**CS458**

**Name: Yash Vardhan Sharma**

**ID: A20575676**

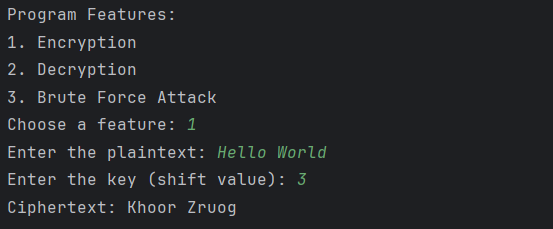
**Coding Assignment 1**

**Code:**

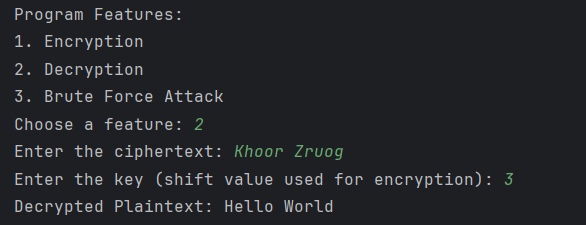
import string  
  
def shift\_cipher\_encrypt(plaintext, key):  
 ciphertext = ""  
 if not key.lstrip('-').isdigit(): # Checking if key is numeric or not, to throw error if key is non-numeric  
 raise ValueError("Error: The key is non-numeric.")  
 elif plaintext == "": # Checking if plaintext is empty or not, to throw error if plaintext is empty  
 raise ValueError("Error: The plaintext is an empty string.")  
 else:  
 for char in plaintext: # Going to perform encryption on each character of the plaintext  
 if char.islower(): # Checking if plaintext character is in lowercase  
 shifted\_char = (ord(char) - ord('a') + int(key)) % 26 + ord('a') # Getting the shift in the character. The algorithm is first removing the ASCII value of 'a' from the character of plaintext to get the difference of the character from the start of alphabet. Then adding the key to it, and doing a modulo by 26(char-'a'+key % 26) to get the remainder so if the sum is greater than 26(if key is added so the sum exceeds the ASCII value of 'z'), we restart the count from the start of the alphabets.  
 ciphertext = ciphertext + chr(shifted\_char) # Adding shifted character to a string to get the ciphertext word  
 elif char.isupper(): # Checking if plaintext character is in uppercase  
 shifted\_char = (ord(char) - ord('A') + int(key)) % 26 + ord('A') # Getting the shift in the character. The algorithm is first removing the ASCII value of 'A' from the character of plaintext to get the difference of the character from the start of alphabet. Then adding the key to it, and doing a modulo by 26(char-'A'+key % 26) to get the remainder so if the sum is greater than 26(if key is added so the sum exceeds the ASCII value of 'Z'), we restart the count from the start of the alphabets.  
 ciphertext = ciphertext + chr(shifted\_char) # Adding shifted character to a string to get the ciphertext word  
 elif char in string.punctuation: # Checking if plaintext character is a punctuation  
 ciphertext = ciphertext + char # Adding plaintext character to ciphertext string directly  
 elif char.isspace(): # Checking if plaintext character is a white space  
 ciphertext = ciphertext + char # Adding plaintext character to ciphertext string directly  
 else: # If none of the above conditions satisfy, throw an error.  
 raise ValueError("Error: The plaintext is numeric.")  
 return ciphertext  
  
def shift\_cipher\_decrypt(ciphertext, key):  
 plaintext = ""  
 if not key.lstrip('-').isdigit(): # Checking if key is numeric or not, to throw error if key is non-numeric  
 raise ValueError("Error: The key is non-numeric.")  
 elif ciphertext == "": # Checking if plaintext is empty or not, to throw error if plaintext is empty  
 raise ValueError("Error: The ciphertext is an empty string.")  
 else:  
 for char in ciphertext: # Going to perform decryption on each character of the ciphertext  
 if char.islower(): # Checking if plaintext character is in lowercase  
 shifted\_char = (ord(char) - ord('a') - int(key)) % 26 + ord('a') # Getting the shift in the character. The algorithm is first removing the ASCII value of 'a' from the character of plaintext to get the difference of the character from the start of alphabet. Then subtracting the key from it, and doing a modulo by 26(char-'a'-key % 26) to get the remainder so if the result is less than 0(if key is subtracted so the result is less than the ASCII value of 'a'), we restart the count from the end of the alphabets.  
 plaintext = plaintext + chr(shifted\_char) # Adding shifted character to a string to get the plaintext word  
 elif char.isupper(): # Checking if plaintext character is in uppercase  
 shifted\_char = (ord(char) - ord('A') - int(key)) % 26 + ord('A') # Getting the shift in the character. The algorithm is first removing the ASCII value of 'A' from the character of plaintext to get the difference of the character from the start of alphabet. Then subtracting the key from it, and doing a modulo by 26(char-'A'-key % 26) to get the remainder so if the result is less than 0(if key is subtracted so the result is less than the ASCII value of 'A'), we restart the count from the end of the alphabets.  
 plaintext = plaintext + chr(shifted\_char) # Adding shifted character to a string to get the plaintext word  
 elif char in string.punctuation: # Checking if plaintext character is a punctuation  
 plaintext = plaintext + char # Adding ciphertext character to plaintext string directly  
 elif char.isspace(): # Checking if plaintext character is a white space  
 plaintext = plaintext + char # Adding ciphertext character to plaintext string directly  
 else: # If none of the above conditions satisfy, throw an error.  
 raise ValueError("Error: The ciphertext is numeric.")  
 return plaintext  
  
