import random

def hamming\_encode(data\_bits):

d1, d2, d3, d4 = data\_bits

p1 = d1 ^ d2 ^ d4

p2 = d1 ^ d3 ^ d4

p3 = d2 ^ d3 ^ d4

return [p1, p2, d1, p3, d2, d3, d4]

def inject\_error(encoded\_bits):

bits = encoded\_bits[:]

index = random.randint(0, 6)

bits[index] ^= 1 # Flip the bit

print(f" Error injected at bit position {index + 1}")

return bits, index + 1

def hamming\_decode(received):

p1, p2, d1, p3, d2, d3, d4 = received

s1 = p1 ^ d1 ^ d2 ^ d4

s2 = p2 ^ d1 ^ d3 ^ d4

s3 = p3 ^ d2 ^ d3 ^ d4

error\_position = s3 \* 4 + s2 \* 2 + s1

if error\_position == 0:

print("No error detected in received message.")

else:

print(f"Error detected at bit position {error\_position}. Correcting...")

received[error\_position - 1] ^= 1

corrected\_data = [received[2], received[4], received[5], received[6]]

return corrected\_data

def run\_simulation():

print("=== Hamming Code (7,4) Simulation ===")

message = input("Enter 4-bit message (e.g. 1011): ")

if len(message) != 4 or any(bit not in "01" for bit in message):

print("Invalid input. Please enter exactly 4 bits.")

return

data\_bits = [int(bit) for bit in message]

print(f" Original Data: {data\_bits}")

encoded = hamming\_encode(data\_bits)

print(f" Encoded Message: {encoded}")

corrupted, error\_index = inject\_error(encoded)

print(f" Received Message: {corrupted}")

corrected\_data = hamming\_decode(corrupted)

print(f" Corrected Data Extracted: {corrected\_data}")

if corrected\_data == data\_bits:

print(" Message successfully corrected and recovered!")

else:

print("⚠Correction failed.")

if \_\_name\_\_ == "\_\_main\_\_":

run\_simulation()