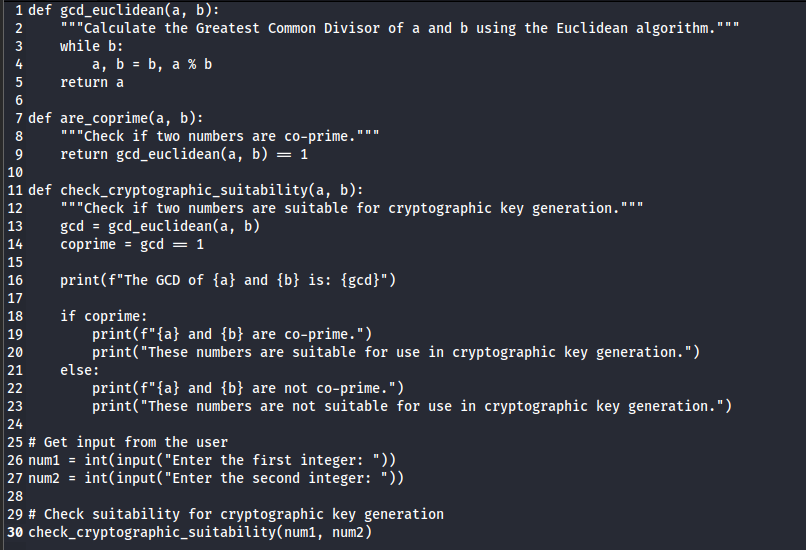
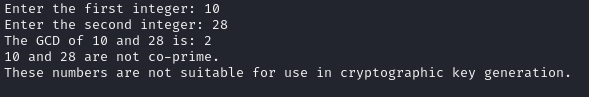
1. Write a Python script that takes two integers as input and calculates their GCD using the Euclidean algorithm.

Based on the result, determine whether these numbers are co-prime.

If they are co-prime, print a message indicating that they can be used in cryptographic key generation; otherwise, print a message that they are not suitable.





2. Write a python script to take two integer values (number (n) and modulo (m)) from the user and find the modular inverse using extended Euclidean algorithm. \

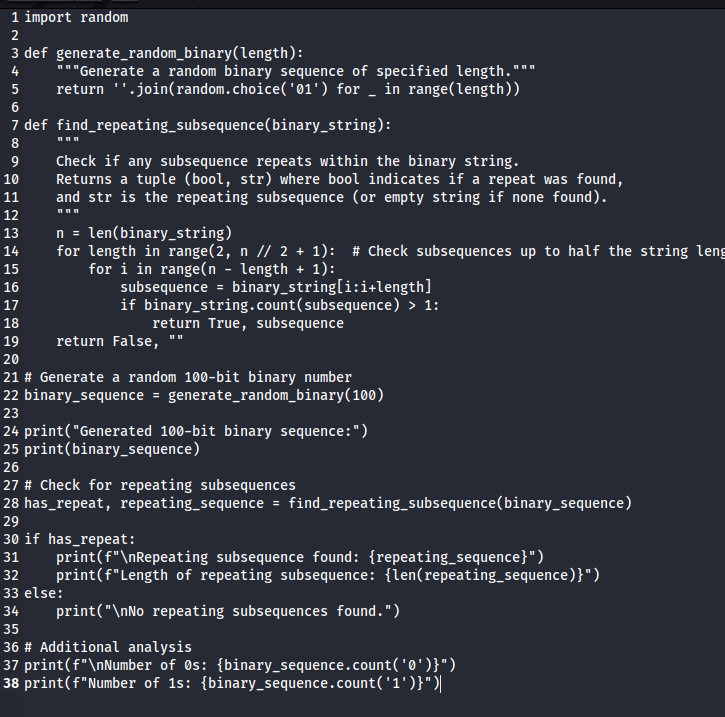
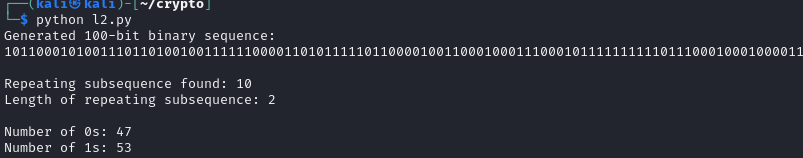
A screenshot of a computer program

Description automatically generated A black background with white text

Description automatically generated

3. Write a Python script that generates a random binary number of length 100. The output should be a string of 100 binary digits (0s and 1s).

