```
""" Assignment 2 -- Implementing the RSA cryptosystem

 Advanced Algorithms 1 (7081) --- Spring 2021

             Team Members -- Homework Group 19:

 1. Harshitha Banda
 2. Hemanth Krishna Paidipalli
 3. Ravindra Guntaka
 4. Vishnu Koteswara Reddy Katam

"""


import random

bearcatii= " ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# The following method converts a message(String) into its BEARCATII (base 27) form.
def convert_message_to_number(input_message):
  message_in_number=0
  j=0
  for i in range(len(input_message)-1,-1,-1):
    position=bearcatti.index(input_message[i].upper())
    message_in_number=message_in_number+position*pow(27,j)
    j=j+1
  return message_in_number

#The following method converts a decimal in to its message(String).

def convert_number_to_message(messagee_in_number):
  result=""
  while(messagee_in_number>0):
    quotient=messagee_in_number//27
    remainder=messagee_in_number%27
    result=result+bearcatii[remainder]
    messagee_in_number=quotient
  return result[::-1]

#This method calculates the modular exponentiation (base^exponent mod(n))
def modular_exponentiation(base,exponent,n):
  result = 1
  base = base % n
  while exponent > 0:
    if ((exponent & 1) == 1):
      result = (result * base) % n
    exponent = exponent >> 1
    base = (base * base) % n
  return result

"""The below method tests if the input number is a prime number following the
```

```
 Miller-Rabin Prime Testability probabilistic algorithm."""

def miller_rabin_prime_test(number, k_times):
  if (number <= 1 or number == 4):
    return False
  if (number <= 3):
    return True
  while (k_times > 0):
    random_a = 2 + (random.randrange(2, number - 2) % (number - 4))
    if (modular_exponentiation(random_a, number - 1, number) != 1):
      return False
    k_times = k_times - 1
  return True

# The method is the implementation of Extended Euclidian GCD as discussed in the class.

def extended_euclidian_gcd(a,b):
  if b==0:
    return a, 1, 0
  else:
    r = a%b
    q = a//b
    gcd,s,t = extended_euclidian_gcd(b,r)
    s_temp = s
    s = t
    t = s_temp - t*q
    return gcd,s,t

#The below method calculates the private key d.

def calculate_private_key_d(public_key_e, phi_n):
  gcd, s, t = extended_euclidian_gcd(public_key_e, phi_n)
  private_key_d = s+phi_n
  return private_key_d

#This method gives us the phi_n
def phi_n(p, q):
  return (p-1)*(q-1)

#This method generates prime numbers
def generate_prime_numbers():
  both_prime=True
  while(both_prime):
    first_prime=random.randrange(1000000000000000, 5000000000000000)
    second_prime= random.randrange(1000000000000000, 5000000000000000)
    if(miller_rabin_prime_test(first_prime,3) and miller_rabin_prime_test(second_prime,3)):
      both_prime=False
  return first_prime,second_prime


#print(generate_prime_numbers())
```

```
def calculate_n(p,q):
  return p*q

""" The executaion of the code runs from here. Here at first the program asks for the user
    to give the public key e. The program continues to ask for e till the condition
    gcd(e,phi_n) is equals to 1. Then it asks for user to input the message in the form of st
    with the condition that the input string can only contain alphabets and a space charecter
    And it outputs in the form of print statement"""

def main():
  print("Hi! Welcome to RSA Cryptosystem")
  p,q = generate_prime_numbers()
  n= calculate_n(p,q)
  phi_nn=phi_n(p,q)
  correct_e=True
  public_key_e=0
  while(correct_e):
    public_key_e=int(input("Please enter the public key (Hint: Try to enter a prime number) :
    gcd,s,t=extended_euclidian_gcd(public_key_e,phi_nn)
    if(gcd==1):
      correct_e=False
      break

  print("You have entered public key e as: "+str(public_key_e))

  M_text_message=input("Please enter the message you want to send: ")

  print("you have entered message as: "+M_text_message)

  m=convert_message_to_number(M_text_message)

  encrypted_message_in_number_form=modular_exponentiation(m,public_key_e,n)

  C_cyber_text=convert_number_to_message(encrypted_message_in_number_form)

  private_key_d=calculate_private_key_d(public_key_e,phi_nn)

  received_message_in_number=modular_exponentiation(encrypted_message_in_number_form,private_

  P_received_message=convert_number_to_message(received_message_in_number)


  print("p: "+str(p)+" q: "+str(q)+" M: "+M_text_message +" C_in_number: "+str(encrypted_mess

if __name__ == '__main__':
  main()
```

```
Hi! Welcome to RSA Cryptosystem
Please enter the public key (Hint: Try to enter a prime number) : 3
You have entered public key e as: 3
```

```
Please enter the message you want to send: TEST
you have entered message as: TEST
p: 4430657294669591 q: 4454406629442347 M: TEST C_in_number: 62967838987084472 C_cyber_t
```