Mark Anthony Daño 202200145
BSIT 4-3

ROT13 Source Code:

```
alphabet = "abcdefghijklmnñopqrstuvwxyz"

text = input("Enter a text: ")
key = 13 #Fixed key value for ROT13

ciphertext = ""
i = 0
while i < len(text):
    char = text[i]
    if char.lower() in alphabet:
        pos = alphabet.index(char.lower())
        new_pos = (pos + key) % len(alphabet)
        new_char = alphabet[new_pos]
        if char.isupper():
            ciphertext += new_char.upper()
        else:
            ciphertext += new_char
    else:
        ciphertext += char
    i += 1

print("Encrypted text:", ciphertext)

dec = input("Do you want to decrypt (yes/no): ").lower()
if dec == "yes":
    decryption = ""
    i = 0
    while i < len(ciphertext):
```

```
        char = ciphertext[i]
        if char.lower() in alphabet:
            pos = alphabet.index(char.lower())
            new_pos = (pos - key) % len(alphabet)
            new_char = alphabet[new_pos]
            if char.isupper():
                decryption += new_char.upper()
            else:
                decryption += new_char
        else:
            decryption += char
        i += 1

    print("Decrypted text:", decryption)
```

Output:

```
C:\Users\maki>python try.py
Enter a text: Mark
Encrypted text: Ynew
Do you want to decrypt (yes/no): yes
Decrypted text: Mark
```

Explanation:

The first step I did was to set a variable named alphabet, which contains a string value of all the alphabet letters from a to z, including the letter ñ. This serves as the reference for finding and shifting letters during encryption and decryption.

Next, I created an input statement using input("Enter a text: ") to allow the user to type any text that they want to encrypt. After that, I set a variable named key with a fixed value of 13, since ROT13 always uses a shift of 13 positions.

Then, I initialized an empty string variable called ciphertext, which will store the encrypted result. I also set i = 0 to use as a counter in the while loop.

Inside the loop, I accessed each character of the user's text using text[i]. Then I checked if that character exists in the alphabet by using the condition if char.lower() in alphabet:. The .lower() function ensures that even if the input letter is uppercase, it will still be recognized in the lowercase alphabet.

If the character is part of the alphabet, I used alphabet.index(char.lower()) to find its position or index number. Then I added the key (13) to this index and used the modulus operator % to make sure the value wraps around if it goes past the end of the alphabet — this is written as (pos + key) % len(alphabet). The result is stored in a new variable called new_pos.

After that, I used this new position to get the shifted letter by writing new_char = alphabet[new_pos]. To preserve the original case of the letter, I added a condition: if the original character was uppercase (char.isupper()), the new character would be converted to uppercase using .upper() before adding it to ciphertext. Otherwise, it remains lowercase.

If the character is not found in the alphabet (like spaces, punctuation, or numbers), it is added to ciphertext without any change. The loop continues until all characters are processed. Finally, I printed the result using print("Encrypted text:", ciphertext) to show the encrypted version of the input.

After encryption, the program asks the user if they want to decrypt the text by using another input() function with the question "Do you want to decrypt (yes/no):". If the user types yes, the program starts the decryption process.

The decryption part works the same way as encryption but in reverse. Instead of adding the key, I subtracted it using (pos - key) % len(alphabet). This shifts the letters backward by 13 positions, returning them to their original form. Just like before, uppercase and lowercase letters are preserved.

Finally, the decrypted text is displayed using print("Decrypted text:", decryption).