DATE: 22-08-2025

Applied Cryptography (UE23CS342AA4)

Lab 02

NAME	Keerthan pv
SRN	PES2UG23CS272
SECTION	E

Question 1:

Create and display a file SRN.txt with the

following contents:

THE LEGEND OF KING ARTHUR TELLS OF A SWORD CALLED EXCALIBUR,
SAID TO POSSESS MAGICAL POWERS AND GRANT VICTORY TO ITS WIELDER.
BUT ONLY THE TRUE KING COULD REMOVE THE SWORD FROM THE STONE.

ACCORDING TO THE TALES, ARTHUR WAS A YOUNG BOY OF HUMBLE ORIGINS. HE APPROACHED THE STONE WITHOUT FEAR, WHILE NOBLE KNIGHTS AND RULERS HAD FAILED.
WITH A SIMPLE GRIP, HE PULLED THE SWORD OUT EFFORTLESSLY, PROVING HIMSELF TO BE THE RIGHTFUL KING.

EXCALIBUR BECAME A SYMBOL OF STRENGTH, JUSTICE, AND HONOR.
UNDER ARTHUR'S RULE, THE KINGDOM PROSPERED,
AND THE KNIGHTS OF THE ROUND TABLE FOUGHT BRAVELY FOR PEACE AND EQUALITY.

YET LEGENDS SPEAK OF TRAGEDY AS WELL.
FOR EVERY HEROIC TALE, THERE CAME A GREAT CHALLENGE.
THE BETRAYAL OF FRIENDS, THE LOSS OF TRUST,
AND THE FINAL BATTLE THAT WOULD DETERMINE THE FUTURE OF HIS KINGDOM.

STILL, THE STORY OF KING ARTHUR AND HIS SWORD LIVES ON,
A TIMELESS REMINDER THAT TRUE POWER LIES NOT IN MAGIC, BUT
IN COURAGE, WISDOM, AND HONOR.

```
[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272.txt
[08/22/25]seed@keerthanpv:~/aclab$ cat pes2ug23cs272.txt
THE LEGEND OF KING ARTHUR TELLS OF A SWORD CALLED EXCALIBUR, SAID
TO POSSESS MAGICAL POWERS AND GRANT VICTORY TO ITS WIELDER. BUT ON
LY THE TRUE KING COULD REMOVE THE SWORD FROM THE STONE.
ACCORDING TO THE TALES, ARTHUR WAS A YOUNG BOY OF HUMBLE ORIGINS.
HE APPROACHED THE STONE WITHOUT FEAR, WHILE NOBLE KNIGHTS AND RULE
RS HAD FAILED.
WITH A SIMPLE GRIP, HE PULLED THE SWORD OUT EFFORTLESSLY, PROVING
HIMSELF TO BE THE RIGHTFUL KING.
EXCALIBUR BECAME A SYMBOL OF STRENGTH, JUSTICE, AND HONOR.
UNDER ARTHUR'S RULE, THE KINGDOM PROSPERED,
AND THE KNIGHTS OF THE ROUND TABLE FOUGHT BRAVELY FOR PEACE AND EQ
UALITY.
YET LEGENDS SPEAK OF TRAGEDY AS WELL.
FOR EVERY HEROIC TALE, THERE CAME A GREAT CHALLENGE. THE BETRAYAL
OF FRIENDS, THE LOSS OF TRUST,
AND THE FINAL BATTLE THAT WOULD DETERMINE THE FUTURE OF HIS KINGDO
M.
STILL, THE STORY OF KING ARTHUR AND HIS SWORD LIVES ON, A TIMELESS
 REMINDER THAT TRUE POWER LIES NOT IN MAGIC, BUT IN COURAGE, WISDO
M, AND HONOR.
[08/22/25]seed@keerthanpv:~/aclab$
```

Question 2:

In SRN.txt, convert uppercase letters to lowercase and find the frequencies of the following words:

- **th**
- **he**
- **ar**
- **ing**
- **e**
- **or**

```
[08/22/25]seed@keerthanpv:~/aclab$ tr '[:upper:]' '[:lower:]' < pe
s2ug23cs272.txt | tee temp.txt | grep -o -E 'th|he|ar|ing|e|or' |
sort | uniq -c
      5 ar
     88 e
      4 he
      8 ing
     13 or
     26 th
[08/22/25]seed@keerthanpv:~/aclab$
```

Question 3:

Generate the substitution cipher key

Python's random module has a 'shuffle' functionality that lets us generate random permutations of a list. This has been used to generate the substitution cipher key from the alphabet. However, a key can be generated from online sources like random.org as well.

Print the key mapping clearly (e.g., a → q, b → x, ...).

