CNS_1.cpp

```c
#include <stdio.h>
#include <ctype.h>

void caesarCipher(char *text, int k) {
    char ch;
    for (int i = 0; text[i] != '\0'; ++i) {
        ch = text[i];

        if (isalpha(ch)) {
            char base = islower(ch) ? 'a' : 'A';
            text[i] = (ch - base + k) % 26 + base;
        }
    }
}

int main() {
    char text[100];
    int k;

    printf("Enter a message: ");
    fgets(text, sizeof(text), stdin);

    printf("Enter key (1-25): ");
    scanf("%d", &k);

    if (k < 1 || k > 25) {
        printf("Invalid key. Key must be between 1 and 25.\n");
        return 1;
    }

    caesarCipher(text, k);

    printf("Encrypted message: %s\n", text);
    return 0;
}
```

Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\chani\OneDrive\Documents\CNS_1.exe
- Output Size: 130.3212890625 KiB
- Compilation Time: 1.58s
```

C:\Users\chani\OneDrive\Doc

```
Enter a message: I am Chanikya
Enter key (1-25): 2
Encrypted message: K co Ejcpkmac


--------------------------------
Process exited after 32.41 seconds with return value 0
Press any key to continue . . .
```

CNS_1.cpp | CNS_2.cpp

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>

// Example substitution key (cipher alphabet)
char cipherMap[26] = {
    'Q','W','E','R','T','Y','U','I','O','P','A','S','D',
    'F','G','H','J','K','L','Z','X','C','V','B','N','M'
};

void monoalphabeticEncrypt(char *text) {
    for (int i = 0; text[i] != '\0'; ++i) {
        if (isalpha(text[i])) {
            char base = islower(text[i]) ? 'a' : 'A';
            int index = text[i] - base;
            char newChar = cipherMap[index];
            text[i] = islower(text[i]) ? tolower(newChar) : newChar;
        }
    }
}

int main() {
    char text[100];

    printf("Enter a message: ");
    fgets(text, sizeof(text), stdin);

    monoalphabeticEncrypt(text);

    printf("Encrypted message: %s\n", text);
    return 0;
}
```

Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\chani\OneDrive\Doc
- Output Size: 129.689453125 KiB
- Compilation Time: 0.22s
```

C:\Users\chani\OneDrive\Doc

```
Enter a message: I am Chanikya
Encrypted message: O qd Eiqfoanq


--------------------------------
Process exited after 28.04 seconds with return value 0
Press any key to continue . . .
```

CNS_1.cpp CNS_2.cpp CNS_3.cpp

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE];

void prepareMatrix(char key[]) {
    int used[26] = {0};
    used['J' - 'A'] = 1; // Treat J as I
    int k = 0;

    // Fill matrix with key
    for (int i = 0; key[i] != '\0'; ++i) {
        char c = toupper(key[i]);
        if (isalpha(c) && !used[c - 'A']) {
            matrix[k / SIZE][k % SIZE] = c;
            used[c - 'A'] = 1;
            ++k;
        }
    }

    // Fill remaining letters
    for (char c = 'A'; c <= 'Z'; ++c) {
        if (!used[c - 'A']) {
            matrix[k / SIZE][k % SIZE] = c;
            ++k;
        }
    }
}

void findPosition(char c, int *row, int *col)
    if (c == 'J') c = 'I'; // Treat J as I
    for (int i = 0; i < SIZE; ++i) {
        for (int j = 0; j < SIZE; ++j) {
            if (matrix[i][j] == c) {
                *row = i;
                *col = j;
                return;
```

Compile Log Debug Find Results Clos

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\chani\OneDriv
- Output Size: 131.921875 KiB
- Compilation Time: 0.22s
```

C:\Users\chani\OneDrive\Doc

```
Enter keyword: Challenge
Enter plaintext (no spaces or special characters): Iamthekingofmyownkingdom
Encrypted text: MCOSACMKGBPDOXKYGIQIBFPO

--------------------------------
Process exited after 40.8 seconds with return value 0
Press any key to continue . . .
```

CNS_1.cpp CNS_2.cpp CNS_3.cpp CNS_4.cpp

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void vigenereEncrypt(char *plaintext, char *key) {
    int textLen = strlen(plaintext);
    int keyLen = strlen(key);
    char ch;
    int j = 0;

    for (int i = 0; i < textLen; i++) {
        ch = toupper(plaintext[i]);

        if (isalpha(ch)) {
            char k = toupper(key[j % keyLen]) - 'A';
            char encrypted = ((ch - 'A') + k) % 26 + 'A';
            printf("%c", encrypted);
            j++;
        } else {
            printf("%c", ch); // preserve non-letters
        }
    }
    printf("\n");
}

int main() {
    char plaintext[100], key[100];

    printf("Enter plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';

    printf("Enter key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';

    printf("Encrypted text: ");
    vigenereEncrypt(plaintext, key);

    return 0;
```

Compile Log Debug Find Results Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\chani\OneDrive\Docu
- Output Size: 130.330078125 KiB
- Compilation Time: 0.20s
```

C:\Users\chani\OneDrive\Doc

```
Enter plaintext: Chanikya is my name
Enter key: Chani
Encrypted text: EOAAQMFA VA OF NNUG

--------------------------------
Process exited after 23.44 seconds with return value 0
Press any key to continue . . .
```

CNS_1.cpp CNS_2.cpp CNS_3.cpp CNS_4.cpp CNS_5.cpp

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

void affineEncrypt(char *plaintext, int a, int b) {
    if (gcd(a, 26) != 1) {
        printf("Invalid key 'a'. It must be coprime with 26.\n");
        return;
    }

    for (int i = 0; plaintext[i] != '\0'; ++i) {
        char ch = toupper(plaintext[i]);
        if (isalpha(ch)) {
            int p = ch - 'A';
            int c = (a * p + b) % 26;
            printf("%c", c + 'A');
        } else {
            printf("%c", ch); // keep spaces/punctuation
        }
    }
    printf("\n");
}

int main() {
    char plaintext[100];
    int a, b;

    printf("Enter plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
```

Compile Log Debug Find Results Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\chani\OneDrive\Docum
- Output Size: 131.1845703125 KiB
- Compilation Time: 0.22s
```

C:\Users\chani\OneDrive\Doc

```
Enter plaintext: Chanikya is my name
Enter key a (must be coprime with 26): 7
Enter key b (any number 0û25): 9
Encrypted text: XGJWNBVJ NF PV WJPL

--------------------------------
Process exited after 83.22 seconds with return value 0
Press any key to continue . . .
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_7.py > ...

```python
from collections import Counter

# Step 1: Ciphertext input
ciphertext = '''53‡‡†305))6*;4826)4‡.)4‡);806*;48†8¶60))85;;]8*;:‡*8†83
(88)5*†;46(;88*96*?;8)*‡(;485);5*†2:*‡(;4956*2(5*—4)8¶8*
;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806*81
(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;'''

# Step 2: Frequency analysis
symbols = [c for c in ciphertext if not c.isspace()]
frequency = Counter(symbols)

print("Symbol frequencies (top 10):")
for symbol, count in frequency.most_common(10):
    print(f"{symbol}: {count}")

# Step 3: Manual substitution map (based on known decryption)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_7.py"
Symbol frequencies (top 10):
8: 34
;: 27
4: 19
): 16
‡: 15
*: 14
5: 12
6: 11
(: 9
†: 8

Decrypted text:
hteehtrheeandousaeoeceoeedurandouhuiareeuhddlundbenuhut
luuehnhdoalduunmangduenеldouhedhnhsbneldomhanslhnpoeuiun
doramsuhedeahueoeedylemdouruydubueydouhuhdoeouhhhsuuranuy
lemdoudluudolegtodoueoedyaydbyuudegd
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_8.py > ...

