**Cryptography**

### -Lab report



Submitted By: Deshant Devkota

Submitted To: Suresh Thapal (Lecturer, Cryptography)

Vedas College, Lalitpur, Nepal

**Tribhuvan University**

**Department of Bsc Csit**

Submitted Date: Thursday, June 17, 2021

Submitted Lab Reports:

1. To find the Euler's Totient of a given number.
2. To test whether the given number is prime using a traditional method.
3. To test whether the given number is prime using the Miller-Rabin Algorithm.

## LAB 6

Euler’s Totient

**1. Objectives:**

In this lab we were to find the Euler’s Totient of a given number.

**2.Introduction:**

Euler's totient function φ ( n) is the number of positive integers not exceeding n that have no common divisors with n (other than the common divisor 1). In other words, φ ( n) is the number of integers m coprime to n such that 1 ≤ m ≤ n .

**3.Code:**

def gcd(a, b):

if b == 0:

return a

else:

while b != 0:

a, b = b, a % b

return a

def eulartotient(n):

totientnumber=0

for i in range(1,n):

if gcd(n, i) == 1:

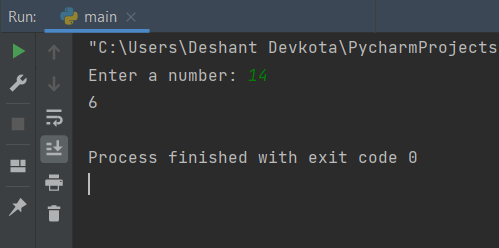
totientnumber += 1

return totientnumber

number = int(input("Enter a number: "))

print(eulartotient(number))

**4.Output:**



## LAB 7

Prime Test - Traditional Method

**1. Objectives:**

In this lab we were to test whether the given number is prime using a traditional method.

**2.Introduction:**

A prime number (or prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. By Euclid's theorem, there are an infinite number of prime numbers. Subsets of the prime numbers may be generated with various formulas for primes.

List of first 10 prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29

**3.Code:**

import math

def isprime(n):

if n == 1 or n == 2:

return True

npt1 = math.floor(math.sqrt(n))+1

for i in range(2, npt1):

if n % i == 0:

return False

return True

number = int(input("Enter a number: "))

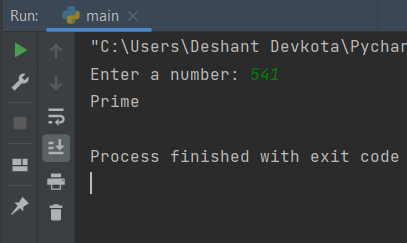
If isprime(number):

print(“Prime”)

else:

print(“Composite”)

**4.Output:**

****

## LAB 9

Miller-Rabin Primality Test

**1.Objectives:**

In this lab we were to test whether the given number is prime using Miller Rabin algorithm.

**2.Introduction:**

A prime number (or prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. By Euclid's theorem, there are an infinite number of prime numbers. Subsets of the prime numbers may be generated with various formulas for primes.

List of first 10 prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29

The Miller–Rabin primality test or Rabin–Miller primality test is a probabilistic primality test: an algorithm which determines whether a given number is likely to be prime.

Miller-Rabin Test:

Let p be an odd number greater than 2.

Then, p-1 can be represented as: p-1 = 2^k \* q, with k >0 and q odd.

Let 1 < a < p.

p is likely to be prime if:

1. a^q ≅ 1 (mod p)
2. For i from 0 to k-1:

a^(2^i \*q) ≅ -1 ≅ p-1 (mod p)

**3.Code:**

import math

import random

def findkq(n):

fkq1 = int(math.log(n, 2))

n = int(n-1)

for i in range(fkq1):

if n % 2 == 0:

n = n / 2

else:

return i, int(n)

return i+1, int(n)

def Miller\_Rabin(n):

k, q = findkq(n)

a = random.randint(2, n-1)

msg = f'{n}-1 = 2 ^ {k} \* {q}, let a = {a}'

print(msg)

if pow(a, q) % n == 1:

return str(n)+" is Prime (1)"

for i in range(k):

if pow(a, pow(2, i) \* q) % n == n-1:

return str(n)+" is Prime"

return str(n)+" is Composite"

# only odd numbers as inputs

print("\nTesting for all odd numbers between 3-100")

print("\nn-1 = 2 ^ q \* k, 2 < a < n\n")

for en in range(3, 100, 2):

print(Miller\_Rabin(en), end="\n\n")

**4.Testing and Output:**

Code for testing:

def bool\_Miller\_Rabin(n):

k, q = findkq(n)

a = random.randint(2, n-1)

if pow(a, q) % n == 1:

return True

for i in range(k):

if pow(a, pow(2, i) \* q) % n == n-1:

return True

return False

def testmiller():

# only odd numbers as inputs

list\_prime = []

for en in range(3, 100, 2):

if bool\_Miller\_Rabin(en):

list\_prime.append(en)

print("\nList of all prime number generated using Python's inbuilt function sympy.primerange: ")

print(list(sp.primerange(3, 100)))

print("\nList of all prime number generated using our algorithm: ")

print(list\_prime)

print("\nTesting for all odd numbers between 3-100")

print("\nn-1 = 2 ^ q \* k, 2 < a < n\n")

for en in range(3, 100, 2):

print(Miller\_Rabin(en), end="\n\n")