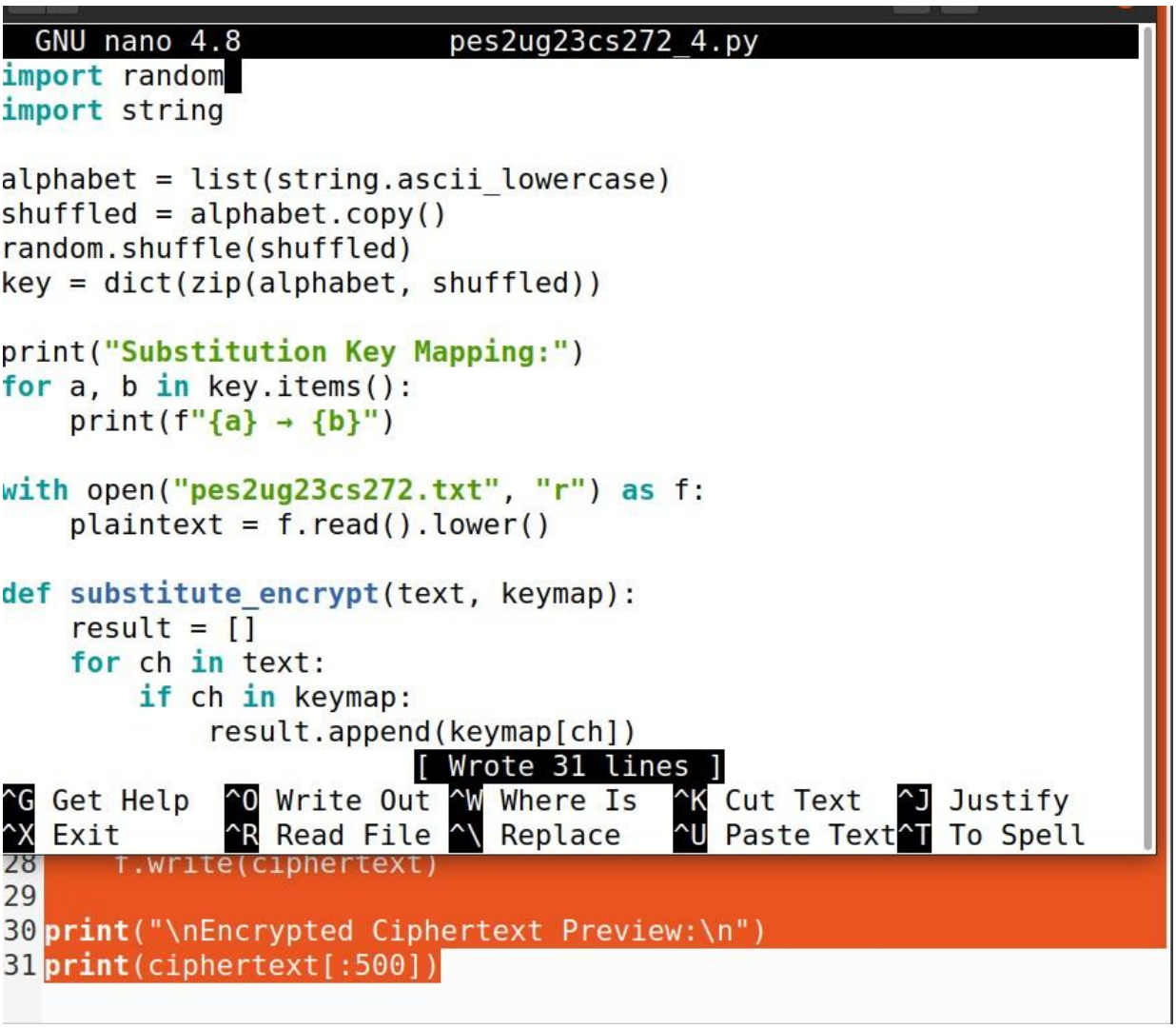
```
[08/22/25]seed@keerthanpv:~/aclab$ cat  pes2ug23cs272_3.py
import random
import string

alphabet = list(string.ascii_lowercase)
shuffled_alphabet = list(alphabet)
random.shuffle(shuffled_alphabet)

print("Plaintext -> Ciphertext")
for i in range(len(alphabet)):
    print(f"{alphabet[i]} -> {shuffled_alphabet[i]}")
[08/22/25]seed@keerthanpv:~/aclab$
```

```
[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272_3.py
[08/22/25]seed@keerthanpv:~/aclab$ python3 pes2ug23cs272_3.py
Plaintext -> Ciphertext
a -> b
b -> p
c -> s
d -> k
e -> v
f -> w
g -> l
h -> m
i -> d
j -> t
k -> h
l -> i
m -> c
n -> q
o -> u
p -> z
q -> g
r -> j
s -> o
t -> x
u -> e
v -> n
w -> y
x -> a
y -> f
z -> r
[08/22/25]seed@keerthanpv:~/aclab$
```

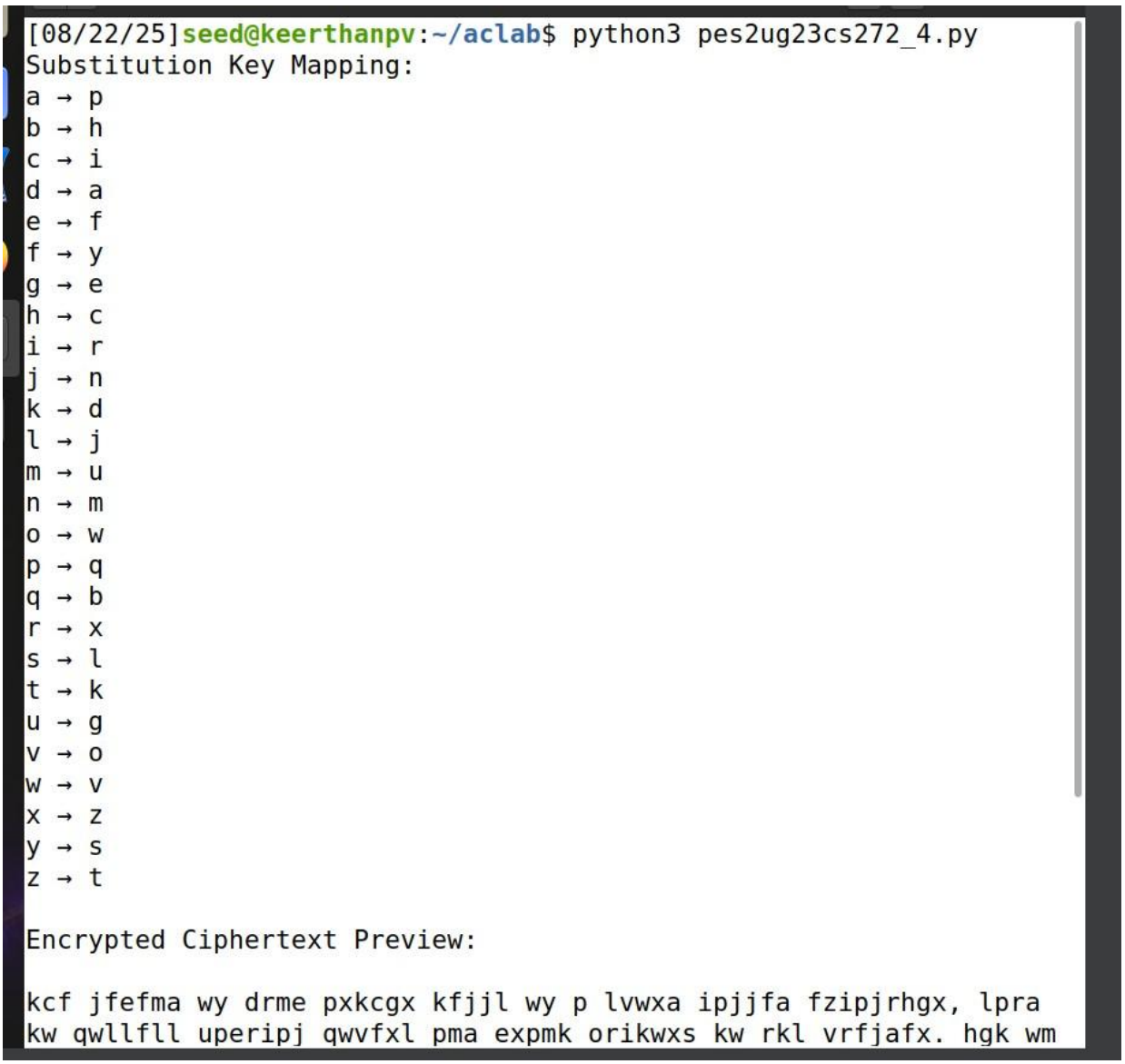
Question 4:

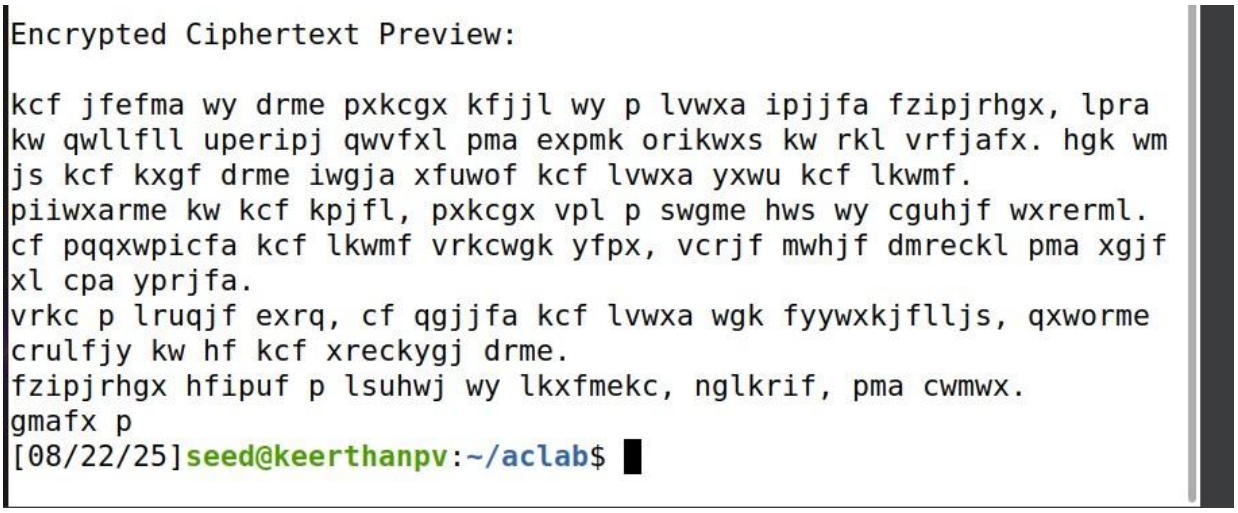
Generate the ciphertext using the key generated in question 4.

Write a python script to achieve it.


GNU nano 4.8 pes2ug23cs272_4.py
import random
import string
alphabet = list(string.ascii_lowercase)
shuffled = alphabet.copy()
random.shuffle(shuffled)
key = dict(zip(alphabet, shuffled))
print("Substitution Key Mapping:")
for a, b in key.items():
print(f"{a} → {b}")
with open("pes2ug23cs272.txt", "r") as f:
plaintext = f.read().lower()
def substitute_encrypt(text, keymap):
result = []
for ch in text:
if ch in keymap:
result.append(keymap[ch])
[Wrote 31 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell
f.write(ciphertext)
print("\nEncrypted Ciphertext Preview:\n")
print(ciphertext[:500])


GNU nano 4.8 pes2ug23cs272_4.py
shuffled = alphabet.copy()
random.shuffle(shuffled)
key = dict(zip(alphabet, shuffled))
print("Substitution Key Mapping:")
for a, b in key.items():
print(f"{a} → {b}")
with open("pes2ug23cs272.txt", "r") as f:
plaintext = f.read().lower()
def substitute_encrypt(text, keymap):
result = []
for ch in text:
if ch in keymap:
result.append(keymap[ch])
else:
result.append(ch)
return "".join(result)
ciphertext = substitute_encrypt(plaintext, key)
with open("PES2UG23CS272_cipher.txt", "w") as f:
f.write(ciphertext)
print("\nEncrypted Ciphertext Preview:\n")
print(ciphertext[:500])
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