```python
def generate_cipher_alphabet(keyword):
    keyword = keyword.upper()
    seen = set()
    cipher_alpha = []

    # Add keyword letters (no duplicates)
    for char in keyword:
        if char not in seen and char.isalpha():
            seen.add(char)
            cipher_alpha.append(char)

    # Add remaining letters
    for char in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
        if char not in seen:
            cipher_alpha.append(char)

    return cipher_alpha

def encrypt(plaintext, cipher_alpha):
    plaintext = plaintext.upper()
    encrypted = ''
    for char in plaintext:
        if char.isalpha():
            encrypted += cipher_alpha[ord(char) - ord('A')]
        else:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_8.py"
Cipher Alphabet Mapping:
Plain : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Cipher: C I P H E R A B D F G J K L M N O Q S T U V W X Y Z

Sample Encryption:
Plaintext : ATTACK AT DAWN
Ciphertext: CTTCPG CT HCWL

Decryption:
Decrypted : ATTACK AT DAWN
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_9.py > ...

```python
def create_playfair_matrix(keyword):
    keyword = keyword.upper().replace("J", "I")
    matrix = []
    used = set()

    for char in keyword:
        if char not in used and char.isalpha():
            matrix.append(char)
            used.add(char)

    for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":
        if char not in used:
            matrix.append(char)
            used.add(char)

    return [matrix[i*5:(i+1)*5] for i in range(5)]

def find_position(matrix, char):
    for row in range(5):
        for col in range(5):
            if matrix[row][col] == char:
                return row, col
    return None, None

def decrypt_pair(matrix, a, b):
    row1, col1 = find_position(matrix, a)
    row2, col2 = find_position(matrix, b)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_9.py"
Decrypted Message:
IZGKC WMKCI XVFCG UNDLZ CFHGT IMKCF FNHGG XSSLZ MPITH DXBFV HALKK XMOSZ YHYAQ KMOGC MLXVN KBTMO ISHAW CZNCF AXOBC WLIYT
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_10.py > ...

```python
def create_fixed_matrix():
    return [
        ['M', 'F', 'H', 'I', 'K'],
        ['U', 'N', 'O', 'P', 'Q'],
        ['Z', 'V', 'W', 'X', 'Y'],
        ['E', 'L', 'A', 'R', 'G'],
        ['D', 'S', 'T', 'B', 'C']
    ]

def find_position(matrix, char):
    for row in range(5):
        for col in range(5):
            if matrix[row][col] == char:
                return row, col
    return None, None

def prepare_text(text):
    text = text.upper().replace("J", "I")
    text = ''.join(filter(str.isalpha, text))
    result = ''
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if (i+1) < len(text) else 'X'
        if a == b:
            result += a + 'X'
            i += 1
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

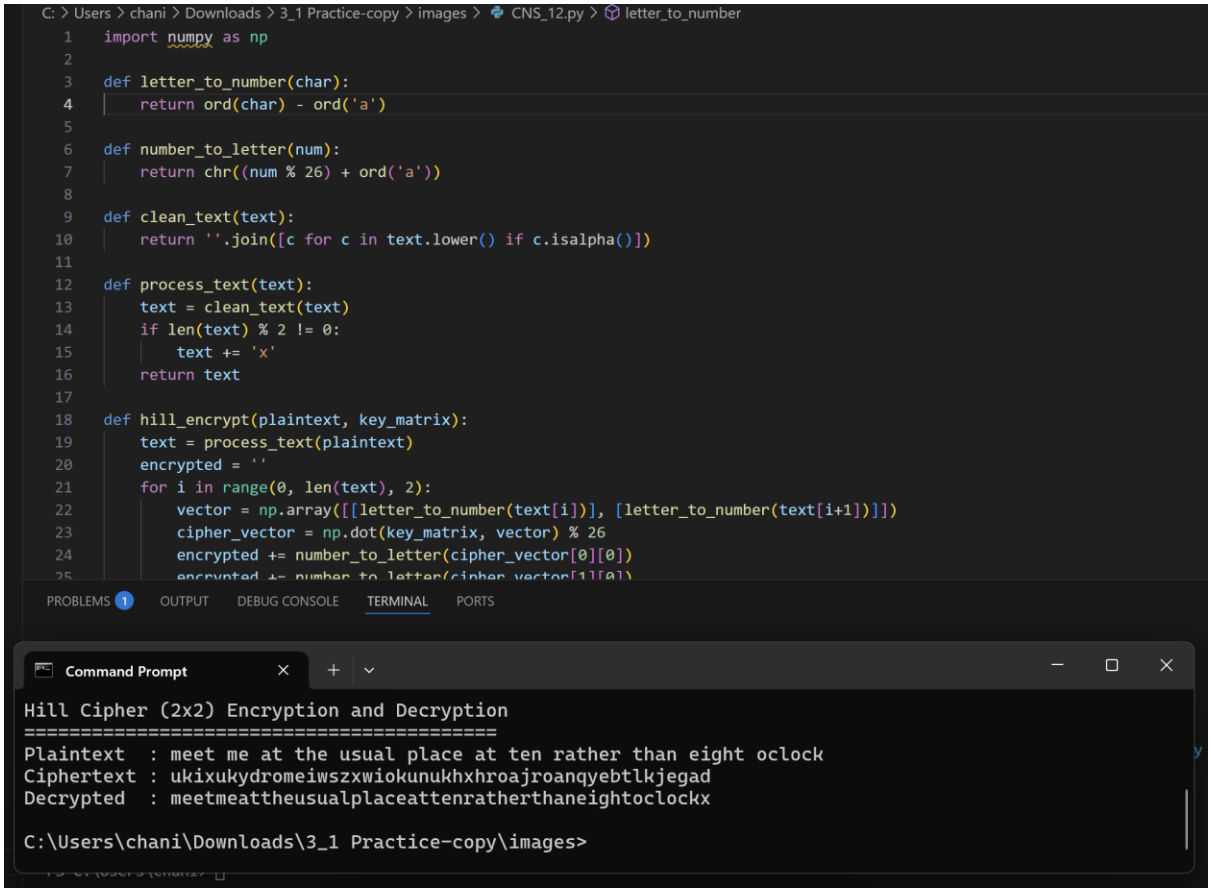
```
/images/CNS_10.py"
Playfair Cipher Encryption
==========================
Plaintext :  Must see you over Cadogan West. Coming at once.
Ciphertext:  UZTBDLGZPNNWLGTGTUEROVLDBDUHFPERHWQSRZ
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_11.py > ...

