ASSIGNMENT

Shahjalal University of Science and Technology

Course: Introduction to Computer Security and Forensics

Course No: CSE-461

Submitted By	Submitted To
Name: MD Taohid Imam Khan Tamim Dept: CSE Session:2019-20 Reg no: 2019331018 Section B	Md. Shadmim Hasan Shifat Lecturer Dept. Of CSE SUST


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
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
U S A
S. S.

Task 1(a)

```
def caesar_decrypt(ciphertext, shift):
    decrypted_text = "" # empty string to store the decrypted text
    for char in ciphertext: # iterate through each character in the
ciphertext
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a') #
set the ASCII offset based on the case of the character
            decrypted_text += chr((ord(char) - ascii_offset - shift) %
26 + ascii_offset)
        else:
            decrypted_text += char
    return decrypted_text

ciphertext = input("Enter the ciphertext: ")
shift = int(input("Enter the shift value: "))
decrypted_text = caesar_decrypt(ciphertext, shift)
print("Decrypted text:", decrypted_text)
```

```
Decrypted text: CSE CARNIVAL AT IICT BUILDING SUST IS GOING TO BE
GREAT AGAIN.
```

Task 1(b)

```
def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr((ord(char) - ascii_offset - shift) %
26 + ascii_offset)
            # shift the character back by the shift value and wrap
around the alphabet
        else:
            decrypted_text += char
    return decrypted_text

ciphertext = "XNZ XVMIDQVG VO DDXO WPDGYDIB NPNO DN BJDIB OJ WZ BMZVO
VBVDI."
for shift in range(26):
    decrypted_text = caesar_decrypt(ciphertext, shift)
    print("Key:", shift, "Decrypted text:", decrypted_text)
```

```
Key: 0 Decrypted text: XNZ XVMIDQVG VO DDXO WPDGYDIB NPNO DN BJDIB OJ
WZ BMZVO VBVDI.
Key: 1 Decrypted text: WMY WULHCPUF UN CCWN VOCFXCHA MOMN CM AICHA NI
VY ALYUN UAUCH.
Key: 2 Decrypted text: VLX VTKGBOTE TM BBVM UNBEWBGZ LNLM BL ZHBGZ MH
UX ZKXTM TZTBG.
```

```
Key: 3 Decrypted text: UKW USJFANSD SL AAUL TMADVAFY KMKL AK YGAFY LG
TW YJWSL SYSAF.
Key: 4 Decrypted text: TJV TRIEZMRC RK ZZTK SLZCUZEX JLJK ZJ XFZEX KF
SV XIVRK RXRZE.
Key: 5 Decrypted text: SIU SQHDYLQB QJ YYSJ RKYBTYDW IKIJ YI WEYDW JE
RU WHUQJ QWQYD.
Key: 6 Decrypted text: RHT RPGCXKPA PI XXRI QJXASXCV HJHI XH VDXCV ID
QT VGTPI PVPXC.
Key: 7 Decrypted text: QGS QOFBWJOZ OH WWQH PIWZRWBU GIGH WG UCWBU HC
PS UFSOH OUOWB.
Key: 8 Decrypted text: PFR PNEAVINY NG VVPG OHVYQVAT FHFG VF TBVAT GB
OR TERNG NTNVA.
Key: 9 Decrypted text: OEQ OMDZUHMX MF UUOF NGUXPUZS EGEF UE SAUZS FA
NQ SDQMF MSMUZ.
Key: 10 Decrypted text: NDP NLCYTGLW LE TTNE MFTWOTYR DFDE TD RZTYR EZ
MP RCPLE LRLTY.
Key: 11 Decrypted text: MCO MKBXSFKV KD SSMD LESVNSXQ CECD SC QYSXQ DY
LO QBOKD KQKSX.
Key: 12 Decrypted text: LBN LJAWREJU JC RRLC KDRUMRWP BDBC RB PXRWP CX
KN PANJC JPJRW.
Key: 13 Decrypted text: KAM KIZVQDIT IB QQKB JCQTLQVO ACAB QA OWQVO BW
JM OZMIB IOIQV.
Key: 14 Decrypted text: JZL JHYUPCHS HA PPJA IBPSKPUN ZBZA PZ NVPUN AV
IL NYLHA HNHPU.
Key: 15 Decrypted text: IYK IGXTOBGR GZ OOIZ HAORJOTM YAYZ OY MUOTM ZU
HK MXKGZ GMGOT.
Key: 16 Decrypted text: HXJ HFWSNAFQ FY NNHY GZNQINSL XZXY NX LTNSL YT
GJ LWJFY FLFNS.
Key: 17 Decrypted text: GWI GEVRMZEP EX MMGX FYMPHMRK WYWX MW KSMRK XS
FI KVIEX EKEMR.
Key: 18 Decrypted text: FVH FDUQLYDO DW LLFW EXLOGLQJ VXVW LV JRLQJ WR
EH JUHDW DJDLQ.
Key: 19 Decrypted text: EUG ECTPKXCN CV KKEV DWKNFKPI UWUV KU IQKPI VQ
DG ITGCV CICKP.
Key: 20 Decrypted text: DTF DBSOJWBM BU JJDU CVJMEJOH TVTU JT HPJOH UP
CF HSFBU BHBJO.
Key: 21 Decrypted text: CSE CARNIVAL AT IICT BUILDING SUST IS GOING TO
BE GREAT AGAIN.
Key: 22 Decrypted text: BRD BZQMHUZK ZS HHBS ATHKCHMF RTRS HR FNHMF SN
AD FQDZS ZFZHM.
Key: 23 Decrypted text: AQC AYPLGTYJ YR GGAR ZSGJBGLE QSQR GQ EMGLE RM
ZC EPCYR YEYGL.
Key: 24 Decrypted text: ZPB ZXOKFSXI XQ FFZQ YRFIAFKD PRPQ FP DLFKD QL
YB DOBXQ XDXFK.
Key: 25 Decrypted text: YOA YWNJERWH WP EEYP XQEHZEJC OQOP EO CKEJC PK
XA CNAWP WCWEJ.
```

This explanation details the implementation and usage of a Python function to decrypt Caesar cipher text and the process of identifying the original message.

Decryption Function:

Inputs:

- `ciphertext`: The encrypted text to decrypt.
- `key`: The number of positions to shift each letter back.

Process:

1. **Creating alphabet reference:**
   - Defining a string `alphabet` containing all uppercase letters (`'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`).
2. **Iterate through characters:**
   - Looping through each character in the `ciphertext`:
     - **If letter:**
       - Convert character to uppercase using `upper()`.
       - Find its index in the `alphabet`.
       - Subtract the `key` and apply modulo 26 to handle shifts beyond the alphabet range.
       - Append the decrypted lowercase letter (using `lower()`) to `plaintext`.
     - **If not a letter:**
       - Append the character directly to `plaintext`.
3. **Return plaintext:**
   - Return the constructed `plaintext` string.

Cracking Process:

1. **Store ciphertext:**
   - Assign the provided ciphertext to a variable, e.g., `ciphertext = "XNZ XVMIDQVG VO DDXO WPDGYDIB NPNO DN BJDIB OJ WZ BMZVO VBVDI."`.
2. **Iterate through keys:**
   - Loop through possible key values (1 to 26).
3. **Decrypt with each key:**
   - Call the `caesar_decrypt` function with the current `key` and `ciphertext`.
   - Store the decrypted text in a `plaintext` variable.
4. **Print results:**
   - Print the current `key` and the corresponding `plaintext`.
5. **Manual identification:**
   - Examine the printed outputs to find the `plaintext` that makes grammatical and semantic sense, revealing the correct `key` and the original message.

I am showing the table of all the possible keys and their corresponding decrypted text

Shift	Decrypted text
0	XNZ XVMIDQVG VO DDXO WPDGYDIB NPNO DN BJDIB OJ WZ BMZVO VBVDI.
1	WMY WULHCPUF UN CCWN VOCFXCHA MOMN CM