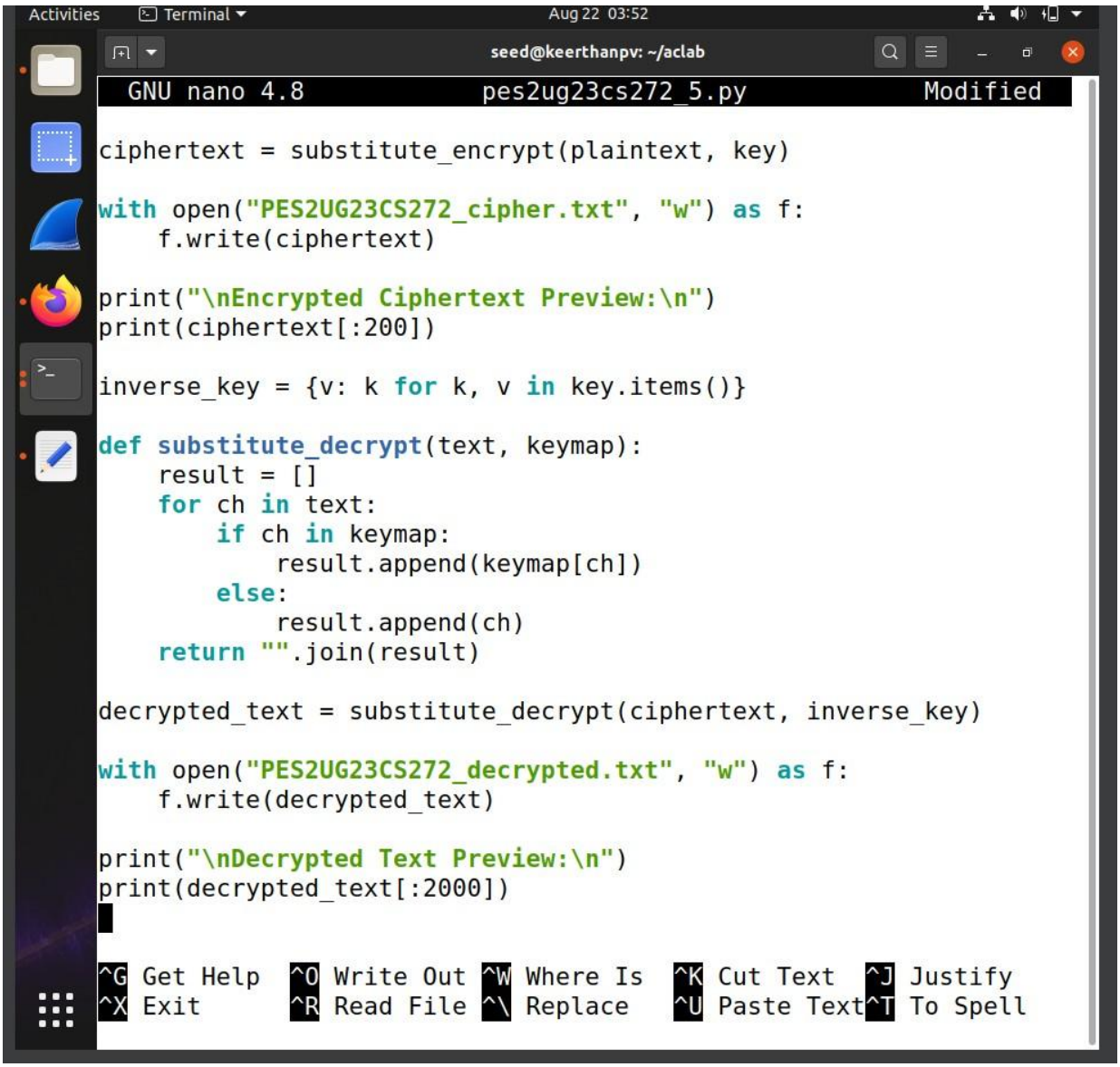




Question 5:

Decrypt the ciphertext back to plaintext.

```
 GNU nano 4.8              pes2ug23cs272_5.py              Modified
import random
import string

alphabet = list(string.ascii_lowercase)
shuffled = alphabet.copy()
random.shuffle(shuffled)
key = dict(zip(alphabet, shuffled))

print("Substitution Key Mapping:")
for a, b in key.items():
    print(f"{a} → {b}")

with open("pes2ug23cs272.txt", "r") as f:
    plaintext = f.read().lower()

def substitute_encrypt(text, keymap):
    result = []
    for ch in text:
        if ch in keymap:
            result.append(keymap[ch])
        else:
            result.append(ch)
    return "".join(result)

ciphertext = substitute_encrypt(plaintext, key)

with open("PES2UG23CS272_cipher.txt", "w") as f:
    f.write(ciphertext)

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell
```


Activities
Terminal
Aug 22 03:52
seed@keerthanpv: ~/aclab
GNU nano 4.8 pes2ug23cs272_5.py Modified
ciphertext = substitute_encrypt(plaintext, key)
with open("PES2UG23CS272_cipher.txt", "w") as f:
f.write(ciphertext)
print("\nEncrypted Ciphertext Preview:\n")
print(ciphertext[:200])
inverse_key = {v: k for k, v in key.items()}
def substitute_decrypt(text, keymap):
result = []
for ch in text:
if ch in keymap:
result.append(keymap[ch])
else:
result.append(ch)
return "".join(result)
decrypted_text = substitute_decrypt(ciphertext, inverse_key)
with open("PES2UG23CS272_decrypted.txt", "w") as f:
f.write(decrypted_text)
print("\nDecrypted Text Preview:\n")
print(decrypted_text[:2000])
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