```python
import math

def factorial_log2(n):
    return math.log2(math.factorial(n))

def estimate_effective_keys(total_keys_log2, reduction_factor=10):
    return total_keys_log2 - math.log2(reduction_factor)

# Step 1: Total keys (25!)
total_keys_log2 = factorial_log2(25)

# Step 2: Approximate effectively unique keys (~10x fewer)
effective_keys_log2 = estimate_effective_keys(total_keys_log2)

# Display results
print("Playfair Cipher Key Analysis")
print("============================")
print(f"Total number of possible keys (25!): ≈ 2^{total_keys_log2:.2f}")
print(f"Effectively unique keys (considering equivalence): ≈ 2^{effective_keys_log2:.2f}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/images/CNS_11.py"
Playfair Cipher Key Analysis
============================
Total number of possible keys (25!): ≈ 2^83.68
Effectively unique keys (considering equivalence): ≈ 2^80.36
PS C:\Users\chani>
```


C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_12.py > letter_to_number
import numpy as np
def letter_to_number(char):
return ord(char) - ord('a')
def number_to_letter(num):
return chr((num % 26) + ord('a'))
def clean_text(text):
return ''.join([c for c in text.lower() if c.isalpha()])
def process_text(text):
text = clean_text(text)
if len(text) % 2 != 0:
text += 'x'
return text
def hill_encrypt(plaintext, key_matrix):
text = process_text(plaintext)
encrypted = ''
for i in range(0, len(text), 2):
vector = np.array([[letter_to_number(text[i])], [letter_to_number(text[i+1])]])
cipher_vector = np.dot(key_matrix, vector) % 26
encrypted += number_to_letter(cipher_vector[0][0])
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Command Prompt
Hill Cipher (2x2) Encryption and Decryption
Plaintext : meet me at the usual place at ten rather than eight oclock
Ciphertext : ukixukydromeiwszxwiokunukhxhroajroanqyebtlkjegad
Decrypted : meetmeattheusualplaceattenratherthaneightoclockx
C:\Users\chani\Downloads\3_1 Practice-copy\images>

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_13.py > ...

```python
import numpy as np

def letter_to_number(char):
    return ord(char) - ord('a')

def number_to_letter(num):
    return chr((num % 26) + ord('a'))

def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    raise ValueError("No modular inverse exists.")

def matrix_mod_inv(matrix, mod):
    det = int(round(np.linalg.det(matrix))) % mod
    det_inv = mod_inverse(det, mod)

    # Adjugate matrix
    adj = np.array([
        [ matrix[1][1], -matrix[0][1]],
        [-matrix[1][0],  matrix[0][0]]
    ]) % mod
```

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_13.py
Known Plaintext Attack on Hill Cipher
====================================
Plaintext Matrix:
[[ 7 11]
 [ 4 15]]
Ciphertext Matrix:
[[ 2 12]
 [25 15]]
Recovered Key Matrix K:
[[24  4]
 [ 9 10]]
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_14.py > ...

```python
import random

def clean_text(text):
    return ''.join([c for c in text.lower() if c.isalpha()])

def letter_to_number(c):
    return ord(c) - ord('a')

def number_to_letter(n):
    return chr((n % 26) + ord('a'))

def generate_key(length):
    return [random.randint(0, 25) for _ in range(length)]

def encrypt_otp_vigenere(plaintext, key):
    cipher = ''
    for i in range(len(plaintext)):
        shift = (letter_to_number(plaintext[i]) + key[i]) % 26
        cipher += number_to_letter(shift)
    return cipher

def decrypt_otp_vigenere(ciphertext, key):
    plaintext = ''
    for i in range(len(ciphertext)):
        shift = (letter_to_number(ciphertext[i]) - key[i] + 26) % 26
        plaintext += number_to_letter(shift)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_14.py"
One-Time Pad Vigenère Cipher
============================
Plaintext : meetmeattheusualplaceattenratherthaneightoclock
Key       : [2, 1, 23, 21, 14, 20, 18, 25, 14, 22, 15, 0, 25, 4, 20, 16, 19, 4, 23, 24, 17, 20, 25, 19, 12, 4, 8, 24, 10, 2, 21, 7
, 14, 4, 22, 11, 12, 23, 15, 14, 14, 18, 18, 10, 18, 13, 24]
Ciphertext: ofboaysshdturyubipxavusmqrzydjzyhlwyqfvvhguvgpi
Decrypted : meetmeattheusualplaceattenratherthaneightoclock
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_15.py > ...

```python
from collections import Counter

# Common English letters in frequency order
english_freq_order = 'etaoinshrdlcumwfgypbvkjxqz'

def letter_to_number(c):
    return ord(c) - ord('a')

def number_to_letter(n):
    return chr((n % 26) + ord('a'))

def decrypt_caesar(ciphertext, key):
    return ''.join(number_to_letter(letter_to_number(c) - key) if c.isalpha() else c for c in ciphertext)

def frequency_attack(ciphertext, top_n=10):
    ciphertext = ciphertext.lower()
    letters_only = ''.join(c for c in ciphertext if c.isalpha())

    freq_counter = Counter(letters_only)
    most_common_in_cipher = [item[0] for item in freq_counter.most_common(5)]

    guesses = []
    for cipher_letter in most_common_in_cipher:
        for english_letter in english_freq_order[:5]:  # Try mapping to top 5 English letters
            key = (letter_to_number(cipher_letter) - letter_to_number(english_letter)) % 26
            plaintext = decrypt_caesar(ciphertext, key)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_15.py"
Letter Frequency Attack on Additive Cipher
========================================
Ciphertext: ymnx nx f yjxy
1. Key = 20 | Plaintext: estd td l epde
2. Key =  5 | Plaintext: this is a test
3. Key = 24 | Plaintext: aopz pz h alza
4. Key = 10 | Plaintext: ocdn dn v ozno
5. Key = 16 | Plaintext: iwxh xh p ithi
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_16.py > ...