Shift	Decrypted text
	AICHA NI VY ALYUN UAUCH.
2	VLX VTKGBOTE TM BBVM UNBEWBGZ LNLM BL ZHBGZ MH UX ZKXTM TZTBG.
3	UKW USJFANSD SL AAUL TMADVAFY KMKL AK YGAFY LG TW YJWSL SYSAF.
4	TJV TRIEZMRC RK ZZTK SLZCUZEX JLJK ZJ XFZEX KF SV XIVRK RXRZE.
5	SIU SQHDYLQB QJ YYSJ RKYBTYDW IKIJ YI WEYDW JE RU WHUQJ QWQYD.
6	RHT RPGCXKPA PI XXRI QJXASXCV HJHI XH VDXCV ID QT VGTPI PVPXC.
7	QGS QOFBWJOZ OH WWQH PIWZRWBU GIGH WG UCWBU HC PS UFSOH OUOWB.
8	PFR PNEAVINY NG VVPG OHVYQVAT FHFG VF TBVAT GB OR TERNG NTNVA.
9	OEQ OMDZUHMX MF UUOF NGUXPUZS EGEF UE SAUZS FA NQ SDQMF MSMUZ.
10	NDP NLCYTGLW LE TTNE MFTWOTYR DFDE TD RZTYR EZ MP RCPLE LRLTY.
11	MCO MKBXSFKV KD SSMD LESVNSXQ CECD SC QYSXQ DY LO QBOKD KQKSX.
12	LBN LJAWREJU JC RRLC KDRUMRWP BDBC RB PXRWP CX KN PANJC JPJRW.
13	KAM KIZVQDIT IB QQKB JCQTLQVO ACAB QA OWQVO BW JM OZMIB IOIQV.
14	JZL JHYUPCHS HA PPJA IBPSKPUN ZBZA PZ NVPUN AV IL NYLHA HNHPU.
15	IYK IGXTOBGR GZ OOIZ HAORJOTM YAYZ OY MUOTM ZU HK MXKGZ GMGOT.
16	HXJ HFWSNAFQ FY NNHY GZNQINSL XZXY NX LTNSL YT GJ LWJFY FLFNS.
17	GWI GEVRMZEP EX MMGX FYMPHMRK WYWX MW KSMRK XS FI KVIEX EKEMR.
18	FVH FDUQLYDO DW LLFW EXLOGLQJ VXVW LV JRLQJ WR EH JUHDW DJDLQ.
19	EUG ECTPKXCN CV KKEV DWKNFKPI UWUV KU IQKPI VQ DG ITGCV CICKP.
20	DTF DBSOJWBM BU JJDU CVJMEJOH TVTU JT HPJOH UP CF HSFBU BHBJO.
21	CSE CARNIVAL AT IICT BUILDING SUST IS GOING TO BE GREAT AGAIN.

Shift	Decrypted text
22	BRD BZQMHUZK ZS HHBS ATHKCHMF RTRS HR FNHMF SN AD FQDZS ZFZHM.
23	AQC AYPLGTYJ YR GGAR ZSGJBGLE QSQR GQ EMGLE RM ZC EPCYR YEYGL.
24	ZPB ZXOKFSXI XQ FFZQ YRFIAFKD PRPQ FP DLFKD QL YB DOBXQ XDXFK.
25	YOA YWNJERWH WP EEYP XQEHZEJC OQOP EO CKEJC PK XA CNAWP WCWEJ.

So after manyally checking all the decrypted text the text on key=21 is grammatically and syntactically correct

Decrypted Text=CSE CARNIVAL AT IICT BUILDING SUST IS GOING TO BE GREAT AGAIN.
key=21

Task-01(c)

```
def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr((ord(char) - ascii_offset - shift) %
26 + ascii_offset)
            # shift the character back by the shift value and wrap
around the alphabet
        else:
            decrypted_text += char
    return decrypted_text

ciphertext = "ZKDW GR BRX JHW ZKHQ BRX FURVV D VQRZPDQ ZLWK D YDPSLUH?
IURVWELWH"
for shift in range(26):
    decrypted_text = caesar_decrypt(ciphertext, shift)
    print("Key:", shift, "Decrypted text:", decrypted_text)
```

```
Key: 0 Decrypted text: ZKDW GR BRX JHW ZKHQ BRX FURVV D VQRZPDQ ZLWK D
YDPSLUH? IURVWELWH
Key: 1 Decrypted text: YJCV FQ AQW IGV YJGP AQW ETQUU C UPQYOCP YKVJ C
XCORKTG? HTQUVDKVG
Key: 2 Decrypted text: XIBU EP ZPV HFU XIFO ZPV DSPTT B TOPXNBO XJUI B
WBNQJSF? GSPTUCJUF
Key: 3 Decrypted text: WHAT DO YOU GET WHEN YOU CROSS A SNOWMAN WITH A
VAMPIRE? FROSTBITE
Key: 4 Decrypted text: VGZS CN XNT FDS VGDM XNT BQNRR Z RMNVLZM VHSG Z
UZLOHQD? EQNRSAHSD
Key: 5 Decrypted text: UFYR BM WMS ECR UFCL WMS APMQQ Y QLMUKYL UGRF Y
TYKNGPC? DPMQRZGRC
```

```
Key: 6 Decrypted text: TEXQ AL VLR DBQ TEBK VLR ZOLPP X PKLTJXK TFQE X SXJMFOB? COLPQYFQB
Key: 7 Decrypted text: SDWP ZK UKQ CAP SDAJ UKQ YNKOO W OJKSIWJ SEPD W RWILENA? BNKOPXEPA
Key: 8 Decrypted text: RCVO YJ TJP BZO RCZI TJP XMJNN V NIJRHVI RDOC V QVHKDMZ? AMJNOWDOZ
Key: 9 Decrypted text: QBUN XI SIO AYN QBYH SIO WLIMM U MHIQGUH QCNB U PUGJCLY? ZLIMNVCNY
Key: 10 Decrypted text: PATM WH RHN ZXM PAXG RHN VKHLL T LGHPFTG PBMA T OTFIBKX? YKHLMUBMX
Key: 11 Decrypted text: OZSL VG QGM YWL OZWF QGM UJGKK S KFGOESF OALZ S NSEHAJW? XJGKLTALW
Key: 12 Decrypted text: NYRK UF PFL XVK NYVE PFL TIFJJ R JEFNDRE NZKY R MRDGZIV? WIFJKSZKV
Key: 13 Decrypted text: MXQJ TE OEK WUJ MXUD OEK SHEII Q IDEMCQD MYJX Q LQCFYHU? VHEIJRYJU
Key: 14 Decrypted text: LWPI SD NDJ VTI LWTC NDJ RGDHH P HCDLBPC LXIW P KPBEXGT? UGDHIQXIT
Key: 15 Decrypted text: KVOH RC MCI USH KVSB MCI QFCGG O GBCKAOB KWHV O JOADWFS? TFCGHPWHS
Key: 16 Decrypted text: JUNG QB LBH TRG JURA LBH PEBFF N FABJZNA JVGU N INZCVER? SEBFGOVGR
Key: 17 Decrypted text: ITMF PA KAG SQF ITQZ KAG ODAEE M EZAIYMZ IUFT M HMYBUDQ? RDAEFNUFQ
Key: 18 Decrypted text: HSLE OZ JZF RPE HSPY JZF NCZDD L DYZHXLY HTES L GLXATCP? QCZDEMTEP
Key: 19 Decrypted text: GRKD NY IYE QOD GROX IYE MBYCC K CXYGWKX GSDR K FKWZSBO? PBYCDLSDO
Key: 20 Decrypted text: FQJC MX HXD PNC FQNW HXD LAXBB J BWXFVJW FRCQ J EJVYRAN? OAXBCKRCN
Key: 21 Decrypted text: EPIB LW GWC OMB EPMV GWC KZWAA I AVWEUIV EQBP I DIUXQZM? NZWABJQBM
Key: 22 Decrypted text: DOHA KV FVB NLA DOLU FVB JYVZZ H ZUVDTHU DPAO H CHTWPYL? MYVZAIPAL
Key: 23 Decrypted text: CNGZ JU EUA MKZ CNKT EUA IXUYY G YTUCSGT COZN G BGSVOXK? LXUYZHOZK
Key: 24 Decrypted text: BMFY IT DTZ LJY BMJS DTZ HWTXX F XSTBRFS BNYM F AFRUNWJ? KWTXYGNYJ
Key: 25 Decrypted text: ALEX HS CSY KIX ALIR CSY GVSWW E WRSAQER AMXL E ZEQTMVI? JVSWXFMXI
```

In this case the cipher text was : "ZKDW GR BRX JHW ZKHQ BRX FURVV D VQRZPDQ ZLWK D YDPSLUH? IURVWELWH" I have described the cesar cipher decryption process in the above cell. I have used the same process to decrypt the given cipher text. I'm showing the result in the table below.