After generating the binary sequence, implement a function to check whether any subsequence of digits repeats itself within the sequence.

4. Write a Python script that performs the Golomb test to the numbers provided below.

101011001010

111111000000

The script should

- Perform and print the results of the three Golomb tests on the sequence.

- Print a message indicating whether the sequence passes the Golomb tests or not.

def count\_runs(sequence):

"""Count the number of runs of 0s and 1s in the sequence."""

runs = []

current\_run = 1

for i in range(1, len(sequence)):

if sequence[i] == sequence[i-1]:

current\_run += 1

else:

runs.append(current\_run)

current\_run = 1

runs.append(current\_run)

return runs

def golomb\_test(sequence):

n = len(sequence)

ones = sequence.count('1')

zeros = sequence.count('0')

runs = count\_runs(sequence)

# Test 1: Balance property

balance = abs(ones - zeros) <= 1

print(f"Test 1 (Balance property): {'Passed' if balance else 'Failed'}")

print(f" Number of 1s: {ones}")

print(f" Number of 0s: {zeros}")

# Test 2: Run property

expected\_runs = (n + 1) // 2

actual\_runs = len(runs)

run\_property = abs(actual\_runs - expected\_runs) <= 2

print(f"Test 2 (Run property): {'Passed' if run\_property else 'Failed'}")

print(f" Expected number of runs: {expected\_runs}")

print(f" Actual number of runs: {actual\_runs}")

# Test 3: Run length property

run\_lengths = {i: runs.count(i) for i in range(1, max(runs) + 1)}

run\_length\_property = True

for k in range(1, len(run\_lengths)):

if k+1 in run\_lengths:

if run\_lengths[k] < run\_lengths[k+1]:

run\_length\_property = False

break

print(f"Test 3 (Run length property): {'Passed' if run\_length\_property else 'Failed'}")

print(" Run lengths:")

for length, count in run\_lengths.items():

print(f" Length {length}: {count} runs")

# Overall result

passed\_all = balance and run\_property and run\_length\_property

print(f"\nOverall result: {'Passed' if passed\_all else 'Failed'} all Golomb tests")

return passed\_all

# Test sequences

sequences = [

"101011001010",

"111111000000"

]

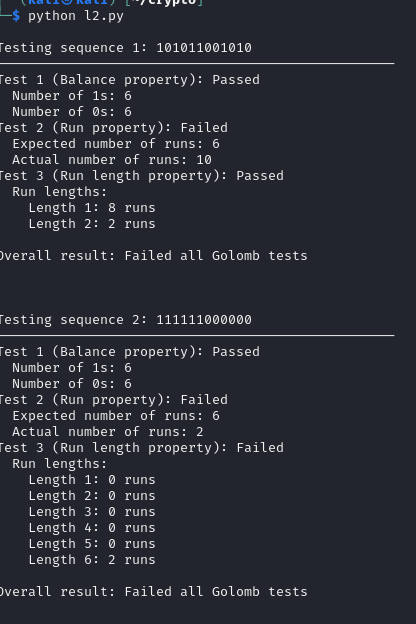
for i, seq in enumerate(sequences, 1):

print(f"\nTesting sequence {i}: {seq}")

print("-" \* 50)

golomb\_test(seq)

print("\n")



================================================================================================

"Always do your best. What you plant now, you will harvest later." —Og Mandino

================================================================================================