```python
import string
from collections import Counter

# English letter frequencies (approximate frequencies in English language)
ENGLISH_FREQ = {
    'E': 12.02, 'T': 9.10, 'A': 8.12, 'O': 7.68, 'I': 7.31, 'N': 6.95, 'S': 6.28, 'R': 6.02,
    'H': 5.92, 'D': 4.32, 'L': 3.98, 'U': 2.88, 'C': 2.71, 'M': 2.61, 'F': 2.30, 'Y': 2.11,
    'P': 2.03, 'B': 1.49, 'V': 1.11, 'K': 0.69, 'J': 0.49, 'X': 0.17, 'Q': 0.11, 'Z': 0.07
}

def clean_text(text):
    """Clean the text by removing non-alphabetic characters and converting to uppercase."""
    text = text.upper()
    return ''.join(filter(lambda x: x in string.ascii_uppercase, text))

def letter_frequency(text):
    """Calculate letter frequency in the ciphertext."""
    letter_counts = Counter(text)
    total_letters = sum(letter_counts.values())
    frequencies = {char: (count / total_letters) * 100 for char, count in letter_counts.items()}
    return frequencies

def sort_by_frequency(freqs):
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_16.py"
Enter the ciphertext: NYFTVVTCSUS
Enter the number of possible plaintexts to show (e.g., 10): 3

Top possible plaintexts:
1. OINETTESARA
2. OINETTESARA
3. OINETTESARA
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_17.py > ...

```
33  def des_decrypt(ciphertext, key):
48
49      decrypted = permute(R + L, FP)
50      return decrypted
51
52  # Convert binary string to ASCII
53  def bin_to_text(b):
54      return ''.join(chr(int(b[i:i + 8], 2)) for i in range(0, len(b), 8))
55
56  # Driver
57  cipher_bin = '0110110001101111011101100110010101101110011001110110001001101001'  # Example 64-bit binary
58  key_bin = '0001001100110100010101110111100110011011101111001101111111110001'     # Example key
59
60  print("===== DES Decryption (Simplified) =====")
61  plaintext_bin = des_decrypt(cipher_bin, key_bin)
62  print("Decrypted Binary:", plaintext_bin)
63  print("Decrypted Text  :", bin_to_text(plaintext_bin))
64
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
                                ^
SyntaxError: expected ':'
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_17.py"
===== DES Decryption (Simplified) =====
Decrypted Binary: 1001110111111000100101100011011011101001100111001111011111101100
Decrypted Text  : ø6ôÎ{ì
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_18.py > ...

```python
def left_shift(bits, n):
    return bits[n:] + bits[:n]

def generate_subkeys(key_64bit):
    # PC-1 removes 8 parity bits to get 56-bit key
    PC1 = [i for i in range(1, 65) if i % 8 != 0]

    # Simplified example shift schedule
    SHIFT_SCHEDULE = [1, 1, 2, 2, 2, 2, 2, 2,
                      1, 2, 2, 2, 2, 2, 2, 1]

    # Permute to 56-bit
    key_56 = ''.join(key_64bit[i - 1] for i in PC1)
    C, D = key_56[:28], key_56[28:]

    subkeys = []
    for i in range(16):
        C = left_shift(C, SHIFT_SCHEDULE[i])
        D = left_shift(D, SHIFT_SCHEDULE[i])
        K = C[:24] + D[:24]  # 24 bits from C and 24 bits from D
        subkeys.append(K)

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_18.py"
===== DES Subkeys (24 bits from C, 24 bits from D) =====
Round  1: 001001001101001010110111001101101111011011111111
Round  2: 010010011010010101101111011010111011011111111110
Round  3: 001001101001010110111100101101111011011111111000
Round  4: 100110100101011011111000011011110110111111100010
Round  5: 011010010101101111000001011110110111111110001001
Round  6: 101001010110111100000100111011011111111000100110
Round  7: 100101011011110000010010101101111111100010011011
Round  8: 010101101111000001001001110111111110001001101101
Round  9: 101011011110000010010011101111111100010011011011
Round 10: 101101111000001001001101111111110001001101101111
Round 11: 110111100000100100110100111111000100110110111101
Round 12: 011110000010010011010010111100010011011011110110
Round 13: 111000001001001101001010110001001101101111011011
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_19.py > ...

```python
from Crypto.Cipher import DES3
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

# Generate a 24-byte (192-bit) key for 3DES
key = DES3.adjust_key_parity(get_random_bytes(24))
iv = get_random_bytes(8)  # 3DES block size is 8 bytes

data = b"Sensitive information that needs encryption"
cipher = DES3.new(key, DES3.MODE_CBC, iv)
padded_data = pad(data, DES3.block_size)
ciphertext = cipher.encrypt(padded_data)

# Decryption
decipher = DES3.new(key, DES3.MODE_CBC, iv)
decrypted_data = unpad(decipher.decrypt(ciphertext), DES3.block_size)

print("Original:", data)
print("Encrypted:", ciphertext)
print("Decrypted:", decrypted_data)

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
ModuleNotFoundError: No module named 'Crypto'
```

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_19.py
Original: b'Sensitive information that needs encryption'
Encrypted: b'\x88{V)\xd0\xa58\xd2\xe3\xe8\xb6\xf9*\xfa\xa8\xc4\xefx\xd3\x10\x00\xc1\x9eGs{@\x17~\xe4\xb0\xb4\xb5d\x1
8\xe9W\x85\xf1\x80\xd5^\xb9\xe9cZ\xf2\r'
Decrypted: b'Sensitive information that needs encryption'
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_20.py > ...

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

key = get_random_bytes(16)
iv = get_random_bytes(16)
data = b"This is a test message for CBC mode."

# Encryption
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(pad(data, AES.block_size))

# Introduce error in first ciphertext block
corrupted_ciphertext = bytearray(ciphertext)
corrupted_ciphertext[0] ^= 0x01  # Flip a bit

# Decryption
decipher = AES.new(key, AES.MODE_CBC, iv)
try:
    decrypted_data = unpad(decipher.decrypt(bytes(corrupted_ciphertext)), AES.block_size)
    print("Decrypted Data:", decrypted_data)
except ValueError as e:
    print("Decryption error:", e)
```

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_20.py
Decrypted Data: b'\t*l\x8c\r\x8dSl\xac\xcd\x0b\x15\x7f\x07S&dssage for CBC mode.'

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_21.py > ...

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

data = b"Message requiring padding."
key = get_random_bytes(16)
iv = get_random_bytes(16)

# ECB Mode
cipher_ecb = AES.new(key, AES.MODE_ECB)
ciphertext_ecb = cipher_ecb.encrypt(pad(data, AES.block_size))
decrypted_ecb = unpad(cipher_ecb.decrypt(ciphertext_ecb), AES.block_size)

# CBC Mode
cipher_cbc = AES.new(key, AES.MODE_CBC, iv)
ciphertext_cbc = cipher_cbc.encrypt(pad(data, AES.block_size))
decipher_cbc = AES.new(key, AES.MODE_CBC, iv)
decrypted_cbc = unpad(decipher_cbc.decrypt(ciphertext_cbc), AES.block_size)

# CFB Mode (does not require padding)
cipher_cfb = AES.new(key, AES.MODE_CFB, iv)
ciphertext_cfb = cipher_cfb.encrypt(data)
decipher_cfb = AES.new(key, AES.MODE_CFB, iv)
decrypted_cfb = decipher_cfb.decrypt(ciphertext_cfb)

print("ECB Decrypted:", decrypted_ecb)
print("CBC Decrypted:", decrypted_cbc)
print("CFB Decrypted:", decrypted_cfb)

```

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_21.py
ECB Decrypted: b'Message requiring padding.'
CBC Decrypted: b'Message requiring padding.'
CFB Decrypted: b'Message requiring padding.'