Shift	Decrypted text
0	ZKDW GR BRX JHW ZKHQ BRX FURVV D VQRZPDQ ZLWK D YDPSLUH? IURVWELWH

Shift	Decrypted text
1	YJCV FQ AQW IGV YJGP AQW ETQUU C UPQYOCP YKVJ C XCORKTG? HTQUVDKVG
2	XIBU EP ZPV HFU XIFO ZPV DSPTT B TOPXNBO XJUI B WBNQJSF? GSPTUCJUF
3	WHAT DO YOU GET WHEN YOU CROSS A SNOWMAN WITH A VAMPIRE? FROSTBITE
4	VGZS CN XNT FDS VGDM XNT BQNRR Z RMNVLZM VHSG Z UZLOHQD? EQNRSAHSD
5	UFYR BM WMS ECR UFCL WMS APMQQ Y QLMUKYL UGRF Y TYKNGPC? DPMQRZGRC
6	TEXQ AL VLR DBQ TEBK VLR ZOLPP X PKLTJXK TFQE X SXJMFOB? COLPQYFQB
7	SDWP ZK UKQ CAP SDAJ UKQ YNKOO W OJKSIWJ SEPD W RWILENA? BNKOPXEPA
8	RCVO YJ TJP BZO RCZI TJP XMJNN V NIJRHVI RDOC V QVHKDMZ? AMJNOWDOZ
9	QBUN XI SIO AYN QBYH SIO WLIMM U MHIQGUH QCNB U PUGJCLY? ZLIMNVCNY
10	PATM WH RHN ZXM PAXG RHN VKHLL T LGHPFTG PBMA T OTFIBKX? YKHLMUBMX
11	OZSL VG QGM YWL OZWF QGM UJGKK S KFGOESF OALZ S NSEHAJW? XJGKLTALW
12	NYRK UF PFL XVK NYVE PFL TIFJJ R JEFNDRE NZKY R MRDGZIV? WIFJKSZKV
13	MXQJ TE OEK WUJ MXUD OEK SHEII Q IDEMCQD MYJX Q LQCFYHU? VHEIJRYJU
14	LWPI SD NDJ VTI LWTC NDJ RGDHH P HCDLBPC LXIW P KPBEXGT? UGDHIQXIT
15	KVOH RC MCI USH KVSB MCI QFCGG O GBCKAOB KWHV O JOADWFS? TFCGHPWHS
16	JUNG QB LBH TRG JURA LBH PEBFF N FABJZNA JVGU N INZCVER? SEBFGOVGR
17	ITMF PA KAG SQF ITQZ KAG ODAEE M EZAIYMZ IUFT M HMYBUDQ? RDAEFNUFQ
18	HSLE OZ JZF RPE HSPY JZF NCZDD L DYZHXLY HTES L GLXATCP? QCZDEMTEP
19	GRKD NY IYE QOD GROX IYE MBYCC K CXYGWKX GSDR K FKWZSBO? PBYCDLSDO
20	FQJC MX HXD PNC FQNW HXD LAXBB J BWXFVJW FRCQ J EJVYRAN? OAXBCKRCN
21	EPIB LW GWC OMB EPMV GWC KZWAA I AVWEUIV EQBP I DIUXQZM? NZWABJQBM

Shift	Decrypted text
22	DOHA KV FVB NLA DOLU FVB JYVZZ H ZUVDTHU DPAO H CHTWPYL? MYVZAIPAL
23	CNGZ JU EUA MKZ CNKT EUA IXUYY G YTUCSGT COZN G BGSVOXK? LXUYZHOZK
24	BMFY IT DTZ LJY BMJS DTZ HWTXX F XSTBRFS BNYM F AFRUNWJ? KWTXYGNYJ
25	ALEX HS CSY KIX ALIR CSY GVSWW E WRSAQER AMXL E ZEQTMVI? JVSWXFMXI

So after manyally checking all the decrypted text the text on key=3 is grammatically and syntactically correct And the decrypted text is : WHAT DO YOU GET WHEN YOU CROSS A SNOWMAN WITH A VAMPIRE? FROSTBITE

Task_2(a)

```
from collections import Counter
def print_text_with_spaces(text, words_per_line):
    words = text.split()
    for i in range(0, len(words), words_per_line):
        print(" ".join(words[i:i+words_per_line]))


def decrypt(text, mapping):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            if char in mapping:
                decrypted_text += mapping[char]
            else:
                decrypted_text += "*"
        else:
            decrypted_text += char  # Preserve punctuation
    return decrypted_text

def analyze_frequency(ciphertext):
    # Remove spaces and punctuation
    ciphertext = ''.join(char for char in ciphertext if
char.isalpha())
    # Count the frequency of each letter
    frequency = Counter(ciphertext)
    # Sort the letters by frequency
    sorted_frequency = sorted(frequency.items(), key=lambda x: x[1],
reverse=True)
    return sorted_frequency

def algo(text, mapping, words_per_line):
    while True:
        print("Current mapping:")
        print(mapping)
        reverse_mapping = {}
        found_duplicate = False
        for key, value in mapping.items():
          if value in reverse_mapping:
            print(f"{reverse_mapping[value]} maps to the same value as
{key}: {value}")
            found_duplicate = True
            break
          else:
            reverse_mapping[value] = key
        if found_duplicate:
```

```
          break
        frequency_list = analyze_frequency(text)
        decrypted_text = decrypt(text, mapping)
        print("Decrypted text:")
        print_text_with_spaces(decrypted_text, words_per_line)

        print("Frequency list:")
        for char, freq in frequency_list:
            print(f"{char}: {freq}")

        replacement = input("Enter replacement (or 'q' to quit): ")
        if replacement.lower() == 'q':
            break
        if len(replacement) != 2 or replacement[0] not in
'abcdefghijklmnopqrstuvwxyz' or replacement[1] not in
'abcdefghijklmnopqrstuvwxyz':
            print("Invalid input. Please enter a valid replacement.")
            continue
        if replacement[0] in mapping.values() or replacement[1] in
mapping.values():
            print("Error: One of the replacement characters is already
mapped to another character.")
            continue
        mapping[replacement[0]] = replacement[1]

        # Check if any character is mapped to the same replacement
        duplicate_mapping = [char for char, mapped_char in
mapping.items() if mapped_char == replacement[1] and char !=
replacement[0]]
        if duplicate_mapping:
            print(f"Error: Character '{duplicate_mapping[0]}' is
already mapped to '{replacement[1]}'.")
            del mapping[replacement[0]]  # Rollback the mapping
            continue

text = " gtd bsvgl vf fgedsugt dffml dkcymvsf gtmg gtd chjde ha \
aevdsxftvc tdycf bf gh id fgehsu aehz tmexftvcf. aevdsxf qms \
uvod bf gtd fgedsugt jd sddxjtds yvad udgf ghbut. vs \
mxxvgvhs, cdhcyd dkcedff bsvgl gtehbutyhod, amzvyl, aevdsxf, msx
hgtdef ftmed \
fghevdf ha avsxvsu qhzzhsuehbsx jvgt fhzdhsd. gtded med zmsl idsdavgf
\
ha fgmlvsu bsvgdx vsghbut gvzdf, mf vg tdycf gh amqd \
qtmyydsuvsu fvgbmgvhsf jvgtqhbemud. gtd vzchegmsqd ha fgmlvsu bsvgdx
tmf fgebqp \
m qthex mzhsuzmsl cdhcyd gtehbuthbg tvfghel. pddcvsu zdzhevdf ha jtmg
\
jd tmodmqqhzcyvftdx gtehbuthbg tvfghel qms tdyc bf fdd thj vsxvovxbmyf
\
```

```
msxqhzzbsvgvdf tmod cdefdodedx gtehbut ghbut gvzdf msx vsgh m
ievutgdeabgbed. "

mapping = {'m':'a'}
words_per_line = 10
algo(text, mapping, words_per_line)

Current mapping:
{}
Decrypted text:
*** ***** ** ******** ***** ******** **** *** ***** **
********** ***** ** ** ** ****** **** *********. ******* ***
**** ** *** ******** ** ******** **** **** *****. **
********, ****** ******* ***** ***********, ******, *******, ***
****** *****
******* ** ******* *********** **** *******. ***** *** **** ********
** ******* ****** ******* *****, ** ** ***** ** ****
*********** ********** ***********. *** ********** ** ******* ******
*** ******
* ***** ********* ****** ********** *******. ******* ******** ** ****
** ***************** ********** ******* *** **** ** *** *** ***********
************** **** ********** ******* ***** ***** *** **** *
***************.
Frequency list:
d: 65
g: 50
v: 45
f: 43
h: 42
t: 41
s: 40
e: 32
m: 32
b: 24
u: 22
x: 19
c: 16
z: 16
a: 15
y: 13
q: 12
l: 10
j: 8
o: 6
i: 3
k: 2
p: 2
Invalid input. Please enter a valid replacement.
Current mapping:
{}
```

```
Decrypted text:
*** ***** ** ******** ***** ******** **** *** ***** **
********** ***** ** ** ** ****** **** *********. ******* ***
**** ** *** ******** ** ******** **** **** *****. **
********, ****** ******* ***** ***********, ******, *******, ***
****** *****
******* ** ******* ************ **** *******. ***** *** **** ********
** ******* ****** ******* *****, ** ** ***** ** ****
*********** ********** ***********. *** ********** ** ******* ******
*** ******
* ***** ********* ****** ********** *******. ******* ******** ** ****
** ***************** ********** ******* *** **** ** *** *** ***********
************** **** ********** ******* ***** ***** *** **** *
***************.
Frequency list:
d: 65
g: 50
v: 45
f: 43
h: 42
t: 41
s: 40
e: 32
m: 32
b: 24
u: 22
x: 19
c: 16
z: 16
a: 15
y: 13
q: 12
l: 10
j: 8
o: 6
i: 3
k: 2
p: 2
Invalid input. Please enter a valid replacement.
Current mapping:
{}
Decrypted text:
*** ***** ** ******** ***** ******** **** *** ***** **
********** ***** ** ** ** ****** **** *********. ******* ***
**** ** *** ******** ** ******** **** **** *****. **
********, ****** ******* ***** ***********, ******, *******, ***
****** *****
******* ** ******* ************ **** *******. ***** *** **** ********
** ******* ****** ******* *****, ** ** ***** ** ****
```

