program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”

#!/usr/bin/env python3

"""

Simple automated letter-frequency attack on an additive cipher (Caesar-style).

Produces candidate plaintexts ordered by chi-square score (lower = more likely English).

"""

from collections import Counter

import math

# Standard English letter frequency percentages (A..Z)

EN\_FREQ = [

8.167, 1.492, 2.782, 4.253, 12.702, 2.228, 2.015, 6.094, 6.966, 0.153,

0.772, 4.025, 2.406, 6.749, 7.507, 1.929, 0.095, 5.987, 6.327, 9.056,

2.758, 0.978, 2.361, 0.150, 1.974, 0.074

]

def shift\_decrypt(ciphertext: str, shift: int) -> str:

"""Decrypt ciphertext with additive shift (0..25). Non-letters are preserved."""

result\_chars = []

for ch in ciphertext:

if 'A' <= ch <= 'Z':

result\_chars.append(chr((ord(ch) - ord('A') - shift) % 26 + ord('A')))

elif 'a' <= ch <= 'z':

result\_chars.append(chr((ord(ch) - ord('a') - shift) % 26 + ord('a')))

else:

result\_chars.append(ch)

return ''.join(result\_chars)

def chi\_squared\_score(text: str) -> float:

"""

Compute chi-square statistic between observed letter counts in text and expected English counts.

Lower chi-square indicates a better fit to English letter frequency.

"""

# Count letters A..Z (case-insensitive)

letters = [c.upper() for c in text if c.isalpha()]

N = len(letters)

if N == 0:

return float('inf') # no letters -> not useful

obs = Counter(letters)

chi2 = 0.0

for i, pct in enumerate(EN\_FREQ):

letter = chr(ord('A') + i)

expected = pct / 100.0 \* N

observed = obs.get(letter, 0)

# avoid division by zero; if expected is extremely small, add a tiny epsilon

exp = expected if expected > 1e-8 else 1e-8

chi2 += (observed - exp) \*\* 2 / exp

return chi2

def attack\_additive(ciphertext: str, top\_n: int = 10):

"""

Try all 26 shifts, score each candidate by chi-square, and return top\_n candidates as a list:

[(shift, score, plaintext), ...] ordered by increasing score (more likely first).

"""

candidates = []

for shift in range(26):

plain = shift\_decrypt(ciphertext, shift)

score = chi\_squared\_score(plain)

candidates.append((shift, score, plain))

# Sort by score (lower = better)

candidates.sort(key=lambda x: x[1])

return candidates[:top\_n]

def pretty\_print\_results(results):

print(f"{'Rank':>4} {'Shift':>5} {'Score':>9} {'Plaintext (start)'}")

print("-" \* 60)

for rank, (shift, score, plain) in enumerate(results, start=1):

# show a snippet of plaintext for readability (first 80 chars)

snippet = plain if len(plain) <= 80 else plain[:77] + "..."

print(f"{rank:>4} {shift:>5} {score:9.3f} {snippet}")

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

# Example interactive usage:

ciphertext = input("Enter ciphertext: ").rstrip("\n")

top\_n\_str = input("How many top candidates do you want? (default 10): ").strip()

try:

top\_n = int(top\_n\_str) if top\_n\_str else 10

except ValueError:

top\_n = 10

results = attack\_additive(ciphertext, top\_n=top\_n)

pretty\_print\_results(results)