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_22.py > ...

```python
def s_des_encrypt_block(block, key):
    # Placeholder for S-DES encryption logic
    return block  # Replace with actual encryption

def s_des_decrypt_block(block, key):
    # Placeholder for S-DES decryption logic
    return block  # Replace with actual decryption

def xor_bytes(a, b):
    return bytes([i ^ j for i, j in zip(a, b)])

key = b'\x3F'  # Example 8-bit key
iv = b'\xAA'   # Initialization vector: 10101010
plaintext = b'\x01\x23'  # Example plaintext

# Encryption
prev_cipher = iv
ciphertext = b''
for i in range(0, len(plaintext), 1):
    block = plaintext[i:i+1]
    block = xor_bytes(block, prev_cipher)
    cipher_block = s_des_encrypt_block(block, key)
    ciphertext += cipher_block
    prev_cipher = cipher_block

# Decryption
prev_cipher = iv
decrypted = b''
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_22.py"
Ciphertext: b'\xab\x88'
Decrypted: b'\x01#'
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_23.py > ...

```python
def s_des_encrypt_block(block, key):
    # Placeholder for S-DES encryption logic
    return block  # Replace with actual encryption

def xor_bytes(a, b):
    return bytes([i ^ j for i, j in zip(a, b)])

key = b'\x3F'  # Example 8-bit key
counter = 0
plaintext = b'\x01\x02\x04'  # Example plaintext
ciphertext = b''

for i in range(len(plaintext)):
    counter_block = counter.to_bytes(1, 'big')
    keystream = s_des_encrypt_block(counter_block, key)
    cipher_byte = xor_bytes(bytes([plaintext[i]]), keystream)
    ciphertext += cipher_byte
    counter += 1

# Decryption is identical to encryption in CTR mode
counter = 0
decrypted = b''
for i in range(len(ciphertext)):
    counter_block = counter.to_bytes(1, 'big')
    keystream = s_des_encrypt_block(counter_block, key)
    plain_byte = xor_bytes(bytes([ciphertext[i]]), keystream)
    decrypted += plain_byte
    counter += 1
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_23.py"
Ciphertext: b'\x01\x03\x06'
Decrypted: b'\x01\x02\x04'
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_24.py > ...

```python
    return gcd, x, y

def modinv(e, phi):
    gcd, x, _ = gcd_extended(e, phi)
    if gcd != 1:
        raise Exception('Modular inverse does not exist')
    else:
        return x % phi

n = 3599
e = 31

# Factor n to find p and q
for i in range(2, n):
    if n % i == 0:
        p = i
        q = n // i
        break

phi = (p - 1) * (q - 1)
d = modinv(e, phi)

print(f"p = {p}, q = {q}")
print(f"Private key d = {d}")
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy
/images/CNS_24.py"
p = 59, q = 61
Private key d = 3031
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_25.py > ...

```python
import math

n = 3233  # Example modulus
e = 17    # Public exponent
m = 61    # Known plaintext sharing a factor with n

gcd = math.gcd(m, n)
if 1 < gcd < n:
    p = gcd
    q = n // p
    print(f"Found factors: p = {p}, q = {q}")
    phi = (p - 1) * (q - 1)
    d = pow(e, -1, phi)
    print(f"Private key d = {d}")
else:
    print("No common factor found; RSA remains secure.")
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

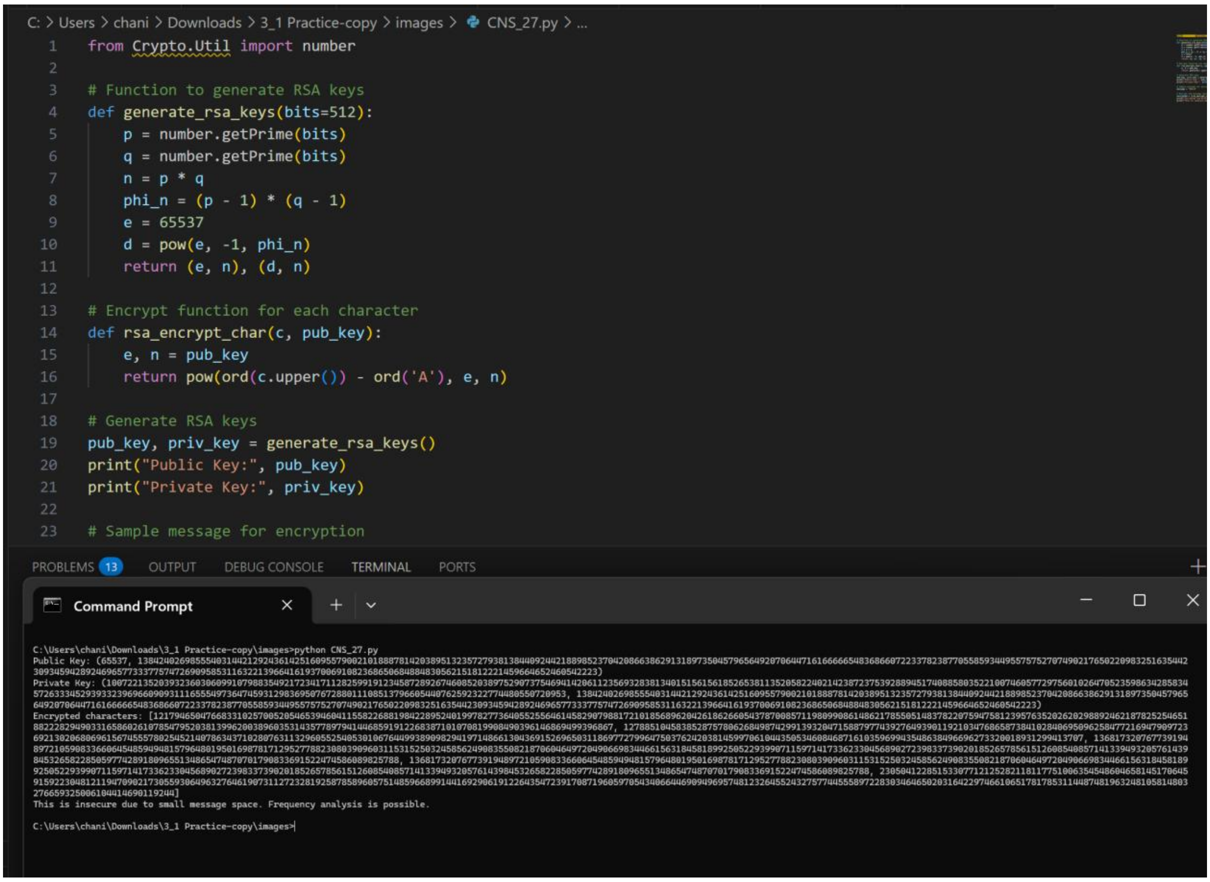
```
> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Do
/images/CNS_24.py"
p = 59, q = 61
Private key d = 3031
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Do
/images/CNS_25.py"
Found factors: p = 61, q = 53
Private key d = 2753
PS C:\Users\chani>
```