********** ********** **********. *** ********** ** ******* ******
*** ******
* ***** ********* ****** ********** *******. ******* ******** ** ****
** ***************** ********** ******* *** **** ** *** *** ***********
************** **** ********** ******* ***** ***** *** **** *
**************.
Frequency list:
d: 65
g: 50
v: 45
f: 43
h: 42
t: 41
s: 40
e: 32
m: 32
b: 24
u: 22
x: 19
c: 16
z: 16
a: 15
y: 13
q: 12
l: 10
j: 8
o: 6
i: 3
k: 2
p: 2
Invalid input. Please enter a valid replacement.
Current mapping:
{}
Decrypted text:
*** ***** ** ******** ***** ******** **** *** ***** **
********** ***** ** ** ** ****** **** *********. ******* ***
**** ** *** ******** ** ******** **** **** *****. **
********, ****** ******* ***** ***********, ******, *******, ***
****** *****
******* ** ******* ************ **** *******. ***** *** **** ********
** ******* ****** ******* *****, ** ** ***** ** ****
********** ********** **********. *** ********** ** ******* ******
*** ******
* ***** ********* ****** ********** *******. ******* ******** ** ****
** ***************** ********** ******* *** **** ** *** *** ***********
************** **** ********** ******* ***** ***** *** **** *
**************.
Frequency list:
d: 65

```
g: 50
v: 45
f: 43
h: 42
t: 41
s: 40
e: 32
m: 32
b: 24
u: 22
x: 19
c: 16
z: 16
a: 15
y: 13
q: 12
l: 10
j: 8
o: 6
i: 3
k: 2
p: 2
```

Character	Frequency	Character	Frequency	Character	Frequency	Character	Frequency
d	65	g	50	v	45	f	43
h	42	t	41	s	40	e	32
m	32	b	24	u	22	x	19
c	16	z	16	a	15	y	13
q	12	l	10	j	8	o	6
i	3	k	2	p	2		

Cryptanalysis Hints

Order Of Frequency Of Single Letters	E T A O I N S H R D L U
Order Of Frequency Of Digraphs	th er on an re he in ed nd ha at en es of or nt ea ti to it st io le is ou ar as de rt ve
Order Of Frequency Of Trigraphs	the and tha ent ion tio for nde has nce edt tis oft sth men
Order Of Frequency Of Most Common Doubles	ss ee tt ff ll mm oo
Order Of Frequency Of Initial Letters	T O A W B C D S F M R H I Y E G L N P U J K
Order Of Frequency Of Final Letters	E S T D N R Y F L O G H A K M P U W
One-Letter Words	a, I.
Most Frequent Two-Letter Words	of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am
Most Frequent Three-Letter Words	the, and, for, are, but, not, you, all, any, can, had, her, was, one, our, out, day, get, has, him, his, how, man, new, now, old, see, two, way, who, boy, did, its, let, put, say, she, too, use
Most Frequent Four-Letter Words	that, with, have, this, will, your, from, they, know, want, been, good, much, some, time

I will the use the above hints attached to the lecture slide to solve the problem.

Mapping

1) **Current mapping:** {'m': 'a'}

```
Decrypted text:
*** ***** ** ******** ***a* ****a*** **a* *** ***** **
********** ***** ** ** ** ****** **** *a*******. ******* *a*
**** ** *** ******** ** ******** **** **** *****. **
a*******, ****** ******* ***** ***********, *a****, *******, a**
****** **a**
******* ** ******* *********** **** *******. ***** a** *a** ********
** **a**** ****** ******* *****, a* ** ***** ** *a**
**a******** ****a***** ********a**. *** ******a*** ** **a**** ******
*a* ******
a ***** a*****a** ****** ********** *******. ******* ******** ** **a*
** *a**a*********** ********** ******* *a* **** ** *** *** ********a**
a************ *a** ********** ******* ***** ***** a** **** a
**************.
```

Reason for mapping 'm' to 'a':

- In english lagnuage one letter word is 'a' and 'I'. But in the given text 'a' is more frequent than 'I'. So, I mapped 'm' to 'a'.
- Also the word I always positiond at the start of a sentence. But in the given text 'm' is not always at the start of a sentence. So, I mapped 'm' to 'a'.

2 **Current mapping:** {'m': 'a', 'd': 'e'}

```
Decrypted text:
**e ***** ** ***e**** e**a* e***a*** **a* **e ***e* **
***e****** *e*** ** ** *e ****** **** *a*******. ***e*** *a*
***e ** **e ***e**** *e *ee***e* ***e *e** *****. **
a*******, *e***e e***e** ***** **********e, *a****, ***e***, a**
***e** **a*e
*****e* ** ******* ************ **** ***e**e. **e*e a*e *a** *e*e****
** **a**** ****e* ******* ***e*, a* ** *e*** ** *a*e
**a**e***** ****a***** ********a*e. **e ******a**e ** **a**** ****e*
*a* ******
a ***** a*****a** *e***e ********** *******. *ee**** *e****e* ** **a*
*e *a*ea*********e* ********** ******* *a* *e** ** *ee *** ********a**
a***********e* *a*e *e**e*e*e* ******* ***** ***e* a** **** a
******e******e.
```

Reason for mapping d to e:

- In the given 'd' is the most frequent character.And we know in english language 'e' is most frequent letter .So, I mapped 'd' to 'e'.

1. **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's'}

```
Decrypted text:
**e ***** *s s**e**** essa* e***a**s **a* **e ***e* **
***e**s*** *e**s *s ** *e s***** **** *a**s***s. ***e**s *a*
***e *s **e s**e**** *e *ee***e* ***e *e*s *****. **
a*******, *e***e e***ess ***** **********e, *a****, ***e**s, a**
***e*s s*a*e
s****es ** ******* ************ **** s**e**e. **e*e a*e *a** *e*e***s
** s*a**** ****e* ******* ***es, as ** *e**s ** *a*e
**a**e***** s***a****s ********a*e. **e ******a**e ** s*a**** ****e*
*as s*****
a ***** a*****a** *e***e ********** **s****. *ee**** *e****es ** **a*
*e *a*ea*******s*e* ********** **s**** *a* *e** *s see *** ********a*s
a***********es *a*e *e*se*e*e* ******* ***** ***es a** **** a
******e******e.
```

Reason for mapping f to s:

- In the 6 the line we get a meaningful two letter word named "as" if we map f to s

4.**Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't'}

```
Decrypted text:
t*e ***t* *s st*e**t* essa* e***a**s t*at t*e ***e* **
***e**s*** *e**s *s t* *e st**** **** *a**s***s. ***e**s *a*
***e *s t*e st*e**t* *e *ee***e* ***e *ets t****. **
a***t***, *e***e e***ess ***t* t*********e, *a****, ***e**s, a**
```

```
*t*e*s s*a*e
st***es ** ******* ********** **t* s**e**e. t*e*e a*e *a** *e*e**ts
** sta**** ***te* **t**** t**es, as *t *e**s t* *a*e
**a**e***** s*t*at***s **t*****a*e. t*e *****ta**e ** sta**** ***te*
*as st****
a ***** a*****a** *e***e t********t **st***. *ee**** *e****es ** **at
*e *a*ea*******s*e* t********t **st*** *a* *e** *s see *** ********a*s
a*********t*es *a*e *e*se*e*e* t****** t**** t**es a** **t* a
*****te***t**e.
```

Reason for mapping g to t:

- I have assumed that the 1st word may be **the**. That's why I've mapped g to t

5.__Current mapping:__ {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h'}

```
Decrypted text:
the ***t* *s st*e**th essa* e***a**s that the ***e* **
***e**sh** he**s *s t* *e st**** **** ha**sh**s. ***e**s *a*
***e *s the st*e**th *e *ee**he* ***e *ets t***h. **
a***t***, *e***e e***ess ***t* th****h***e, *a****, ***e**s, a**
*the*s sha*e
st***es ** ******* ********** **th s**e**e. the*e a*e *a** *e*e**ts
** sta**** ***te* **t***h t**es, as *t he**s t* *a*e
*ha**e***** s*t*at***s **th****a*e. the *****ta**e ** sta**** ***te*
has st****
a *h*** a*****a** *e***e th****h**t h*st***. *ee**** *e****es ** *hat
*e ha*ea*******she* th****h**t h*st*** *a* he** *s see h** ********a*s
a*********t*es ha*e *e*se*e*e* th****h t***h t**es a** **t* a
****hte***t**e.
```

Reason for mapping t to h:

- I have assumed that the 1st word may be **the**. That's why I've mapped t to h

1. **Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i'}

```
Decrypted text:
the **it* is st*e**th essa* e***ai*s that the ***e* **
**ie**shi* he**s *s t* *e st**** **** ha**shi*s. **ie**s *a*
*i*e *s the st*e**th *e *ee**he* *i*e *ets t***h. i*
a**iti**, *e***e e***ess **it* th****h***e, *a*i**, **ie**s, a**
*the*s sha*e
st**ies ** *i**i** ********** *ith s**e**e. the*e a*e *a** *e*e*its
** sta*i** **ite* i*t***h ti*es, as it he**s t* *a*e
*ha**e**i** sit*ati**s *ith****a*e. the i****ta**e ** sta*i** **ite*
has st****
a *h*** a*****a** *e***e th****h**t hist***. *ee*i** *e***ies ** *hat
*e ha*ea******ishe* th****h**t hist*** *a* he** *s see h** i**i*i**a*s
a********ities ha*e *e*se*e*e* th****h t***h ti*es a** i*t* a
```

```
**i*hte***t**e.
```

Reason for mapping v to i:

- In the 1st line we get a meaningful word **is** if we map v to i
- The word **it** is also a meaningful word if we map v to i

1. **Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r'}

```
Decrypted text:
the **it* is stre**th essa* e***ai*s that the ***er **
*rie**shi* he**s *s t* *e str*** *r** har*shi*s. *rie**s *a*
*i*e *s the stre**th *e *ee**he* *i*e *ets t***h. i*
a**iti**, *e***e e**ress **it* thr***h***e, *a*i**, *rie**s, a**
*thers share
st*ries ** *i**i** *******r**** *ith s**e**e. there are *a** *e*e*its
** sta*i** **ite* i*t***h ti*es, as it he**s t* *a*e
*ha**e**i** sit*ati**s *ith***ra*e. the i***rta**e ** sta*i** **ite*
has str***
a *h*r* a*****a** *e***e thr***h**t hist*r*. *ee*i** *e**ries ** *hat
*e ha*ea******ishe* thr***h**t hist*r* *a* he** *s see h** i**i*i**a*s
a********ities ha*e *erse*ere* thr***h t***h ti*es a** i*t* a
*ri*hter**t*re.
```

Reason for mapping e to r:

- In the 5th line we get a meaningful word **there** if we map e to r

1. **Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w'}

```
Decrypted text:
the **it* is stre**th essa* e***ai*s that the **wer **
*rie**shi* he**s *s t* *e str*** *r** har*shi*s. *rie**s *a*
*i*e *s the stre**th we *ee*whe* *i*e *ets t***h. i*
a**iti**, *e***e e**ress **it* thr***h***e, *a*i**, *rie**s, a**
*thers share
st*ries ** *i**i** *******r**** with s**e**e. there are *a** *e*e*its
** sta*i** **ite* i*t***h ti*es, as it he**s t* *a*e
*ha**e**i** sit*ati**s with***ra*e. the i***rta**e ** sta*i** **ite*
has str***
a *h*r* a*****a** *e***e thr***h**t hist*r*. *ee*i** *e**ries ** what
we ha*ea******ishe* thr***h**t hist*r* *a* he** *s see h*w i**i*i**a*s
a********ities ha*e *erse*ere* thr***h t***h ti*es a** i*t* a
*ri*hter**t*re.
```

Reason for mapping j to w:

- In the 9th line we get a meaningful word **what** if we map j to w
- In the 11th line we get a meaningful word **with** if we map j to w

1. **Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g'}

```
Decrypted text:
the *nit* is strength essa* e***ains that the **wer **
*rien*shi* he**s *s t* *e str*ng *r** har*shi*s. *rien*s *an
gi*e *s the strength we nee*when *i*e gets t**gh. in
a**iti*n, *e***e e**ress *nit* thr**gh***e, *a*i**, *rien*s, an*
*thers share
st*ries ** *in*ing *****ngr**n* with s**e*ne. there are *an* *ene*its
** sta*ing *nite* int**gh ti*es, as it he**s t* *a*e
*ha**enging sit*ati*ns with***rage. the i***rtan*e ** sta*ing *nite*
has str***
a *h*r* a**ng*an* *e***e thr**gh**t hist*r*. *ee*ing *e**ries ** what
we ha*ea******ishe* thr**gh**t hist*r* *an he** *s see h*w in*i*i**a*s
an******nities ha*e *erse*ere* thr**gh t**gh ti*es an* int* a
*righter**t*re.
```

Reason for mapping s to n and u to g:

- In the 1st line we get a meaningful word **strength** if we map s to n and u to g. I assume that the word may be **strength**

1. **Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o'}

```
Decrypted text:
the *nit* is strength essa* e***ains that the *ower o*
*rien*shi* he**s *s to *e strong *ro* har*shi*s. *rien*s *an
gi*e *s the strength we nee*when *i*e gets to*gh. in
a**ition, *eo**e e**ress *nit* thro*gh*o*e, *a*i**, *rien*s, an*
others share
stories o* *in*ing *o**ongro*n* with so*eone. there are *an* *ene*its
o* sta*ing *nite* into*gh ti*es, as it he**s to *a*e
*ha**enging sit*ations with*o*rage. the i**ortan*e o* sta*ing *nite*
has str***
a *hor* a*ong*an* *eo**e thro*gho*t histor*. *ee*ing *e*ories o* what
we ha*ea**o***ishe* thro*gho*t histor* *an he** *s see how in*i*i**a*s
an**o***nities ha*e *erse*ere* thro*gh to*gh ti*es an* into a
*righter**t*re.
```

Reason for mapping h to o:

- In the last line we get a meaningful word **how** if we map h to o
- In the 5th line we get a meaningful word **other** if we map h to o

11 **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm'}

```
Decrypted text:
the *nit* is strength essa* e***ains that the *ower o*
*rien*shi* he**s *s to *e strong *rom har*shi*s. *rien*s *an
gi*e *s the strength we nee*when *i*e gets to*gh. in
```

```
a**ition, *eo**e e**ress *nit* thro*gh*o*e, *ami**, *rien*s, an*
others share
stories o* *in*ing *ommongro*n* with someone. there are man* *ene*its
o* sta*ing *nite* into*gh times, as it he**s to *a*e
*ha**enging sit*ations with*o*rage. the im*ortan*e o* sta*ing *nite*
has str***
a *hor* amongman* *eo**e thro*gho*t histor*. *ee*ing memories o* what
we ha*ea**om**ishe* thro*gho*t histor* *an he** *s see how in*i*i**a*s
an**omm*nities ha*e *erse*ere* thro*gh to*gh times an* into a
*righter**t*re.
```

Reason for mapping z to m:

- we get __so*eone__ to **someone** if we map z to m

Current mapping:

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f'}

```
Decrypted text:
the *nit* is strength essa* e***ains that the *ower of
frien*shi* he**s *s to *e strong from har*shi*s. frien*s *an
gi*e *s the strength we nee*when *ife gets to*gh. in
a**ition, *eo**e e**ress *nit* thro*gh*o*e, fami**, frien*s, an*
others share
stories of fin*ing *ommongro*n* with someone. there are man* *enefits
of sta*ing *nite* into*gh times, as it he**s to fa*e
*ha**enging sit*ations with*o*rage. the im*ortan*e of sta*ing *nite*
has str***
a *hor* amongman* *eo**e thro*gho*t histor*. *ee*ing memories of what
we ha*ea**om**ishe* thro*gho*t histor* *an he** *s see how in*i*i**a*s
an**omm*nities ha*e *erse*ere* thro*gh to*gh times an* into a
*righterf*t*re.
```

Reason for mapping a to f:

- we get a valid two letter word **of** if we map a to f

1. **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p'}

```
Decrypted text:
the *nit* is strength essa* e*p*ains that the power of
frien*ship he*ps *s to *e strong from har*ships. frien*s *an
gi*e *s the strength we nee*when *ife gets to*gh. in
a**ition, peop*e e*press *nit* thro*gh*o*e, fami**, frien*s, an*
others share
stories of fin*ing *ommongro*n* with someone. there are man* *enefits
of sta*ing *nite* into*gh times, as it he*ps to fa*e
*ha**enging sit*ations with*o*rage. the importan*e of sta*ing *nite*
```

```
has str***
a *hor* amongman* peop*e thro*gho*t histor*. *eeping memories of what
we ha*ea**omp*ishe* thro*gho*t histor* *an he*p *s see how in*i*i**a*s
an**omm*nities ha*e perse*ere* thro*gh to*gh times an* into a
*righterf*t*re.
```

Reason for mapping c to p:

- we get __*ower of__ to **power of** if we map c to p

1. **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd'}

```
Decrypted text:
the *nit* is strength essa* e*p*ains that the power of
friendship he*ps *s to *e strong from hardships. friends *an
gi*e *s the strength we needwhen *ife gets to*gh. in
addition, peop*e e*press *nit* thro*gh*o*e, fami**, friends, and
others share
stories of finding *ommongro*nd with someone. there are man* *enefits
of sta*ing *nited into*gh times, as it he*ps to fa*e
*ha**enging sit*ations with*o*rage. the importan*e of sta*ing *nited
has str***
a *hord amongman* peop*e thro*gho*t histor*. *eeping memories of what
we ha*ea**omp*ished thro*gho*t histor* *an he*p *s see how indi*id*a*s
and*omm*nities ha*e perse*ered thro*gh to*gh times and into a
*righterf*t*re.
```

