Name: Dean Allen V. Dela Cruz
Course and Section: BSIT 4-2

main.py

```python
import random  # Importing random module for generating random numbers

# Function to check if a number is prime
def check_prime(num):
    if num < 2:  # Numbers less than 2 are not prime
        return False
    for i in range(2, num // 2 + 1):  # Check divisors from 2 to half of the number
        if num % i == 0:  # If divisible by any number, it's not prime
            return False
    return True  # Return True if no divisors are found

# Function to generate a random prime number within a given range
def find_prime(start, end):
    candidate = random.randint(start, end)  # Start with a random number in range
    while not check_prime(candidate):  # Keep trying until a prime number is found
```

main.py

```python
        candidate = random.randint(start, end)  # Generate a new random number
    return candidate  # Return the prime number

# Function to calculate the greatest common divisor (GCD)
def gcd(a, b):
    while b != 0:  # Continue until remainder becomes 0
        a, b = b, a % b  # Update values using the Euclidean algorithm
    return a  # Return the greatest common divisor

# Function to calculate the modular inverse
def modular_inverse(public_key, totient_value):
    for private_key in range(3, totient_value):  # Test values for private_key
        if (public_key * private_key) % totient_value == 1:  # Check modular condition
            return private_key  # Return private_key if condition is satisfied
    raise ValueError("Modular inverse does not exist!")  # Raise error if not found

```

Name: Dean Allen V. Dela Cruz
Course and Section: BSIT 4-2

main.py

```python
# Generate two distinct prime numbers
prime_one = find_prime(1, 50)  # Find the first prime number
prime_two = find_prime(1, 50)  # Find the second prime number
while prime_one == prime_two:  # Ensure the primes are distinct
    prime_two = find_prime(1, 50)  # Generate a new prime if they are the same

# Calculate modulus (n) and totient (phi)
modulus = prime_one * prime_two  # n = product of the two primes
totient = (prime_one - 1) * (prime_two - 1)  # phi = (p1-1)*(p2-1)

# Generate the public key (e)
public_key = random.randint(3, totient - 1)  # Randomly choose e in range
while gcd(public_key, totient) != 1:  # Ensure e is coprime with phi
    public_key = random.randint(3, totient - 1)  # Retry with another value

# Calculate the private key (d)
private_key = modular_inverse(public_key, totient)  # Compute
```

main.py

```python
modular inverse of e mod phi

# Display the generated keys and important values
print("Prime number 1:", prime_one)
print("Prime number 2:", prime_two)
print("Public Key (e):", public_key)  # Public key used for encryption
print("Private Key (d):", private_key)  # Private key used for decryption
print("Modulus (n):", modulus)  # Modulus shared between public and private keys
print("Totient (phi):", totient, "(secret)")  # Totient is a secret value

# Input message for encryption
message = input("Enter the message to encrypt: ")

# Convert the message to ASCII values
ascii_values = [ord(char) for char in message]  # Convert characters to ASCII codes
```

Name: Dean Allen V. Dela Cruz
Course and Section: BSIT 4-2

```python
print("Message in ASCII code:", ascii_values)

# Encrypt the message using the public key
encrypted_message = [pow(char, public_key, modulus) for char in ascii_values]  # Perform encryption
print("Encrypted Message (Ciphertext):", encrypted_message)

# Decrypt the message using the private key
decrypted_ascii = [pow(char, private_key, modulus) for char in encrypted_message]  # Perform decryption
print("Decrypted ASCII values:", decrypted_ascii)

# Convert decrypted ASCII values back to characters
decrypted_message = ''.join(chr(char) for char in decrypted_ascii)  # Retrieve original text
print("Decrypted Message (Text):", decrypted_message)

```

Output Clear

```
Prime number 1: 31
Prime number 2: 5
Public Key (e): 37
Private Key (d): 13
Modulus (n): 155
Totient (phi): 120 (secret)
Enter the message to encrypt: hello
Message in ASCII code: [104, 101, 108, 108, 111]
Encrypted Message (Ciphertext): [44, 126, 23, 23, 71]
Decrypted ASCII values: [104, 101, 108, 108, 111]
Decrypted Message (Text): hello

=== Code Execution Successful ===
```