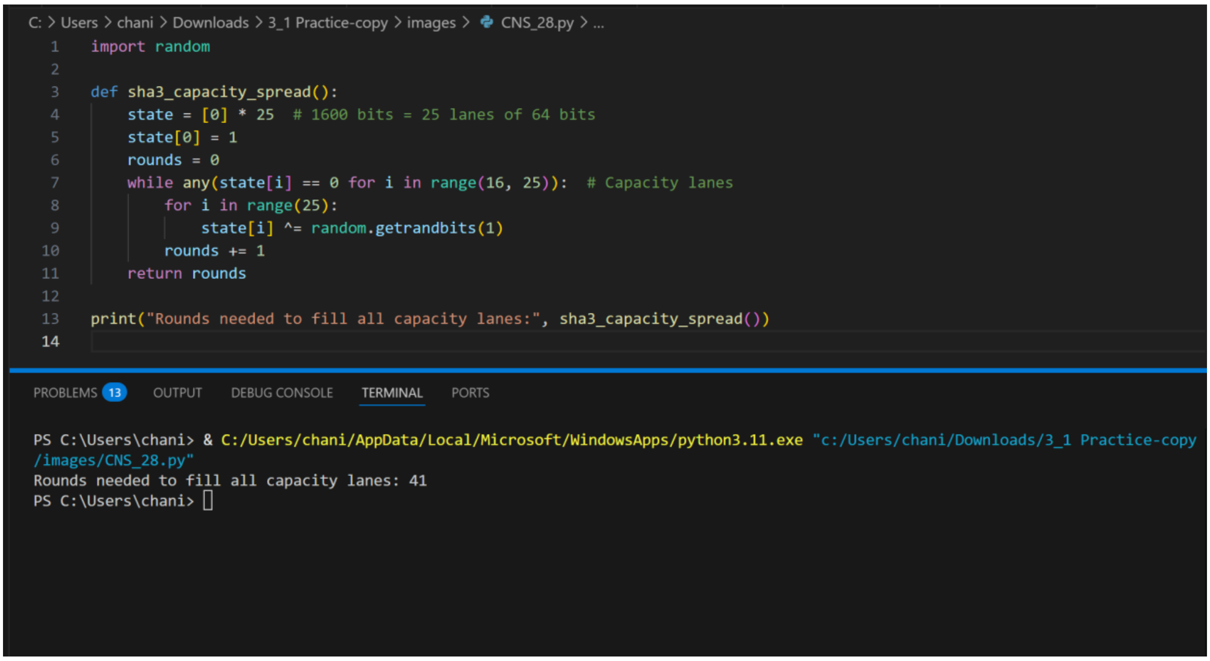
C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_26.py > ...

```python
from Crypto.Util import number

def generate_rsa_keys(bits=512):
    p = number.getPrime(bits)
    q = number.getPrime(bits)
    n = p * q
    phi_n = (p - 1) * (q - 1)
    e = 65537
    d = pow(e, -1, phi_n)
    return (e, n), (d, n)

# Unsafe to reuse 'n' after key leak
pub_key, priv_key = generate_rsa_keys()
print("Original Public Key:", pub_key)
print("Original Private Key:", priv_key)
print("If private key is leaked, 'n' must be regenerated!")
```

Command Prompt

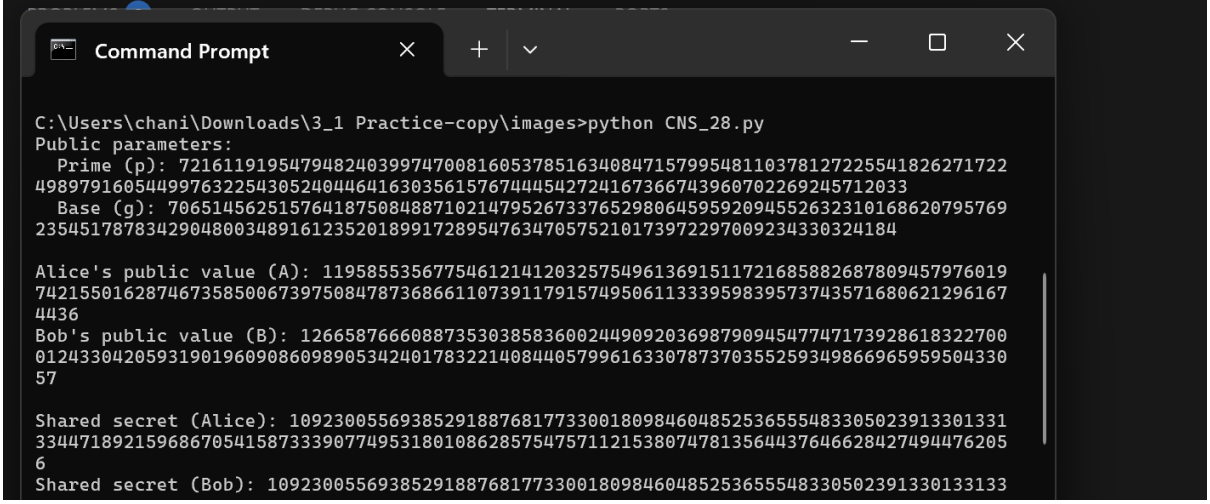
```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_26.py
Original Public Key: (65537, 76060097475044738050944091738374590625499856589657241759111316731386874136856758371340408015656216621609506649858482372
74533074679988674464968478660521070142465441252975150112869795473125654471542802475965947332537388135559415570868225604005979913119158936327401788
17281351258260910248298822831415623118143)
Original Private Key: (16129563982302695720463569998320491638512550267529126233549431312583956799274230249246822262257810070755892448841619055135489
97847055596040620289552831557870975132797142850508491531057486593533155524219245150283831940265008164277009700247838079360704263772298871234537309883
95978222516089432031313913268521153, 76060097475044738050944091738374590625499856589657241759111316731386874136856758371340408015656216621609506649858
48237274533074679988674464968478660521070142465441252975150112869795473125654471542802475965947332537388135559415570868225604005979913119158936327
40888673172813512582609102482988228314156231181143)
If private key is leaked, 'n' must be regenerated!

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```


C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_27.py > ...
from Crypto.Util import number
# Function to generate RSA keys
def generate_rsa_keys(bits=512):
p = number.getPrime(bits)
q = number.getPrime(bits)
n = p * q
phi_n = (p - 1) * (q - 1)
e = 65537
d = pow(e, -1, phi_n)
return (e, n), (d, n)
# Encrypt function for each character
def rsa_encrypt_char(c, pub_key):
e, n = pub_key
return pow(ord(c.upper()) - ord('A'), e, n)
Generate RSA keys
pub_key, priv_key = generate_rsa_keys()
print("Public Key:", pub_key)
print("Private Key:", priv_key)
# Sample message for encryption
PROBLEMS 13
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Command Prompt
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_27.py
This is insecure due to small message space. Frequency analysis is possible.
C:\Users\chani\Downloads\3_1 Practice-copy\images>


C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_28.py > ...
import random
def sha3_capacity_spread():
state = [0] * 25 # 1600 bits = 25 lanes of 64 bits
state[0] = 1
rounds = 0
while any(state[i] == 0 for i in range(16, 25)): # Capacity lanes
for i in range(25):
state[i] ^= random.getrandbits(1)
rounds += 1
return rounds
print("Rounds needed to fill all capacity lanes:", sha3_capacity_spread())
PROBLEMS 13
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3_1 Practice-copy/images/CNS_28.py"
Rounds needed to fill all capacity lanes: 41
PS C:\Users\chani>

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_28.py > ...