Reason for mapping x to d:

- we get __*nited__ to **united** if we map x to d
- we get friendship, hardships, and addition if we map x to d

Current mapping:

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l'}

```
Decrypted text:
the *nit* is strength essa* explains that the power of
friendship helps *s to *e strong from hardships. friends *an
gi*e *s the strength we needwhen life gets to*gh. in
addition, people express *nit* thro*ghlo*e, famil*, friends, and
others share
stories of finding *ommongro*nd with someone. there are man* *enefits
of sta*ing *nited into*gh times, as it helps to fa*e
*hallenging sit*ations with*o*rage. the importan*e of sta*ing *nited
has str***
a *hord amongman* people thro*gho*t histor*. *eeping memories of what
```

```
we ha*ea**omplished thro*gho*t histor* *an help *s see how indi*id*als
and*omm*nities ha*e perse*ered thro*gh to*gh times and into a
*righterf*t*re.
```

Reason for mapping k to x and y to l:

- I assume that the word **e*p*ains** may be **explains** if we map k to x and y to l

1. **Current mapping:** {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l', 'b': 'u'}

```
Decrypted text:
the unit* is strength essa* explains that the power of
friendship helps us to *e strong from hardships. friends *an
gi*e us the strength we needwhen life gets tough. in
addition, people express unit* throughlo*e, famil*, friends, and
others share
stories of finding *ommonground with someone. there are man* *enefits
of sta*ing united intough times, as it helps to fa*e
*hallenging situations with*ourage. the importan*e of sta*ing united
has stru**
a *hord amongman* people throughout histor*. *eeping memories of what
we ha*ea**omplished throughout histor* *an help us see how indi*iduals
and*ommunities ha*e perse*ered through tough times and into a
*righterfuture.
```

Reason for mapping b to u:

- we get through tough times if we map b to u
- we get united if we map b to u

Current mapping:

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l', 'b': 'u', 'l': 'y'}

```
Decrypted text:
the unity is strength essay explains that the power of
friendship helps us to *e strong from hardships. friends *an
gi*e us the strength we needwhen life gets tough. in
addition, people express unity throughlo*e, family, friends, and
others share
stories of finding *ommonground with someone. there are many *enefits
of staying united intough times, as it helps to fa*e
*hallenging situations with*ourage. the importan*e of staying united
has stru**
a *hord amongmany people throughout history. *eeping memories of what
we ha*ea**omplished throughout history *an help us see how indi*iduals
and*ommunities ha*e perse*ered through tough times and into a
*righterfuture.
```

Reason for mapping l to y:

- we get many if we map l to y
- we get unity,eassay if we map l to y in the 1st line
- we get family if we map l to y
- we get history if we map l to y

1. **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l', 'b': 'u', 'l': 'y', 'i': 'b'}

```
Decrypted text:
the unity is strength essay explains that the power of
friendship helps us to be strong from hardships. friends *an
gi*e us the strength we needwhen life gets tough. in
addition, people express unity throughlo*e, family, friends, and
others share
stories of finding *ommonground with someone. there are many benefits
of staying united intough times, as it helps to fa*e
*hallenging situations with*ourage. the importan*e of staying united
has stru**
a *hord amongmany people throughout history. *eeping memories of what
we ha*ea**omplished throughout history *an help us see how indi*iduals
and*ommunities ha*e perse*ered through tough times and into a
brighterfuture.
```

Reason for mapping i to b:

- we get benefits if we map i to b
- we brigherfuture if we map i to b

1. **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l', 'b': 'u', 'l': 'y', 'i': 'b', 'o': 'v'}

```
Decrypted text:
the unity is strength essay explains that the power of
friendship helps us to be strong from hardships. friends *an
give us the strength we needwhen life gets tough. in
addition, people express unity throughlove, family, friends, and
others share
stories of finding *ommonground with someone. there are many benefits
of staying united intough times, as it helps to fa*e
*hallenging situations with*ourage. the importan*e of staying united
has stru**
a *hord amongmany people throughout history. *eeping memories of what
we havea**omplished throughout history *an help us see how individuals
and*ommunities have persevered through tough times and into a
brighterfuture.
```

Reason for mapping o to v:

- we get love if we map o to v
- we get over if we map o to v
- we get have if we map o to v

Current mapping: {'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l', 'b': 'u', 'l': 'y', 'i': 'b', 'o': 'v', 'q': 'c'}

```
Decrypted text:
the unity is strength essay explains that the power of
friendship helps us to be strong from hardships. friends can
give us the strength we needwhen life gets tough. in
addition, people express unity throughlove, family, friends, and
others share
stories of finding commonground with someone. there are many benefits
of staying united intough times, as it helps to face
challenging situations withcourage. the importance of staying united
has struc*
a chord amongmany people throughout history. *eeping memories of what
we haveaccomplished throughout history can help us see how individuals
andcommunities have persevered through tough times and into a
brighterfuture.
```

Reason for mapping q to c:

- we get commonground if we map q to c
- we get face if we map q to c
- we get challenging if we map q to c
- we get chord if we map q to c
- we get accompolished if we map q to c

1. **Current mapping:**

{'m': 'a', 'd': 'e', 'f': 's', 'g': 't', 't': 'h', 'v': 'i', 'e': 'r', 'j': 'w', 's': 'n', 'u': 'g', 'h': 'o', 'z': 'm', 'a': 'f', 'c': 'p', 'x': 'd', 'k': 'x', 'y': 'l', 'b': 'u', 'l': 'y', 'i': 'b', 'o': 'v', 'q': 'c', 'p': 'k'}

```
Decrypted text:
the unity is strength essay explains that the power of
friendship helps us to be strong from hardships. friends can
give us the strength we needwhen life gets tough. in
addition, people express unity throughlove, family, friends, and
others share
stories of finding commonground with someone. there are many benefits
of staying united intough times, as it helps to face
challenging situations withcourage. the importance of staying united
has struck
a chord amongmany people throughout history. keeping memories of what
we haveaccomplished throughout history can help us see how individuals
```

```
andcommunities have persevered through tough times and into a
brighterfuture.
```

Reason for mapping p to k:

- we get struck if we map p to k
- we get **keeping** from __*eeping__ if we map p to k

Deciphered Text:

```
the unity is strength essay explains that the power of friendship
helps us to be
strong from hardships. friends can give us the strength we need when
life gets
tough. in addition, people express unity through love,family, friends,
and others
share stories of finding common ground with someone. there are many
benefits of
staying united in tough times, as it helps to face challenging
situations with courage.
the importance of staying united has struck a chord among many people
throughout history.keeping memories of what we have accomplished
throughout
history can help us see how individuals and communities have
persevered through
tough times and into a brighter future.
```

Final Mapping:

Encrypted Char	Decrypted Char	Encrypted Char	Decrypted Char	Encrypted Char	Decrypted Char	Encrypted Char	Decrypted Char
m	a	d	e	f	s	g	t
t	h	v	i	e	r	j	w
s	n	u	g	h	o	z	m
a	f	c	p	x	d	k	x
y	l	b	u	l	y	i	b
o	v	q	c	p	k		

Lab Task 2(b)