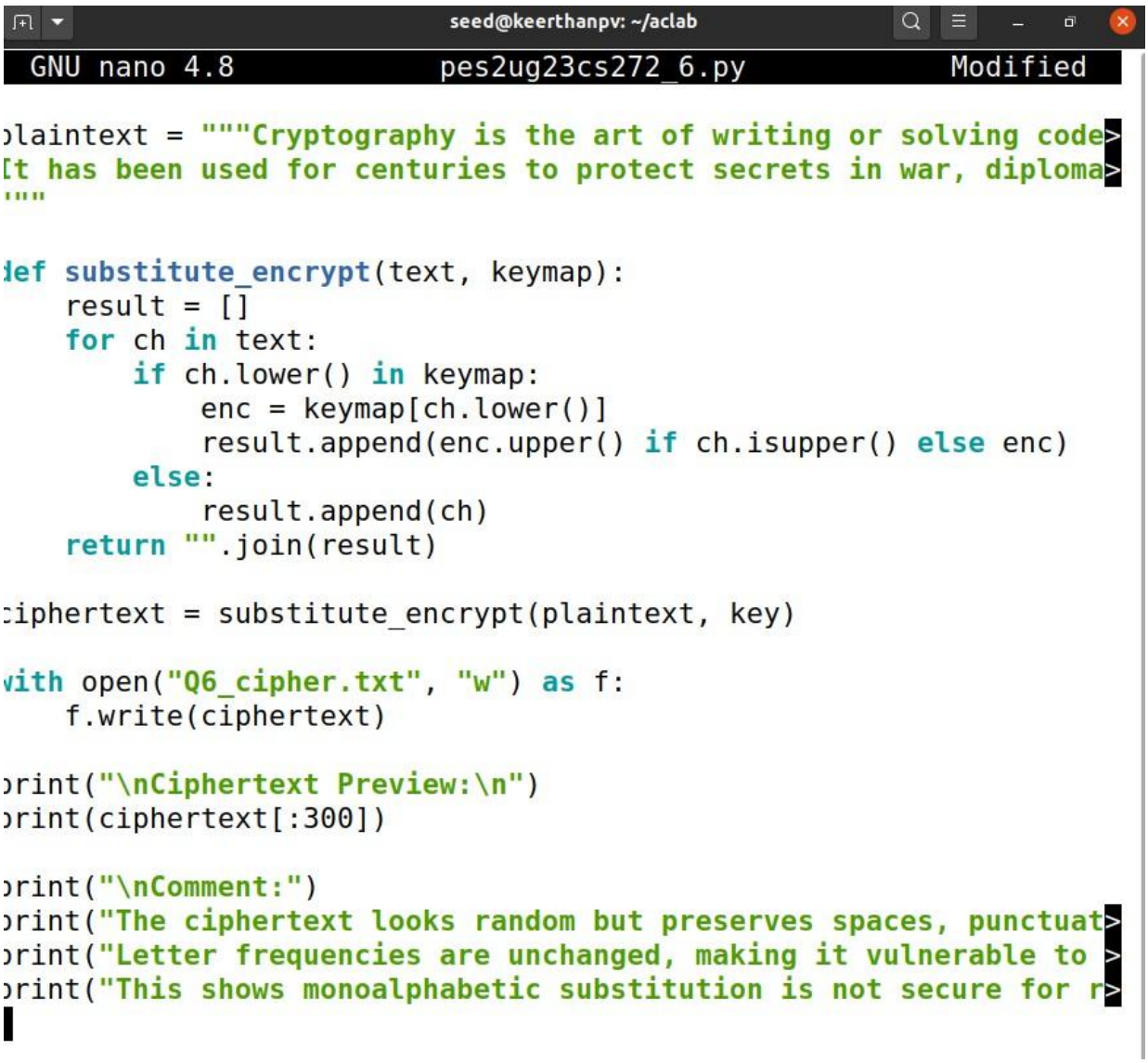
```
[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272_5.py
[08/22/25]seed@keerthanpv:~/aclab$ python3 pes2ug23cs272_5.py
Substitution Key Mapping:
a → m
b → e
c → n
d → q
e → a
f → v
g → d
h → j
i → r
j → y
k → h
l → w
m → s
n → t
o → c
p → k
q → p
r → b
s → l
t → g
u → u
v → x
w → o
x → f
y → z
z → i
```

```
Encrypted Ciphertext Preview:

gja wadatq cv hrtd mbgjub gawwl cv m locbq nmwwaq afnmwreub, lmrq
gc kcllall smdrnmw kcoabl mtq dbmtg xrngcbz gc rgl orawqab. eug ct
wz gja gbua hrtd ncuwq bascxa gja locbq vbcs gja lgcta.
mnncbqrtd gc

Decrypted Text Preview:

the legend of king arthur tells of a sword called excalibur, said
to possess magical powers and grant victory to its wielder. but on
ly the true king could remove the sword from the stone.
according to the tales, arthur was a young boy of humble origins.
he approached the stone without fear, while noble knights and rule
rs had failed.
with a simple grip, he pulled the sword out effortlessly, proving
himself to be the rightful king.
excalibur became a symbol of strength, justice, and honor.
under arthur's rule, the kingdom prospered,
and the knights of the round table fought bravely for peace and eq
uality.
yet legends speak of tragedy as well.
for every heroic tale, there came a great challenge. the betrayal
of friends, the loss of trust,
and the final battle that would determine the future of his kingdo
m.
still, the story of king arthur and his sword lives on, a timeless
 reminder that true power lies not in magic, but in courage, wisdo
m, and honor.

[08/22/25]seed@keerthanpv:~/aclab$
```

Question 6: Suppose the input file is:

Cryptography is the art of writing or solving codes. It has been used for centuries to protect secrets in war, diplomacy, and private communication. **Comment on the ciphertext generated**

```
  GNU nano 4.8              pes2ug23cs272_6.py              Modified
import string
import random

alphabet = list(string.ascii_lowercase)
shuffled = alphabet.copy()
random.shuffle(shuffled)
key = dict(zip(alphabet, shuffled))

print("Substitution Key Mapping:")
for a, b in key.items():
    print(f"{a} → {b}")

plaintext = """Cryptography is the art of writing or solving code>
It has been used for centuries to protect secrets in war, diploma>
"""

def substitute_encrypt(text, keymap):
    result = []
    for ch in text:
        if ch.lower() in keymap:
            enc = keymap[ch.lower()]
            result.append(enc.upper() if ch.isupper() else enc)
        else:
            result.append(ch)
    return "".join(result)

ciphertext = substitute_encrypt(plaintext, key)


^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell
```


