1.0) AIM: To demonstrate how the Caesar (shift) cipher can be broken by a brute-force attack (trying all possible shifts) and to show a basic automated method to select the most likely plain-text using English letter frequency scoring.

2.0) SCOPE:

· Illustrates the weakness of simple substitution ciphers (low key space).

· Brute-force approach and a basic scoring heuristic (frequency match) to pick the correct plaintext automatically.

· Reinforces concepts of cryptanalysis and why modern ciphers are needed.

3.0) FACILITIES/ APPARATUS:

· Windows system with Python installed (≥ 3.6).

· Text editor (Notepad/VS Code) or Python IDE (IDLE).

4.0) THEORY:

· **Caesar (Shift) Cipher:** each letter of plaintext is shifted by a fixed number (key) modulo 26. Example with key=3: A→D, B→E.

· **Brute-Force Attack:** try all possible keys (0..25 for Caesar) and inspect the outputs; for small keyspace the correct key will be found quickly.

· **Frequency Scoring (heuristic):** to automatically choose the most likely plaintext we compare letter frequency of each candidate with expected English letter frequencies (higher match → more likely correct).

Steps

1. Create a cipher text produced by a Caesar cipher (unknown shift).

2. Run a brute-force Python script that tries all 26 shifts and prints candidate plaintexts.
3. Inspect outputs to identify readable English plaintext (manual method).
4. Run the enhanced script that scores each candidate by English letter frequency and selects the best candidate automatically.
5. Record which key recovers the original message and time taken if desired.

Program A — Brute-Force (prints all candidates)

```
# caesar_bruteforce.py# Try all shifts and print candidate plaintexts.
def caesar_decrypt(ciphertext, shift):
    result = []
    for ch in ciphertext:
        if 'A' <= ch <= 'Z':
            result.append(chr((ord(ch) - ord('A') - shift) % 26 + ord('A')))
        elif 'a' <= ch <= 'z':
            result.append(chr((ord(ch) - ord('a') - shift) % 26 + ord('a')))
        else:
            result.append(ch)
    return ''.join(result)
def brute_force(ciphertext):
    print("Brute-force results (shift -> plaintext):\n")
    for k in range(26):
        candidate = caesar_decrypt(ciphertext, k)
        print(f"{k:2d}: {candidate}")
if __name__ == "__main__":
    print("Enter ciphertext (press Enter when done):")
    ct = input().rstrip('\n')
    brute_force(ct)
```

How to run:
Save as `caesar_bruteforce.py`, open terminal/command prompt, `python caesar_bruteforce.py`, paste ciphertext, press Enter.

Output

Enter ciphertext (press Enter when done):

Hello

Brute-force results (shift -> plaintext):

0: Hello

1: Gdkkn

2: Fcjjm

3: Ebiil

4: Dahhk

5: Czggj

6: Byffi

7: Axeeh

8: Zwddg

9: Yvccf

10: Xubbe

11: Wtaad

12: Vszzc

13: Uryyb

14: Tqxxa

15: Spwwz

16: Rovvy

17: Qnuux

18: Pmttw

19: Olssv

20: Nkrru

21: Mjqqt

22: Lipps

23: Khoor

24: Jgnnq

25: Ifmmp

How it works: tries all 26 shifts, computes a simple English-letter-frequency score for each candidate, and lists top matches. Good for short messages and classroom demos.

Program B — Brute-Force + Frequency Scoring (auto-select best)

```
# caesar_score.py# Try all shifts, compute simple English frequency score, and
show ranked candidates.

EN_FREQ = {
    'E': 12.0, 'T': 9.1, 'A': 8.2, 'O': 7.5, 'I': 7.0, 'N': 6.7,
    'S': 6.3, 'R': 6.0, 'H': 6.1, 'L': 4.0, 'D': 4.3, 'C': 2.8,
    'U': 2.8, 'M': 2.4, 'F': 2.2, 'Y': 2.0, 'W': 2.4, 'G': 2.0,
```

```
    'P': 1.9, 'B': 1.5, 'V': 1.0, 'K': 0.8, 'X': 0.15, 'Q': 0.1, 'J': 0.1, 'Z':
0.07
}
def caesar_decrypt(ciphertext, shift):
    result = []
    for ch in ciphertext:
        if 'A' <= ch <= 'Z':
            result.append(chr((ord(ch) - ord('A') - shift) % 26 + ord('A')))
        elif 'a' <= ch <= 'z':
            result.append(chr((ord(ch) - ord('a') - shift) % 26 + ord('a')))
        else:
            result.append(ch)
    return ''.join(result)
def score_text(text):
    # Simple score: sum of EN_FREQ for letters encountered (case-insensitive)
    s = 0.0
    for ch in text.upper():
        if ch.isalpha():
            s += EN_FREQ.get(ch, 0)
    return s
def ranked_candidates(ciphertext):
    candidates = []
    for k in range(26):
        cand = caesar_decrypt(ciphertext, k)
        sc = score_text(cand)
        candidates.append((k, sc, cand))
    # sort by score descending
    candidates.sort(key=lambda x: x[1], reverse=True)
    return candidates
if __name__ == "__main__":
    print("Enter ciphertext:")
    ct = input().rstrip('\n')
    ranked = ranked_candidates(ct)
    print("\nTop 6 candidate plaintexts (shift, score):\n")
```

```
    for k, sc, cand in ranked[:6]:
        print(f"Shift {k:2d} | Score {sc:6.2f} | {cand}")
    print("\n(You can inspect others if needed.)")
```

```
Output
Enter ciphertext:
Hello

Top 6 candidate plaintexts (shift, score):

Shift  7 | Score  38.45 | Axeeh
Shift  0 | Score  33.60 | Hello
Shift 11 | Score  32.20 | Wtaad
Shift  3 | Score  31.50 | Ebiil
Shift 23 | Score  27.90 | Khoor
Shift  4 | Score  25.50 | Dahhk

(You can inspect others if needed.)

=== Code Execution Successful ===
```

Conclusion

- The Caesar cipher is easily broken by brute-force because its keyspace is only 26 keys.
- Automated scoring based on English letter frequency helps pick the correct plaintext without manual inspection.
- Practical demonstrates the need for modern ciphers with large keyspaces and stronger cryptographic designs.