Tasks 1

**Implement decryption:* *Write a function that can decrypt a text encrypted using the Caesar cipher. The decryption function should take two arguments: the encrypted text and the shift value used for encryption. Use a similar algorithm as the encryption function, except reverse the direction of the shift.

```
def caeser_decrypt(ciphertext,shift):
  decrypted_text=""
  for char in ciphertext:
    if char.isalpha():
      ascii_offset = ord('A') if char.isupper() else ord('a')
      decrypted_char = chr((ord(char) - ascii_offset + shift) % 26 + ascii_offset)
      decrypted_text += decrypted_char
    else:
      decrypted_text += char
  return decrypted_text
encrypted_text = input("encrypted text: ")
shift_value = int(input("shift value: "))
decrypted_text = caeser_decrypt(encrypted_text, shift_value)
print("decrypted text: ",decrypted_text)
```

```
encrypted text: aksfhg
shift value: 2
decrypted text:  cmuhji
```

Implement headers: Modify the code to add a “header” to your encrypted message that specifies the shift value used for encryption. With this header, make your decryption function automatically determine the shift value without the user having to manually input it.

```
def caesar_decrypt(ciphertext,shift):
  decrypted_text=""
  for char in ciphertext:
    if char.isalpha():
      ascii_offset=ord('A') if char.isupper() else ord('a')
      decrypted_char = chr((ord(char)- ascii_offset - shift_value) % 26 +ascii_offset)
      decrypted_text += decrypted_char
    else:
      decrypted_text += char
  return decrypted_text
encrypted_text = input("enter encrypteds text: ")

try:
  header= encrypted_text.index(':')
  shift_value = int(encrypted_text[:header])
  decrypted_text=caesar_decrypt(encrypted_text[header+1:],shift_value)
except ValueError:
  shift_value=0
  decrypted_text=caesar_decrypt(encrypted_text,shift_value)
except Exception as e:
  print("an error occurinf in decryption", str(e))
else:
  print("decrypted text: ",decrypted_text)
```

```
enter encrypteds text: 2:alan
decrypted text:  yjyl
```

Tasks 2

Implement brute-force attacks: Write a function to perform a brute-force attack on a Caesar cipher-encrypted message. The brute-force function should try all possible shift values and output the decrypted message for each shift value. This way, the user can see all possible decrypted messages and manually determine which is the correct decryption. (tip, there are only 26 letters in the alphabeta)

```
def brute_force(ciphertext):
  for shift in range(26):
    try:
      decrypted_text=''
      for char in ciphertext:
        if char.isalpha():
          ascii_offset =ord('A') if char.isupper() else ord('a')
          decrypted_char = chr((ord(char)- ascii_offset - shift_value) % 26 +ascii_offset)
          decrypted_text += decrypted_char
        else:
          decrypted_text +=char
      print(f"shift value:  {shift}\tDecrypted text : {decrypted_text}")
    except Exception as e:
      print(f"an error occured in decryption with the shift value{shift}:{str(e)}")
ciphertext=input("enter encrypted test: ")
brute_force(ciphertext)
```

```
enter encrypted test: lol
shift value:  0 Decrypted text : jmj
shift value:  1 Decrypted text : jmj
shift value:  2 Decrypted text : jmj
shift value:  3 Decrypted text : jmj
shift value:  4 Decrypted text : jmj
shift value:  5 Decrypted text : jmj
shift value:  6 Decrypted text : jmj
shift value:  7 Decrypted text : jmj
shift value:  8 Decrypted text : jmj
shift value:  9 Decrypted text : jmj
shift value:  10        Decrypted text : jmj
shift value:  11        Decrypted text : jmj
shift value:  12        Decrypted text : jmj
shift value:  13        Decrypted text : jmj
shift value:  14        Decrypted text : jmj
shift value:  15        Decrypted text : jmj
shift value:  16        Decrypted text : jmj
shift value:  17        Decrypted text : jmj
shift value:  18        Decrypted text : jmj
shift value:  19        Decrypted text : jmj
shift value:  20        Decrypted text : jmj
shift value:  21        Decrypted text : jmj
shift value:  22        Decrypted text : jmj
shift value:  23        Decrypted text : jmj
shift value:  24        Decrypted text : jmj
shift value:  25        Decrypted text : jmj
```

Implement upper and lower limits for shift values: Modify your code to restrict the shift value within a certain range to add more security to your encryption algorithm. Compute the improved range by computing the character count of the plaintext. So for example, if the input text has 100 characters, then the shift range can be set to 1-50.

```
def brute_force(ciphertext):
  ciphertext_length = sum(char.isalpha() for char in ciphertext)
  lower_limit = 1
  upper_limit = min(ciphertext_length // 2, 50) if ciphertext_length > 1 else 1

  for shift in range (lower_limit,upper_limit+1):
    try:
      decrypted_text=''
      for char in ciphertext:
        if char.isalpha():
          ascii_offset =ord('A') if char.isupper() else ord('a')
          decrypted_char = chr((ord(char)- ascii_offset - shift_value) % 26 +ascii_offset)
          decrypted_text += decrypted_char
        else:
          decrypted_text +=char
      print(f"shift value:  {shift}\tDecrypted text : {decrypted_text}")
    except Exception as e:
      print(f"an error occured in decryption with the shift value{shift}:{str(e)}")
ciphertext=input("enter encrypted text: ")
brute_force(ciphertext)
```

```
enter encrypted text: lol
shift value:  1 Decrypted text : jmj
```

✓ 3s completed at 8:57 AM