seed@keerthanpv: ~/aclab
GNU nano 4.8 pes2ug23cs272_6.py Modified
plaintext = """Cryptography is the art of writing or solving code
It has been used for centuries to protect secrets in war, diploma
"""
def substitute_encrypt(text, keymap):
result = []
for ch in text:
if ch.lower() in keymap:
enc = keymap[ch.lower()]
result.append(enc.upper() if ch.isupper() else enc)
else:
result.append(ch)
return "".join(result)
ciphertext = substitute_encrypt(plaintext, key)
with open("Q6_cipher.txt", "w") as f:
f.write(ciphertext)
print("\nCiphertext Preview:\n")
print(ciphertext[:300])
print("\nComment:")
print("The ciphertext looks random but preserves spaces, punctuat
print("Letter frequencies are unchanged, making it vulnerable to
print("This shows monoalphabetic substitution is not secure for r

```
 reminder that true power lies not in magic, but in courage, wisdo
m, and honor.

[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272_6.py
[08/22/25]seed@keerthanpv:~/aclab$ python3 pes2ug23cs272_6.py
Substitution Key Mapping:
a → m
b → h
c → c
d → n
e → b
f → f
g → j
h → o
i → d
j → w
k → k
l → a
m → i
n → g
o → z
p → y
q → p
r → u
s → r
t → e
u → q
v → t
w → v
x → s
y → x
z → l
```

```
Ciphertext Preview:

Cuxyezjumyox dr eob mue zf vudedgj zu rzatdgj cznbr.
De omr hbbg qrbn fzu cbgequdbr ez yuzebce rbcuber dg vmu, ndyazimc
x, mgn yudtmeb cziiqgdcmedzg.


Comment:
The ciphertext looks random but preserves spaces, punctuation, and
 word lengths.
Letter frequencies are unchanged, making it vulnerable to frequenc
y analysis.
This shows monoalphabetic substitution is not secure for real-worl
d use.
[08/22/25]seed@keerthanpv:~/aclab$
```

Question 7:Frequency Analysis on Monoalphabetic Substitution Cipher

You are given the following ciphertext, which was produced using a Monoalphabetic Substitution Cipher:

BFPXTRWTOW CEN BRTEWTA JNFVU E NJLNWFWJWFGV BFPXTR CXTRT TEBX QTWWTR MEPN WG EVGWXTR VGWFBT WXT PEWWTRV GZ WXT CGRAN WXT BGMMGV QTWWTRN GZ WXT TVUQFNX EQPXELTW XTQP FV ATBGAFVU E NMEQQ UJTNN EW ZFRNW WTNWFVU WXTM FV WXT WTOW EVA WXT RTNJQW LTBGMTN BQTERTR WXT PRGBTNN RTDTEQN WXT NTBRTW GZ WXT BGAT GVBT WXT MEPPFVU FN BGRRTBW WXT MTNNEUT XFAATV FN WXEW NJLNWFWJWFGV BFPXTRN ERT NFMPQT LJW ZRTIJTVBK EVEQKNFN MESTN WXTM CTES

Tasks

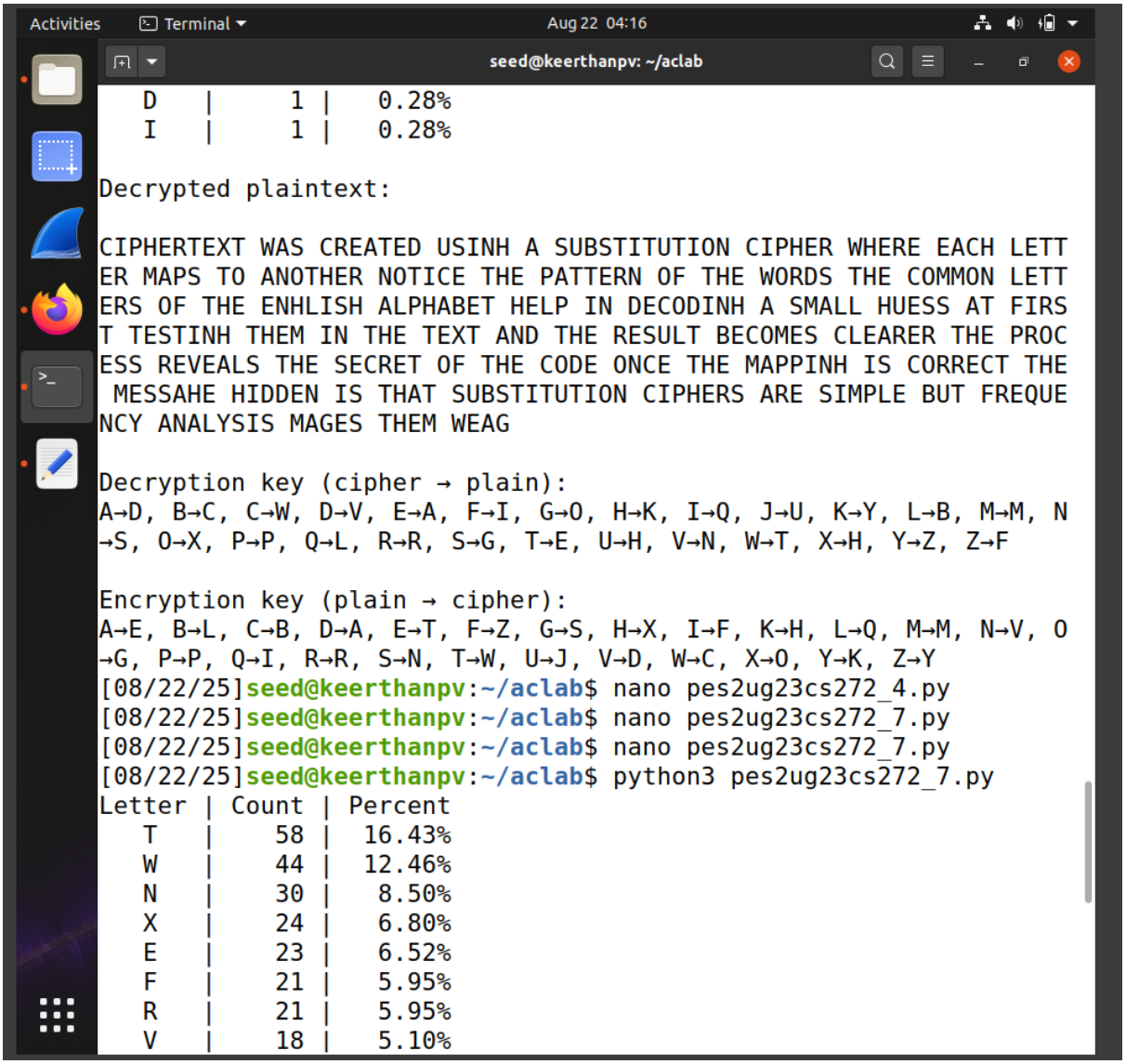
1. Perform **letter frequency analysis** on the ciphertext. Create a table of letter counts and percentages.
2. Compare the letter frequencies with **standard English frequencies**.
3. Make educated guesses for the most common ciphertext letters (e.g., likely e, t, a, o).
4. Look for common patterns:
    - Single-letter words (a, I) ○ Two-letter words (to, of, in, …) ○ Three-letter words (the, and, for, …)
5. Iteratively apply substitutions and reveal more plaintext.
6. Fully decrypt the message.
7. Submit:
    - Ciphertext letter frequency table ○ Substitution mapping (ciphertext → plaintext)
    - Decrypted plaintext

