PYTHON Lab_2 Assignment

Meghana D Duttargi 241059042

ME CYS, MSIS

1. Write a Python program to reverse the content of the string.

Do not use built in

```python
"""Write a Python program to reverse the content of the string.
Do not use built in """
def reverse_string(input_string):
    # Initialize an empty string to store the reversed content
    reversed_string = ""

    # Loop through the input string backwards
    for i in range(len(input_string) - 1, -1, -1):
        reversed_string += input_string[i]

    return reversed_string

# Test the function
input_string = "Hello, World!"
reversed_result = reverse_string(input_string)
print(f"Original String: {input_string}")
print(f"Reversed String: {reversed_result}")

~
~
```

```
┌──(kali㉿kali)-[~]
└─$ vi ctlab4.py

┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
Original String: Hello, World!
Reversed String: !dlroW ,olleH

┌──(kali㉿kali)-[~]
└─$
```

2. Create a program that performs basic string compression using the counts of repeated characters. For example, the string “aabccccaaa” would become “a2b1c5a3”.

```
┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
a2b1c5a3

┌──(kali㉿kali)-[~]
└─$
```

```python
#2)Create a program that performs basic string compression using the counts
5a3

def compress_string(s):
    compressed = ""
    count = 1
    for i in range(1, len(s)):
        if s[i] == s[i - 1]:
            count += 1
        else:
            compressed += s[i - 1] + str(count)
            count = 1
    compressed += s[-1] + str(count)
    return compressed

input_string = "aabcccccaaa"
result = compress_string(input_string)
print(result)
```

3. Get the Caesar cipher from the user Decrypt the cipher

```python
#3. Get the Caesar cipher from the user Decrypt the cipher

def decrypt_caesar_cipher(cipher_text, shift):
    decrypted_text = ""

    for char in cipher_text:
        if char.isalpha():
            shift_amount = shift % 26
            if char.islower():
                decrypted_text += chr((ord(char) - shift_amount - 97) % 26 + 97)
            elif char.isupper():
                decrypted_text += chr((ord(char) - shift_amount - 65) % 26 + 65)
        else:
            decrypted_text += char

    return decrypted_text

cipher_text = input("Enter the Caesar cipher text: ")
shift = int(input("Enter the shift value used for encryption: "))

decrypted_message = decrypt_caesar_cipher(cipher_text, shift)
print("Decrypted message:", decrypted_message)

~
```

```
┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
Enter the Caesar cipher text: khoor
Enter the shift value used for encryption: 3
Decrypted message: hello
```

4. Get the cipher encrypted using shift cipher. Identify the key used to encrypt using brute force

ie all the values in the key space

```python
#4)Get the cipher encrypted using shift cipher. Identify the key used to encrypt using brute force ie all the values in the key space

def brute_force_caesar(cipher_text):
    for shift in range(1, 26):
        decrypted_text = ""
        for char in cipher_text:
            if char.isalpha():
                shift_amount = shift % 26
                if char.islower():
                    decrypted_text += chr((ord(char) - shift_amount - 97) % 26 + 97)
                elif char.isupper():
                    decrypted_text += chr((ord(char) - shift_amount - 65) % 26 + 65)
            else:
                decrypted_text += char
        print(f"Key {shift}: {decrypted_text}")

cipher_text = input("Enter the Caesar cipher text: ")

brute_force_caesar(cipher_text)
```

```
┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
Enter the Caesar cipher text: khoor
Key 1: jgnnq
Key 2: ifmmp
Key 3: hello
Key 4: gdkkn
Key 5: fcjjm
Key 6: ebiil
Key 7: dahhk
Key 8: czggj
Key 9: byffi
Key 10: axeeh
Key 11: zwddg
Key 12: yvccf
Key 13: xubbe
Key 14: wtaad
Key 15: vszzc
Key 16: uryyb
Key 17: tqxxa
Key 18: spwwz
Key 19: rovvy
Key 20: qnuux
Key 21: pmttw
Key 22: olssv
Key 23: nkrru
Key 24: mjqqt
Key 25: lipps

┌──(kali㉿kali)-[~]
└─$
```

5. Find the k value , Provided cipher text and plain text

```python
#5. Find the k value , Provided cipher text and plain text

def find_key(plain_text, cipher_text):
    for i in range(len(plain_text)):
        if plain_text[i].isalpha() and cipher_text[i].isalpha():
            k = (ord(cipher_text[i]) - ord(plain_text[i])) % 26
            return k
    return None

plain_text = input("Enter the plain text: ")
cipher_text = input("Enter the cipher text: ")

key = find_key(plain_text, cipher_text)

if key is not None:
    print(f"The key (shift value) used is: {key}")
else:
    print("Could not determine the key.")
```

```
┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
Enter the plain text: hello
Enter the cipher text: khoor
The key (shift value) used is: 3
```

6. Encrypt and decrypt the string using Atbash cipher

```python
#6)Encrypt and decrypt the string using Atbash cipher

def atbash_cipher(text):
    result = ""
    for char in text:
        if char.isalpha():
            if char.islower():
                result += chr(122 - (ord(char) - 97))  # 'a' → 'z', 'b' → 'y', etc.
            elif char.isupper():
                result += chr(90 - (ord(char) - 65))  # 'A' → 'Z', 'B' → 'Y', etc.
        else:
            result += char  # Non-alphabetic characters remain unchanged
    return result

input_text = input("Enter the text to encrypt/decrypt using Atbash cipher: ")

output_text = atbash_cipher(input_text)
print("Output:", output_text)
```

```
┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
Enter the text to encrypt/decrypt using Atbash cipher: hello, world
Output: svool, dliow
```

7. Encrypt and decrypt using Affine cipher

add validation

```python
#7)Encrypt and decrypt using Affine cipher
def mod_inverse(a, m):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None

def affine_encrypt(text, a, b):
    encrypted_text = ""
    for char in text:
        if char.isalpha():
            shift = ord('A') if char.isupper() else ord('a')
            encrypted_text += chr(((a * (ord(char) - shift) + b) % 26) + shift)
        else:
            encrypted_text += char
    return encrypted_text

def affine_decrypt(cipher_text, a, b):
    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        return "Invalid key 'a': No modular inverse exists."

    decrypted_text = ""
    for char in cipher_text:
        if char.isalpha():
            shift = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr(((a_inv * ((ord(char) - shift) - b)) % 26) + shift)
        else:
            decrypted_text += char
    return decrypted_text

def validate_key(a):
    return mod_inverse(a, 26) is not None

text = input("Enter the text: ")
a = int(input("Enter the value for 'a' (must be coprime with 26): "))
b = int(input("Enter the value for 'b': "))
```

```
┌──(kali㉿kali)-[~]
└─$ python ctlab4.py
Enter the text: hello
Enter the value for 'a' (must be coprime with 26): 5
Enter the value for 'b': 8
Encrypted text: rclla
Decrypted text: hello
```

The harder you work for something, the greater you will feel when you achieve it.

Do not limit your challenges challenge your limit