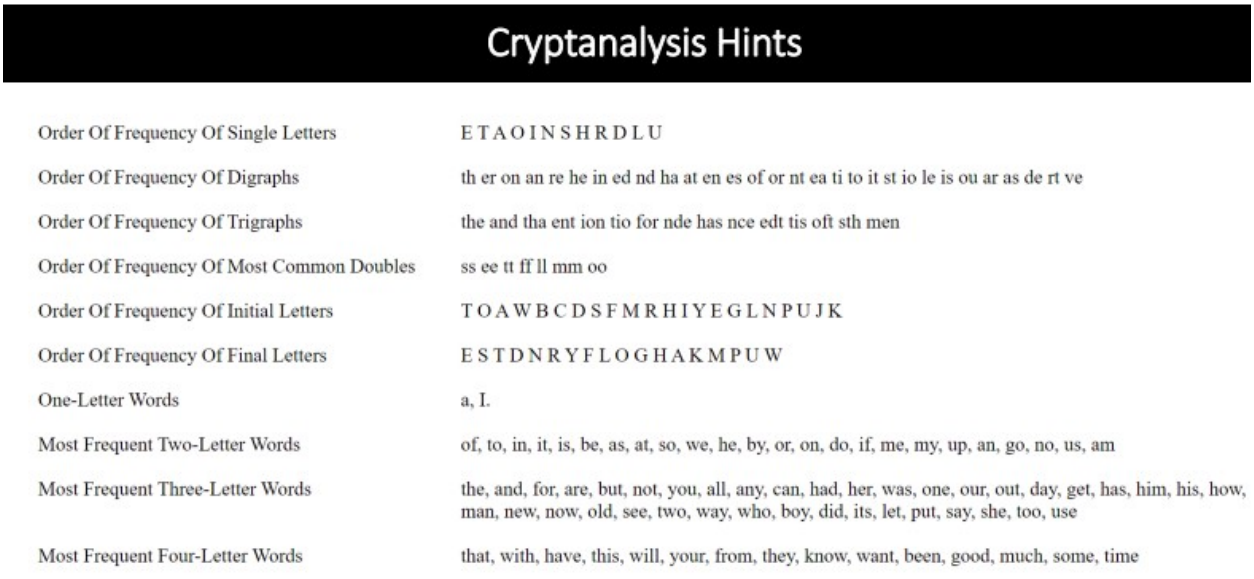
```
from collections import Counter
def print_text_with_spaces(text, words_per_line):
    words = text.split()
    for i in range(0, len(words), words_per_line):
        print(" ".join(words[i:i+words_per_line]))


def decrypt(text, mapping):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            if char in mapping:
                decrypted_text += mapping[char]
            else:
                decrypted_text += "*"
        else:
            decrypted_text += char  # Preserve punctuation
    return decrypted_text

def analyze_frequency(ciphertext):
    # Remove spaces and punctuation
    ciphertext = ''.join(char for char in ciphertext if
char.isalpha())
    # Count the frequency of each letter
    frequency = Counter(ciphertext)
    # Sort the letters by frequency
    sorted_frequency = sorted(frequency.items(), key=lambda x: x[1],
reverse=True)
    return sorted_frequency

def algo(text, mapping, words_per_line):
    while True:
        print("Current mapping:")
        print(mapping)
        reverse_mapping = {}
        found_duplicate = False
        for key, value in mapping.items():
          if value in reverse_mapping:
            print(f"{reverse_mapping[value]} maps to the same value as
{key}: {value}")
            found_duplicate = True
            break
          else:
            reverse_mapping[value] = key
        if found_duplicate:
```

```
            break
        frequency_list = analyze_frequency(text)
        decrypted_text = decrypt(text, mapping)
        print("Decrypted text:")
        print_text_with_spaces(decrypted_text, words_per_line)

        print("Frequency list:")
        for char, freq in frequency_list:
            print(f"{char}: {freq}")

        replacement = input("Enter replacement (or 'q' to quit): ")
        if replacement.lower() == 'q':
            break
        if len(replacement) != 2 or replacement[0] not in
'abcdefghijklmnopqrstuvwxyz' or replacement[1] not in
'abcdefghijklmnopqrstuvwxyz':
            print("Invalid input. Please enter a valid replacement.")
            continue
        if replacement[0] in mapping.values() or replacement[1] in
mapping.values():
            print("Error: One of the replacement characters is already
mapped to another character.")
            continue
        mapping[replacement[0]] = replacement[1]

        # Check if any character is mapped to the same replacement
        duplicate_mapping = [char for char, mapped_char in
mapping.items() if mapped_char == replacement[1] and char !=
replacement[0]]
        if duplicate_mapping:
            print(f"Error: Character '{duplicate_mapping[0]}' is
already mapped to '{replacement[1]}'.")
            del mapping[replacement[0]]  # Rollback the mapping
            continue

text = "exupziu kxwqxagxom, upm gxsm zs l amtwzo exgg rmqzfm kigg \
lok xolquxjm. lgwz, l kxwqxagxomk amtwzo qlo qzoutzg lok plokgm \
upm wxuiluxzo zs gxjxoh xo l wzapxwuxqlumk elc uplo upzwm \
epz kz ozu. fztmzjmt, xs czi pljm l aglo lok \
czi elou uz xfagmfmou xu xo czit gxsm upmo czi \
ommk kxwqxagxom. xu flnmw upxohw mlwc szt czi uz plokgm \
lok iguxflumgc rtxoh wiqqmww uz czit gxsm. xs ulgn lrziu \
upm ucamw zs kxwqxagxom, upmo upmc ltm hmomtlggc zs uez \
ucamw. sxtwu zom xw xokiqmk kxwqxagxom lok upm wmqzok zom \
xw wmgs- kxwqxagxom. xokiqmk kxwqxagxom xw wzfmupxoh uplu zupmtw
ulihpu \
iw zt em gmlto rc wmmxoh zupmtw. epxgm wmgs-kxwqxagxom qzfmw \
stzf exupxo lok em gmlto xu zo zit zeo wmgs. \
wmgs-kxwqxagxom tmyixtmw l gzu zs fzuxjluxzo lok wiaaztu stzf zupmtw.
```

```
\
lrzjm lgg, szggzexoh czit klxgc wqpmkigm exupziu loc fxwulnm xw \
lgwz altu zs rmxoh kxwqxagxomk."

mapping = {'m':'a'}
words_per_line = 10
algo(text, mapping, words_per_line)
```

Frequency table

Character	Frequency	Character	Frequency	Character	Frequency	Character	Frequency
x	74	m	73	z	57	o	55
u	52	w	47	g	44	l	43
k	32	p	26	t	26	i	22
q	22	a	20	s	20	c	17
e	13	f	12	h	9	r	6
j	6	n	3	y	1		



We will the above hints attached to the Lecture slide to solve the problem.

Mapping

Here I am going to map the frequency table to the English alphabet.

1. **Current mapping:**

{'l': 'a'}

```
Decrypted text:
******* **********, *** **** ** a ****** **** ****** ****
a** **a*****. a***, a ********** ****** *a* ******* a** *a****
*** ****a**** ** ****** ** a *********a*** *a* **a* *****
*** ** ***. ********, ** *** *a** a **a* a**
*** *a** ** ********* ** ** **** **** **** ***
**** **********. ** *a*** ****** *a** *** *** ** *a****
a** *****a**** ***** ******* ** **** ****. ** *a** a****
*** ***** ** **********, **** **** a** *****a*** ** ***
*****. ***** *** ** ******* ********** a** *** ****** ***
** ****- **********. ******* ********** ** ********* **a* ******
*a****
** ** ** **a** ** ****** ******. ***** ****-********** *****
**** ****** a** ** **a** ** ** *** *** ****.
****-********** ******** a *** ** *****a**** a** ******* **** ******.
a**** a**, ********* **** *a*** ******** ******* a** ****a** **
a*** *a** ** ***** ***********.
```

Reason for the mapping:

- We know in English lagnunage the one letter word is 'a' and 'I' . But 'I' always comes at the start of the sentence. So, the one letter word is 'a' in this case.

1. **Current mapping:**

{'l': 'a', 'x': 'e'}

```
Decrypted text:
*e***** *e**e**e**, *** *e** ** a ****** *e** ****** ****
a** e*a**e**. a***, a *e**e**e*** ****** *a* ******* a** *a****
*** *e**a*e** ** *e*e** e* a ****e**e*a*** *a* **a* *****
*** ** ***. ********, e* *** *a** a **a* a**
*** *a** ** e******** e* e* **** *e** **** ***
**** *e**e**e**. e* *a*** **e*** *a** *** *** ** *a****
a** ***e*a**** **e** ******* ** **** *e**. e* *a** a****
*** ***** ** *e**e**e**, **** **** a** *****a*** ** ***
*****. *e*** *** e* e****** *e**e**e** a** *** ****** ***
e* ****- *e**e**e**. e****** *e**e**e** e* ******e** **a* ******
*a****
** ** ** **a** ** ***e** ******. **e** ****-*e**e**e** *****
**** *e**e* a** ** **a** e* ** *** *** ****.
****-*e**e**e** ****e*** a *** ** ***e*a*e** a** ******* **** ******.
a**** a**, ******e** **** *ae** ******** *e***** a** *e**a** e*
a*** *a** ** **e** *e**e**e***.
```

Reason for the mapping:

- We know in English lagnunage the most frequent letter is 'e'. So, I have mapped 'x' to 'e'. But I map x to e there come two letters word starting with 'e' and this two words starting with not common. So, I have to change the mapping.

1. **Current mapping:**

{'l': 'a', 'x': 't'}

```
Decrypted text:
*t***** *t**t**t**, *** *t** ** a ****** *t** ****** ****
a** t*a**t**. a***, a *t**t**t*** ****** *a* ******* a** *a****
*** *t**a*t** ** *t*t** t* a ****t**t*a*** *a* **a* *****
*** ** ***. ********, t* *** *a** a **a* a**
*** *a** ** t******** t* t* **** *t** **** ***
**** *t**t**t**. t* *a*** **t*** *a** *** *** ** *a****
a** ***t*a**** **t** ******* ** **** *t**. t* *a** a****
*** ***** ** *t**t**t**, **** **** a** *****a*** ** ***
*****. *t*** *** t* t****** *t**t**t** a** *** ****** ***
t* ****- *t**t**t**. t****** *t**t**t** t* ******t** **a* ******
*a****
** ** ** **a** ** ***t** ******. **t** ****-*t**t**t** *****
**** *t**t* a** ** **a** t* ** *** *** ****.
****-*t**t**t** ****t*** a *** ** ***t*a*t** a** ******* **** ******.
a**** a**, *******t** **** *at** ******** *t***** a** *t**a** t*
a*** *a** ** **t** *t**t**t***.
```

Reason for the mapping:

- We know in English lagnunage the second most frequent letter is 't'. So, I have mapped 'x' to 't'. This mapping is correct because the two letters word starting with 't' and this two words starting with common.

