated with A-Z.
 Ġ.
                                                  # have identical shift for it to be a valid Caesar shift.
                                                  shift = ord(codeword[0]) - ord(original[0])
                                                  debug = True
                                                  for idx in range(len(codeword)):
       ■ WATCH
                                                      if ord(codeword[idx]) - ord(original[idx]) != shift:
                                                          shift_per_idx = ord(codeword[idx]) - ord(original[idx])
                                                          debug = False
                                        87
                                                  return shift
                                              def caesar_fix(original, codeword):

■ CALL STACK

                         PAUSED ON STEP
                                                  Tests whether the provided codeword string
                                                  represents a valid encoding of the original
                  test_debug1.py 87:1
                                                  string using a Caesar cipher. this code is fixed
        test
                 test_debug1.py 131:1
```

## Test\_debug2:

Example one:

String1 = looped

String2 = poodle

The reason why this example runs through the faulty program without any errors is because the two strings are the same exact length and the longest consecutive matching substring is not at the end of the string

Example two:

String1 = Ipoof

String2 = fpool

The reason why this example runs through the faulty program without any errors is because the two strings are the same exact length and the longest consecutive matching substring is not at the end of the string

The bug is that when the program reaches the end of the string, it doesn't stop. The program continues running and tries to compare the next character in the string, so there is a string index out of range error. To fix this error, you need to compare whether the index plus the consecutive matching substring length is greater than or equal to the length of the string. If it is greater than or equal to the length of the string, then it should return the longest consecutive matching substring length instead of continuing to run the function.