```python
from Crypto.Util import number
import random

# Generate large prime and primitive root
p = number.getPrime(512)  # large prime
g = random.randint(2, p - 2)  # base

print("Public parameters:\n  Prime (p):", p, "\n  Base (g):", g)

# Alice's private key and public value
a = random.randint(2, p - 2)
A = pow(g, a, p)

# Bob's private key and public value
b = random.randint(2, p - 2)
B = pow(g, b, p)

# Shared secrets
shared_secret_alice = pow(B, a, p)
shared_secret_bob = pow(A, b, p)

print("\nAlice's public value (A):", A)
```



C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_30.py > ...

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def xor_bytes(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

def cbc_mac(key, message):
    cipher = AES.new(key, AES.MODE_ECB)
    block_size = 16
    blocks = [message[i:i+block_size] for i in range(0, len(message), block_size)]
    iv = bytes([0] * block_size)
    for block in blocks:
        iv = cipher.encrypt(xor_bytes(block, iv))
    return iv

key = get_random_bytes(16)
X = b'TestMessageBlock'  # 16-byte block
T = cbc_mac(key, X)
X2 = xor_bytes(X, T)
T2 = cbc_mac(key, X + X2)

print("T (CBC-MAC of X):", T.hex())
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Rounds needed to fill all capacity lanes: 683

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_30.py
T (CBC-MAC of X): 1ebd9f0bb8f3f077ad8da1b6056b5b8a
T2 (CBC-MAC of X || (X XOR T)): 1ebd9f0bb8f3f077ad8da1b6056b5b8a
T == T2? (Collision Found): True

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_31.py > ...

```python
from Crypto.Cipher import AES, DES

def left_shift_one_bit(input_bytes):
    shifted = int.from_bytes(input_bytes, byteorder='big') << 1
    shifted &= (1 << (len(input_bytes) * 8)) - 1  # Trim to block size
    return shifted.to_bytes(len(input_bytes), byteorder='big')

def generate_cmac_subkeys(key_size_bits=128):
    if key_size_bits == 128:
        block_size = 16
        Rb = 0x87
        cipher = AES.new(bytes([0] * block_size), AES.MODE_ECB)
    elif key_size_bits == 64:
        block_size = 8
        Rb = 0x1B
        cipher = DES.new(bytes([0] * block_size), DES.MODE_ECB)
    else:
        raise ValueError("Unsupported block size")

    zero_block = bytes([0] * block_size)
    L = cipher.encrypt(zero_block)

    # Generate K1
    if (L[0] & 0x80):
        K1 = left_shift_one_bit(L)
        K1 = bytearray(K1)
        K1[-1] ^= Rb
```

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_31.py
CMAC Subkeys for AES (128-bit): {'L': '66e94bd4ef8a2c3b884cfa59ca342b2e', 'K1': 'cdd297a9df1458771099f4b39468565c', 'K2': '9ba52f53be28b0ee2133e96728d0ac3f'}
CMAC Subkeys for DES (64-bit): {'L': '8ca64de9c1b123a7', 'K1': '194c9bd383624755', 'K2': '329937a706c48eaa'}

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_32.py > ...

```python
# Message to sign
message = b"This is a critical message."

# First signature
hash1 = SHA256.new(message)
signer1 = DSS.new(key, 'fips-186-3')
signature1 = signer1.sign(hash1)

# Second signature (same message)
hash2 = SHA256.new(message)
signer2 = DSS.new(key, 'fips-186-3')  # New signer = new k internally
signature2 = signer2.sign(hash2)

# Output results
print("Signature 1:", signature1.hex())
print("Signature 2:", signature2.hex())

# Check if they are different
if signature1 != signature2:
    print("\n✔  DSA signatures differ even for the same message (because of different random k).")
else:
    print("\n❌  Signatures are identical, something is wrong.")

```

Command Prompt

```
C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_32.py
Signature 1: 55b0b9f82256d568c8f1b8f7a3067b3eeb963fc51f4e3237bb05d78e5b1886133cb2276a4bf91441
Signature 2: 132c0878b564d46fa87c42834a63814055f59c8287b20982178df243056248 2d3701aa1dc6aed71f

✔  DSA signatures differ even for the same message (because of different random k).

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_33.py > ...

```python
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

# 8-byte (64-bit) key for DES (only 56 bits are used, last bit of each byte is for parity)
key = get_random_bytes(8)

# DES requires 8-byte blocks
data = b"NetworkSecurity123"  # Must be padded to multiple of 8 bytes

# Encrypt
cipher = DES.new(key, DES.MODE_ECB)
ciphertext = cipher.encrypt(pad(data, DES.block_size))

# Decrypt
decipher = DES.new(key, DES.MODE_ECB)
plaintext = unpad(decipher.decrypt(ciphertext), DES.block_size)

# Results
print("Original Data:", data)
print("Encrypted Data (hex):", ciphertext.hex())
print("Decrypted Data:", plaintext)

```

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Command Prompt

```
DSA signatures differ even for the same message (because of different random k).

C:\Users\chani\Downloads\3_1 Practice-copy\images>python CNS_33.py
Original Data: b'NetworkSecurity123'
Encrypted Data (hex): b67b53d6db1417e91aa45c90bc6d7b7c89b6b96ccc731287
Decrypted Data: b'NetworkSecurity123'

C:\Users\chani\Downloads\3_1 Practice-copy\images>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_34.py > ...