def brute\_force\_attack(ciphertext):  
 possible\_plaintexts = [] # Array to store all the possible plaintexts for different keys  
 for key in range(26): # Running the loop for executing shift\_cipher\_decrypt function for all the values of key and adding them to the array "possible\_plaintexts"  
 decrypted\_text = shift\_cipher\_decrypt(ciphertext, key)  
 possible\_plaintexts.append((key, decrypted\_text))  
 return possible\_plaintexts  
  
def main():  
 while True: # Giving the user menu with options for encryption, decryption and brute-force.  
 print("\nProgram Features:")  
 print("1. Encryption")  
 print("2. Decryption")  
 print("3. Brute Force Attack")  
 choice = input("Choose a feature: ")  
 try: # Using try-except block to catch any error thrown.  
 if choice == '1': # User selects encryption, enters the value of plaintext, key and get the value of ciphertext  
 plaintext = input("Enter the plaintext: ")  
 key = input("Enter the key (shift value): ")  
 ciphertext = shift\_cipher\_encrypt(plaintext, key)  
 print("Ciphertext:", ciphertext)  
  
 elif choice == '2': # User selects decryption, enters the value of ciphertext, key and get the value of plaintext  
 ciphertext = input("Enter the ciphertext: ")  
 key = input("Enter the key (shift value used for encryption): ")  
 plaintext = shift\_cipher\_decrypt(ciphertext, key)  
 print("Decrypted Plaintext:", plaintext)  
  
 elif choice == '3': # User selects brute-force, enters the value of ciphertext, key and get the value of all the possible plaintexts  
 ciphertext = input("Enter the ciphertext: ")  
 possible\_plaintexts = brute\_force\_attack(ciphertext)  
 print("Possible plaintext results from brute force attack:")  
 for key, plaintext in possible\_plaintexts:  
 print(f"Key: {key}, Plaintext: {plaintext}")  
 else: # Handles wrong value for menu choice  
 print("Invalid choice. Please try again.")  
 except ValueError as error: # Handling error thrown during the execution of the code.  
 print(error)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Outputs:**

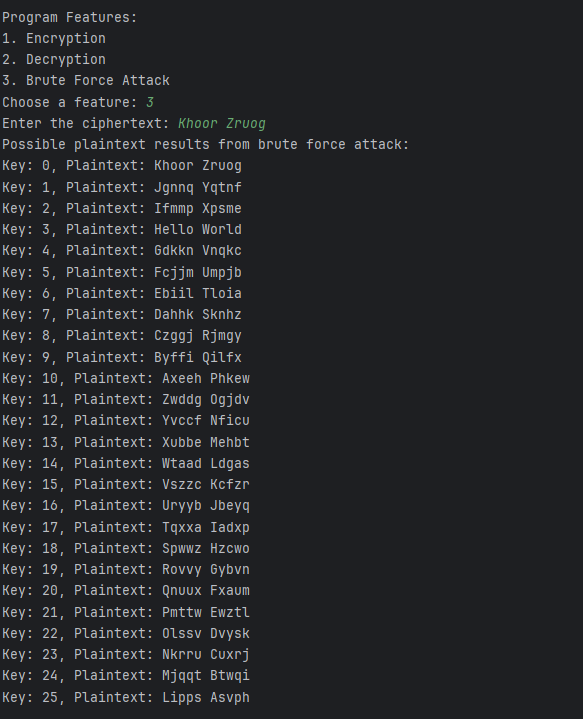
**Encryption:**



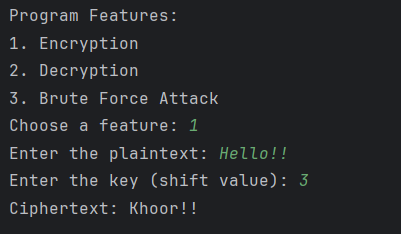
**Decryption:**



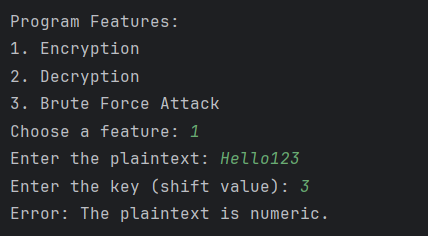
**Brute-Force Attack:**



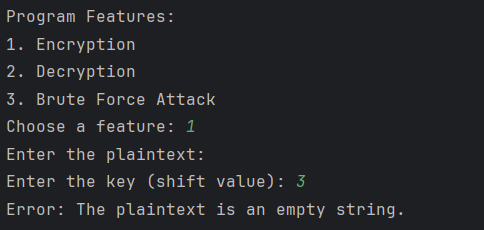
**Encryption with special character:**



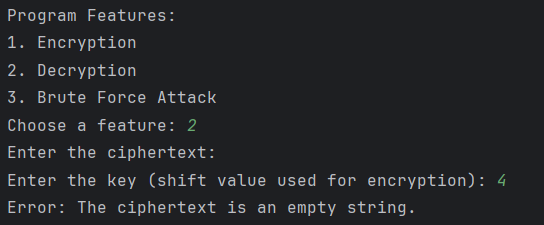
**Error - Plaintext containing numeric:**



**Error - Plaintext is an empty string:**



**Error - Ciphertext is an empty string:**



**Error - Key is non-numeric:**