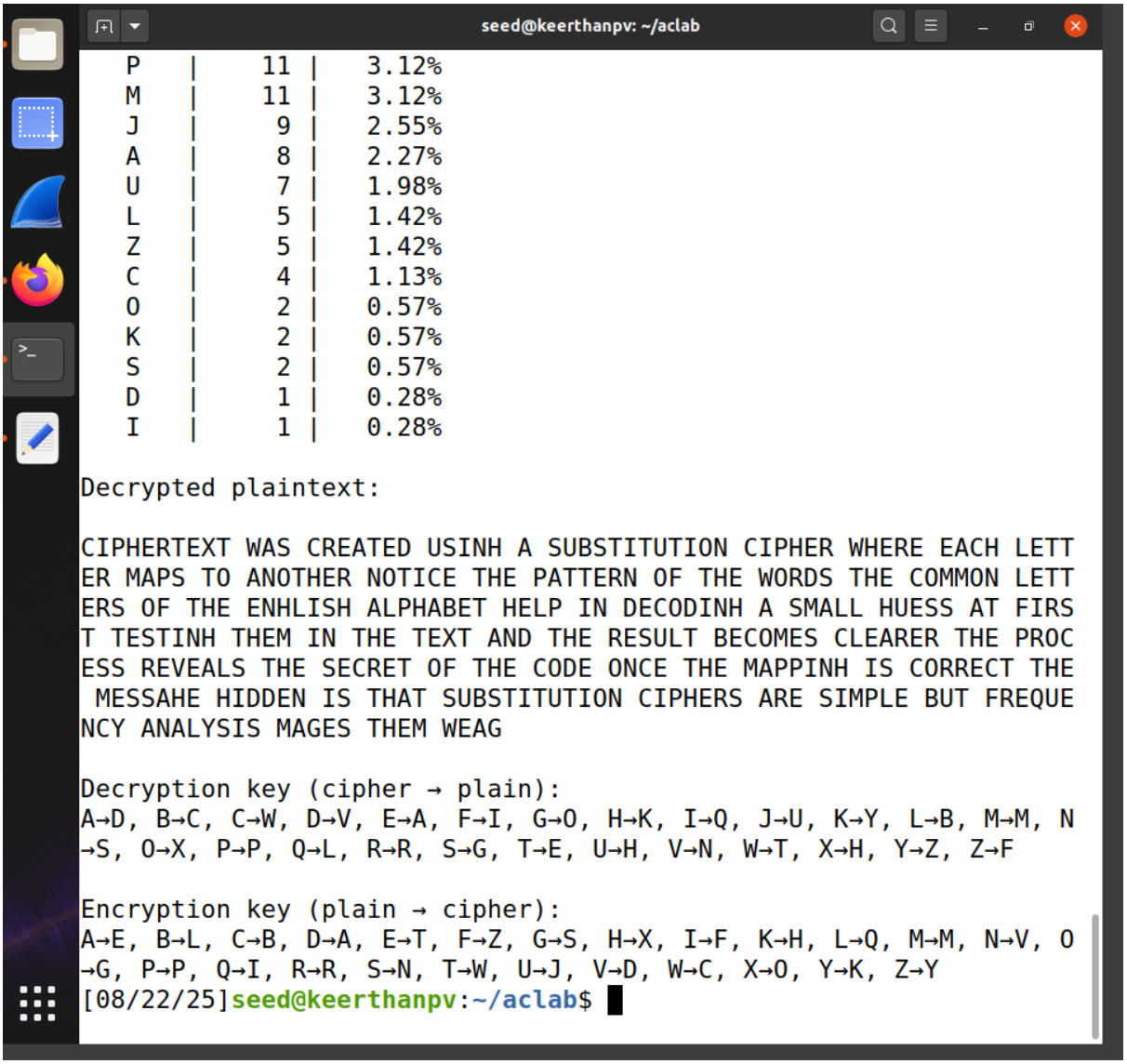
```
# 1) Frequency table
letters_only = [c for c in ciphertext if c.isalpha()]
counts = Counter(letters_only)
total = sum(counts.values())

print("Letter | Count | Percent")
for l, c in counts.most_common():
    print(f"  {l:>2}   | {c:>5} | {c/total*100:6.2f}%")
print()

# 2) Decryption mapping (cipher -> plain), solved for this cipher>
c2p = {
    'A':'D','B':'C','C':'W','D':'V','E':'A','F':'I','G':'O','H':'>
    'K':'Y','L':'B','M':'M','N':'S','O':'X','P':'P','Q':'L','R':'>
    'U':'H','V':'N','W':'T','X':'H','Y':'Z','Z':'F'
}
# (Some letters may be unused in this particular text; mapping pr>

# 3) Decrypt
def decrypt(text, key):
    out = []
    for ch in text:
        if ch.isalpha():
            pt = key[ch.upper()]
            out.append(pt.lower() if ch.islower() else pt)
        else:
            out.append(ch)

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell
```

```
# 3) Decrypt
def decrypt(text, key):
    out = []
    for ch in text:
        if ch.isalpha():
            pt = key[ch.upper()]
            out.append(pt.lower() if ch.islower() else pt)
        else:
            out.append(ch)
    return "".join(out)

plaintext = decrypt(ciphertext, c2p)

print("Decrypted plaintext:\n")
print(plaintext)
print()

# 4) Show key(s)
print("Decryption key (cipher → plain):")
print(", ".join(f"{c}→{p}" for c,p in sorted(c2p.items())))

# Also show the inverse (plain → cipher), i.e., the encryption ke>
p2c = {p:c for c,p in c2p.items()}
print("\nEncryption key (plain → cipher):")
print(", ".join(f"{p}→{c}" for p,c in sorted(p2c.items())))
```

```
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell
```

```
[08/22/25]seed@keerthanpv:~/aclab$ python3  pes2ug23cs272_7.py
Letter | Count | Percent
   T   |    58 |  16.43%
   W   |    44 |  12.46%
   N   |    30 |   8.50%
   X   |    24 |   6.80%
   E   |    23 |   6.52%
   F   |    21 |   5.95%
   R   |    21 |   5.95%
   V   |    18 |   5.10%
   B   |    17 |   4.82%
   G   |    17 |   4.82%
   Q   |    12 |   3.40%
   P   |    11 |   3.12%
   M   |    11 |   3.12%
   J   |     9 |   2.55%
   A   |     8 |   2.27%
   U   |     7 |   1.98%
   L   |     5 |   1.42%
   Z   |     5 |   1.42%
   C   |     4 |   1.13%
   O   |     2 |   0.57%
   K   |     2 |   0.57%
   S   |     2 |   0.57%
   D   |     1 |   0.28%
   I   |     1 |   0.28%

Decrypted plaintext:

CIPHERTEXT WAS CREATED USINH A SUBSTITUTION CIPHER WHERE EACH LETT
ER MAPS TO ANOTHER NOTICE THE PATTERN OF THE WORDS THE COMMON LETT
ERS OF THE ENHLISH ALPHABET HELP IN DECODINH A SMALL HUESS AT FIRS
T TESTINH THEM IN THE TEXT AND THE RESULT BECOMES CLEARER THE PROC
```