```python
def process_mode(mode_name, mode, key, data, iv=None):
    else:
        decipher = DES.new(key, mode)

    decrypted = unpad(decipher.decrypt(encrypted), DES.block_size)

    print(f"\n--- {mode_name} ---")
    print("Encrypted (hex):", encrypted.hex())
    print("Decrypted:", decrypted)

# 8-byte key and sample plaintext
key = get_random_bytes(8)
data = b"HelloDESmodeTest"

# ECB Mode
process_mode("ECB", DES.MODE_ECB, key, data)

# CBC Mode with IV
iv_cbc = get_random_bytes(8)
process_mode("CBC", DES.MODE_CBC, key, data, iv=iv_cbc)

# CFB Mode with IV
iv_cfb = get_random_bytes(8)
process_mode("CFB", DES.MODE_CFB, key, data, iv=iv_cfb)

```

Command Prompt

```
Encrypted (hex): 9ac1ece713acfadaf1cbf0f183c1f692b556bc164b117461
Decrypted: b'HelloDESmodeTest'

--- CFB ---
Encrypted (hex): 0309d77b52523532e5799933b634d34d0b2c8e69c9f59e77
Decrypted: b'HelloDESmodeTest'

C:\Users\chani\Downloads\3_1 Practice-copy\images>=
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_35.py > ...

```python
import random
import string

def generate_key(length):
    return [random.randint(0, 25) for _ in range(length)]

def encrypt_vigenere_otp(plaintext, key):
    plaintext = plaintext.upper().replace(" ", "")
    ciphertext = ""
    for i, char in enumerate(plaintext):
        if char in string.ascii_uppercase:
            shift = key[i]
            encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
            ciphertext += encrypted_char
    return ciphertext

def decrypt_vigenere_otp(ciphertext, key):
    decrypted = ""
    for i, char in enumerate(ciphertext):
        if char in string.ascii_uppercase:
            shift = key[i]
            decrypted_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('A'))
            decrypted += decrypted_char
    return decrypted

```

PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
ModuleNotFoundError: No module named 'Crypto'
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/D
/images/CNS_35.py"
Plaintext: HELLOCRYPTO
Key:       [8, 2, 10, 17, 2, 9, 20, 0, 15, 5, 4]
Encrypted: PGVCQLLYEYS
Decrypted: HELLOCRYPTO
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_36.py > ...

```python
def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def affine_encrypt(text, a, b):
    return ''.join(
        chr(((a * (ord(char) - ord('A')) + b) % 26) + ord('A')) if char.isalpha() else char
        for char in text.upper()
    )

def affine_decrypt(cipher, a, b):
    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        raise ValueError("No modular inverse exists for a = {} under mod 26".format(a))
    return ''.join(
        chr(((a_inv * ((ord(char) - ord('A') - b)) % 26) + ord('A')) if char.isalpha() else char)
        for char in cipher.upper()
    )

# Example usage
plaintext = "HELLO"
a, b = 5, 8
ciphertext = affine_encrypt(plaintext, a, b)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Downloads/3
/images/CNS_36.py"
Plaintext: HELLO
Ciphertext: RCLLA
Decrypted: HELLO
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_37.py > ...

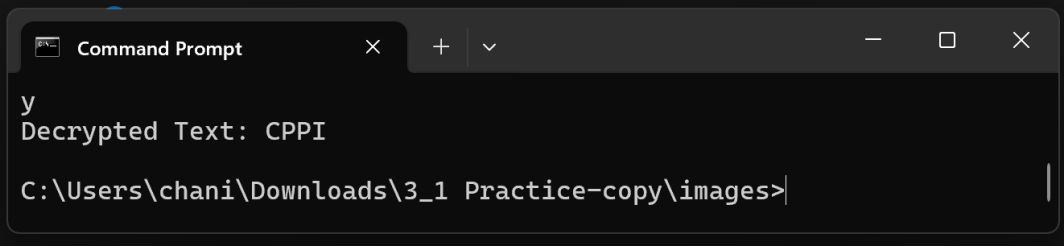
```python
from collections import Counter

# English letter frequency in descending order
english_freq_order = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'

def frequency_attack(ciphertext, top_n=10):
    ciphertext = ciphertext.upper()
    letter_counts = Counter(filter(str.isalpha, ciphertext))
    most_common = [pair[0] for pair in letter_counts.most_common()]

    probable_plaintexts = []

    # Try substitution with most common English letters
    for i in range(top_n):
        if i >= len(most_common):
            break

        mapping = {most_common[i]: english_freq_order[0]}  # Most frequent letter to 'E'
        translation = ''
        for c in ciphertext:
            if c in mapping:
                translation += mapping[c]
            elif c.isalpha():
                translation += '_'
            else:
                
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chani/Download
/images/CNS_37.py"
Guess 1: _E__E_ ____E_ _E__
Guess 2: E_____ E_____ ____
Guess 3: _____E _____E ____
Guess 4: __E___ ______ ____
Guess 5: ___E__ ______ ____
Guess 6: ______ _E____ ____
Guess 7: ______ __E___ ____
Guess 8: ______ ___E__ ____
Guess 9: ______ ______ E___
Guess 10: ______ ______ __E_
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_39.py > ...

```python
import numpy as np

def mod_inverse(a, m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def decrypt_hill(ciphertext, key_matrix):
    n = len(key_matrix)
    ciphertext = ciphertext.upper().replace(" ", "")
    cipher_numbers = [ord(c) - ord('A') for c in ciphertext]

    while len(cipher_numbers) % n != 0:
        cipher_numbers.append(23)  # Padding with 'X'

    cipher_matrix = np.array(cipher_numbers).reshape(-1, n).T

    det = int(round(np.linalg.det(key_matrix))) % 26
    det_inv = mod_inverse(det, 26)
    if det_inv is None:
        raise ValueError("Key matrix is not invertible under mod 26")

    adjugate = np.round(det * np.linalg.inv(key_matrix)).astype(int) % 26
    key_inv = (det_inv * adjugate) % 26

    decrypted_matrix = np.dot(key_inv, cipher_matrix) % 26
```



C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_39.py > ...

```python
from collections import Counter

def decrypt_additive(ciphertext, key):
    decrypted = ""
    for char in ciphertext:
        if char.isalpha():
            offset = ord('A') if char.isupper() else ord('a')
            decrypted += chr((ord(char) - offset - key) % 26 + offset)
        else:
            decrypted += char
    return decrypted

def additive_attack(ciphertext, top_n=10):
    # Assume most frequent letter in English is 'E'
    freq = Counter(c for c in ciphertext.upper() if c.isalpha())
    most_common = freq.most_common()

    guesses = []
    for char, _ in most_common:
        key = (ord(char) - ord('E')) % 26
        plaintext = decrypt_additive(ciphertext, key)
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Guess #8: Key = 3
WLIPP MW XLI TEVXY

Guess #9: Key = 20
FURYY VF GUR CNEGH

Guess #10: Key = 23
CROVV SC DRO ZKBDE
PS C:\Users\chani>
```

C: > Users > chani > Downloads > 3_1 Practice-copy > images > CNS_40.py > ...

```python
import string
from collections import Counter

def monoalphabetic_attack(ciphertext, top_n=10):
    english_freq_order = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'
    cipher_letters = [c for c in ciphertext.upper() if c.isalpha()]
    cipher_freq_order = [pair[0] for pair in Counter(cipher_letters).most_common()]

    mapping = dict(zip(cipher_freq_order, english_freq_order))

    def substitute(text, map_table):
        return ''.join(
            map_table.get(c.upper(), c) if c.isalpha() else c
            for c in text
        )

    plaintext = substitute(ciphertext, mapping)
    print(f"\nTop guess based on frequency:\n{plaintext}")

# Example usage
cipher = "GIEWIV GMTLIV HIQS"  # Example ciphertext
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chan
/images/CNS_40.py"

Top guess based on frequency:
TEOIEA TNSHEA REDL
PS C:\Users\chani>
```