Run 1:

"C:\Users\Deshant Devkota\PycharmProjects\Crypto\venv\Scripts\python.exe" "C:/Users/Deshant Devkota/PycharmProjects/Crypto/main.py"

List of all prime number generated using Python's inbuilt function sympy.primerange:

[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

List of all prime number generated using our algorithm:

[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

Testing for all odd numbers between 3-100

n-1 = 2 ^ q \* k, 2 < a < n

3-1 = 2 ^ 1 \* 1, let a = 2

3 is Prime

5-1 = 2 ^ 2 \* 1, let a = 4

5 is Prime

7-1 = 2 ^ 1 \* 3, let a = 6

7 is Prime

9-1 = 2 ^ 3 \* 1, let a = 7

9 is Composite

11-1 = 2 ^ 1 \* 5, let a = 6

11 is Prime

13-1 = 2 ^ 2 \* 3, let a = 12

13 is Prime

15-1 = 2 ^ 1 \* 7, let a = 6

15 is Composite

17-1 = 2 ^ 4 \* 1, let a = 15

17 is Prime

19-1 = 2 ^ 1 \* 9, let a = 10

19 is Prime

21-1 = 2 ^ 2 \* 5, let a = 13

21 is Composite

23-1 = 2 ^ 1 \* 11, let a = 21

23 is Prime

25-1 = 2 ^ 3 \* 3, let a = 23

25 is Composite

27-1 = 2 ^ 1 \* 13, let a = 4

27 is Composite

29-1 = 2 ^ 2 \* 7, let a = 23

29 is Prime (1)

31-1 = 2 ^ 1 \* 15, let a = 26

31 is Prime

33-1 = 2 ^ 5 \* 1, let a = 7

33 is Composite

35-1 = 2 ^ 1 \* 17, let a = 5

35 is Composite

37-1 = 2 ^ 2 \* 9, let a = 36

37 is Prime

39-1 = 2 ^ 1 \* 19, let a = 12

39 is Composite

41-1 = 2 ^ 3 \* 5, let a = 4

41 is Prime

43-1 = 2 ^ 1 \* 21, let a = 21

43 is Prime (1)

45-1 = 2 ^ 2 \* 11, let a = 31

45 is Composite

47-1 = 2 ^ 1 \* 23, let a = 33

47 is Prime

49-1 = 2 ^ 4 \* 3, let a = 5

49 is Composite

51-1 = 2 ^ 1 \* 25, let a = 49

51 is Composite

53-1 = 2 ^ 2 \* 13, let a = 4

53 is Prime

55-1 = 2 ^ 1 \* 27, let a = 51

55 is Composite

57-1 = 2 ^ 3 \* 7, let a = 9

57 is Composite

59-1 = 2 ^ 1 \* 29, let a = 14

59 is Prime

61-1 = 2 ^ 2 \* 15, let a = 56

61 is Prime (1)

63-1 = 2 ^ 1 \* 31, let a = 45

63 is Composite

65-1 = 2 ^ 6 \* 1, let a = 32

65 is Composite

67-1 = 2 ^ 1 \* 33, let a = 16

67 is Prime (1)

69-1 = 2 ^ 2 \* 17, let a = 3

69 is Composite

71-1 = 2 ^ 1 \* 35, let a = 5

71 is Prime (1)

73-1 = 2 ^ 3 \* 9, let a = 50

73 is Prime

75-1 = 2 ^ 1 \* 37, let a = 17

75 is Composite

77-1 = 2 ^ 2 \* 19, let a = 74

77 is Composite

79-1 = 2 ^ 1 \* 39, let a = 12

79 is Prime

81-1 = 2 ^ 4 \* 5, let a = 4

81 is Composite

83-1 = 2 ^ 1 \* 41, let a = 49

83 is Prime (1)

85-1 = 2 ^ 2 \* 21, let a = 46

85 is Composite

87-1 = 2 ^ 1 \* 43, let a = 53

87 is Composite

89-1 = 2 ^ 3 \* 11, let a = 26

89 is Prime

91-1 = 2 ^ 1 \* 45, let a = 30

91 is Composite

93-1 = 2 ^ 2 \* 23, let a = 7

93 is Composite

95-1 = 2 ^ 1 \* 47, let a = 17

95 is Composite

97-1 = 2 ^ 5 \* 3, let a = 93

97 is Prime

99-1 = 2 ^ 1 \* 49, let a = 94

99 is Composite

Process finished with exit code 0

Run2:

//Only showing the list for comparison:

List of all prime number generated using Python's inbuilt function sympy.primerange:

[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

List of all prime number generated using our algorithm:

[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 39, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

**5.Conclusion**

Thus Miller-Test will properly check and return True if the number is prime, however it might fail to test the composite number properly and return as prime incorrectly. Or simply it checks if the number is likely to be prime.

In our first test run, all odd numbers from 3-100 are checked correctly for being prime, while in the second run, 39 is incorrectly identified as prime.