Activities
Terminal
Aug 22 04:16
seed@keerthanpv: ~/aclab
D | 1 | 0.28%
I | 1 | 0.28%
Decrypted plaintext:
CIPHERTEXT WAS CREATED USINH A SUBSTITUTION CIPHER WHERE EACH LETT
ER MAPS TO ANOTHER NOTICE THE PATTERN OF THE WORDS THE COMMON LETT
ERS OF THE ENHLISH ALPHABET HELP IN DECODINH A SMALL HUESS AT FIRS
T TESTINH THEM IN THE TEXT AND THE RESULT BECOMES CLEARER THE PROC
ESS REVEALS THE SECRET OF THE CODE ONCE THE MAPPINH IS CORRECT THE
MESSAHE HIDDEN IS THAT SUBSTITUTION CIPHERS ARE SIMPLE BUT FREQUE
NCY ANALYSIS MAGES THEM WEAG
Decryption key (cipher → plain):
A→D, B→C, C→W, D→V, E→A, F→I, G→O, H→K, I→Q, J→U, K→Y, L→B, M→M, N
→S, O→X, P→P, Q→L, R→R, S→G, T→E, U→H, V→N, W→T, X→H, Y→Z, Z→F
Encryption key (plain → cipher):
A→E, B→L, C→B, D→A, E→T, F→Z, G→S, H→X, I→F, K→H, L→Q, M→M, N→V, O
→G, P→P, Q→I, R→R, S→N, T→W, U→J, V→D, W→C, X→O, Y→K, Z→Y
[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272_4.py
[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272_7.py
[08/22/25]seed@keerthanpv:~/aclab$ nano pes2ug23cs272_7.py
[08/22/25]seed@keerthanpv:~/aclab$ python3 pes2ug23cs272_7.py
Letter | Count | Percent
T | 58 | 16.43%
W | 44 | 12.46%
N | 30 | 8.50%
X | 24 | 6.80%
E | 23 | 6.52%
F | 21 | 5.95%
R | 21 | 5.95%
V | 18 | 5.10%


seed@keerthanpv: ~/aclab
P | 11 | 3.12%
M | 11 | 3.12%
J | 9 | 2.55%
A | 8 | 2.27%
U | 7 | 1.98%
L | 5 | 1.42%
Z | 5 | 1.42%
C | 4 | 1.13%
O | 2 | 0.57%
K | 2 | 0.57%
S | 2 | 0.57%
D | 1 | 0.28%
I | 1 | 0.28%
Decrypted plaintext:
CIPHERTEXT WAS CREATED USINH A SUBSTITUTION CIPHER WHERE EACH LETT
ER MAPS TO ANOTHER NOTICE THE PATTERN OF THE WORDS THE COMMON LETT
ERS OF THE ENHLISH ALPHABET HELP IN DECODINH A SMALL HUESS AT FIRS
T TESTINH THEM IN THE TEXT AND THE RESULT BECOMES CLEARER THE PROC
ESS REVEALS THE SECRET OF THE CODE ONCE THE MAPPINH IS CORRECT THE
MESSAHE HIDDEN IS THAT SUBSTITUTION CIPHERS ARE SIMPLE BUT FREQUE
NCY ANALYSIS MAGES THEM WEAG
Decryption key (cipher → plain):
A→D, B→C, C→W, D→V, E→A, F→I, G→O, H→K, I→Q, J→U, K→Y, L→B, M→M, N
→S, O→X, P→P, Q→L, R→R, S→G, T→E, U→H, V→N, W→T, X→H, Y→Z, Z→F
Encryption key (plain → cipher):
A→E, B→L, C→B, D→A, E→T, F→Z, G→S, H→X, I→F, K→H, L→Q, M→M, N→V, O
→G, P→P, Q→I, R→R, S→N, T→W, U→J, V→D, W→C, X→O, Y→K, Z→Y
[08/22/25]seed@keerthanpv:~/aclab$