1. **Current mapping:**

{'l': 'a', 'x': 't', 'm': 'e'}

```
Decrypted text:
*t***** *t**t**t*e, **e *t*e ** a *e**** *t** *e***e ****
a** t*a**t*e. a***, a *t**t**t*e* *e**** *a* ******* a** *a***e
**e *t**a*t** ** *t*t** t* a ****t**t*a*e* *a* **a* ****e
*** ** ***. ***e**e*, t* *** *a*e a **a* a**
*** *a** ** t***e*e** t* t* **** *t*e **e* ***
*ee* *t**t**t*e. t* *a*e* **t*** ea** *** *** ** *a***e
a** ***t*a*e** **t** ****e** ** **** *t*e. t* *a** a****
**e ***e* ** *t**t**t*e, **e* **e* a*e *e*e*a*** ** ***
***e*. *t*** **e t* t****e* *t**t**t*e a** **e *e**** **e
t* *e**- *t**t**t*e. t****e* *t**t**t*e t* ***e**t** **a* ***e**
*a****
** ** *e *ea** ** *eet** ***e**. **t*e *e**-*t**t**t*e ***e*
**** *t**t* a** *e *ea** t* ** *** *** *e**.
*e**-*t**t**t*e *e**t*e* a *** ** ***t*a*t** a** ******* **** ***e**.
a***e a**, *******t** **** *at** ***e***e *t***** a** *t**a*e t*
a*** *a** ** *et** *t**t**t*e*.
```

Reason for the mapping:

- In the cipher text 'm' is the second most frequent letters. And 'e' is the most frequent letter to any text. As we don't map 'e' with the most frequent letter in the cipher text(the reason stated above), we can map 'm' to 'e'. This one of the probable mappings.

1. **Current mapping:**

{'l': 'a', 'x': 't', 'm': 'e', 'z': 'o'}

```
Decrypted text:
*t**o** *t**t**t*e, **e *t*e o* a *e**o* *t** *e*o*e ****
a** t*a**t*e. a**o, a *t**t**t*e* *e**o* *a* *o***o* a** *a***e
**e *t**a*to* o* *t*t** t* a *o**t**t*a*e* *a* **a* **o*e
**o *o *o*. *o*eo*e*, t* *o* *a*e a **a* a**
*o* *a** *o t***e*e** t* t* *o** *t*e **e* *o*
*ee* *t**t**t*e. t* *a*e* **t*** ea** *o* *o* *o *a***e
a** ***t*a*e** **t** ****e** *o *o** *t*e. t* *a** a*o**
**e ***e* o* *t**t**t*e, **e* **e* a*e *e*e*a*** o* **o
***e*. *t*** o*e t* t****e* *t**t**t*e a** **e *e*o** o*e
t* *e**- *t**t**t*e. t****e* *t**t**t*e t* *o*e**t** **a* o**e**
*a****
** o* *e *ea** ** *eet** o**e**. **t*e *e**-*t**t**t*e *o*e*
**o* *t**t* a** *e *ea** t* o* o** o** *e**.
*e**-*t**t**t*e *e**t*e* a *o* o* *o*t*a*to* a** ****o** **o* o**e**.
a*o*e a**, *o**o*t** *o** *at** ***e***e *t**o** a** *t**a*e t*
a**o *a** o* *et** *t**t**t*e*.
```

Reason for the mapping:

- As the 'z' is the third most frequent letter in the cipher text, I have mapped 'z' to 'o'. This is one of the probable mappings.

1. **Current mapping:**

{'l': 'a', 'x': 't', 'm': 'e', 'z': 'o', 'o': 'i'}

```
Decrypted text:
*t**o** *t**t**tie, **e *t*e o* a *e**oi *t** *e*o*e ****
ai* tia**t*e. a**o, a *t**t**tie* *e**oi *ai *oi**o* ai* *ai**e
**e *t**a*toi o* *t*ti* ti a *o**t**t*a*e* *a* **ai **o*e
**o *o io*. *o*eo*e*, t* *o* *a*e a **ai ai*
*o* *ai* *o t***e*ei* t* ti *o** *t*e **ei *o*
iee* *t**t**tie. t* *a*e* **ti** ea** *o* *o* *o *ai**e
ai* ***t*a*e** **ti* ****e** *o *o** *t*e. t* *a** a*o**
**e ***e* o* *t**t**tie, **ei **e* a*e *eie*a*** o* **o
***e*. *t*** oie t* ti***e* *t**t**tie ai* **e *e*oi* oie
t* *e**- *t**t**tie. ti***e* *t**t**tie t* *o*e**ti* **a* o**e**
*a****
** o* *e *ea*i ** *eeti* o**e**. **t*e *e**-*t**t**tie *o*e*
**o* *t**ti ai* *e *ea*i t* oi o** o*i *e**.
*e**-*t**t**tie *e**t*e* a *o* o* *o*t*a*toi ai* ****o** **o* o**e**.
a*o*e a**, *o**o*ti* *o** *at** ***e***e *t**o** ai* *t**a*e t*
a**o *a** o* *eti* *t**t**tie*.
```

Reason for the mapping:

- As the 'o' is the fourth most frequent letter in the cipher text, I have mapped 'o' to 'i'. But there is pitfall. A two letters word such as **oi**" has been formed, which is not a common word in English. So, I have to change the mapping.

1. **Current mapping:** {'l': 'a', 'x': 't', 'm': 'e', 'z': 'o', 'o': 'n'}

```
Decrypted text:
*t**o** *t**t**tne, **e *t*e o* a *e**on *t** *e*o*e ****
an* tna**t*e. a**o, a *t**t**tne* *e**on *an *on**o* an* *an**e
**e *t**a*ton o* *t*tn* tn a *o**t**t*a*e* *a* **an **o*e
**o *o no*. *o*eo*e*, t* *o* *a*e a **an an*
*o* *an* *o t***e*en* t* tn *o** *t*e **en *o*
nee* *t**t**tne. t* *a*e* **tn** ea** *o* *o* *o *an**e
an* ***t*a*e** **tn* ****e** *o *o** *t*e. t* *a** a*o**
**e ***e* o* *t**t**tne, **en **e* a*e *ene*a*** o* **o
***e*. *t*** one t* tn***e* *t**t**tne an* **e *e*on* one
t* *e**- *t**t**tne. tn***e* *t**t**tne t* *o*e**tn* **a* o**e**
*a****
** o* *e *ea*n ** *eetn* o**e**. **t*e *e**-*t**t**tne *o*e*
**o* *t**tn an* *e *ea*n t* on o** o*n *e**.
*e**-*t**t**tne *e**t*e* a *o* o* *o*t*a*ton an* ****o** **o* o**e**.
a*o*e a**, *o**o*tn* *o** *at** ***e***e *t**o** an* *t**a*e t*
a**o *a** o* *etn* *t**t**tne*.
```

Reason for the mapping:

- There is another two word letter **tn** which is not a common word in English. So, I have to change the x to t. Rather if I map x to i I may get a valid word **in** at the end of the sentence. So, I have to map x to i. In the next cell I will map x to i.

1. **Current mapping:** {'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n'}

```
Decrypted text:
*i**o** *i**i**ine, **e *i*e o* a *e**on *i** *e*o*e ****
an* ina**i*e. a**o, a *i**i**ine* *e**on *an *on**o* an* *an**e
**e *i**a*ion o* *i*in* in a *o**i**i*a*e* *a* **an **o*e
**o *o no*. *o*eo*e*, i* *o* *a*e a **an an*
*o* *an* *o i***e*en* i* in *o** *i*e **en *o*
nee* *i**i**ine. i* *a*e* **in** ea** *o* *o* *o *an**e
an* ***i*a*e** **in* ****e** *o *o** *i*e. i* *a** a*o**
**e ***e* o* *i**i**ine, **en **e* a*e *ene*a*** o* **o
***e*. *i*** one i* in***e* *i**i**ine an* **e *e*on* one
i* *e**- *i**i**ine. in***e* *i**i**ine i* *o*e**in* **a* o**e**
*a****
** o* *e *ea*n ** *eein* o**e**. **i*e *e**-*i**i**ine *o*e*
**o* *i**in an* *e *ea*n i* on o** o*n *e**.
*e**-*i**i**ine *e**i*e* a *o* o* *o*i*a*ion an* ****o** **o* o**e**.
a*o*e a**, *o**o*in* *o** *ai** ***e***e *i**o** an* *i**a*e i*
a**o *a** o* *ein* *i**i**ine*.
```

Reason for the mapping:

- Now the current mapping seems to be correct. So, I will continue with the next letter.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f'}

```
Decrypted text:
*i**o** *i**i**ine, **e *ife of a *e**on *i** *e*o*e ****
an* ina**i*e. a**o, a *i**i**ine* *e**on *an *on**o* an* *an**e
**e *i**a*ion of *i*in* in a *o**i**i*a*e* *a* **an **o*e
**o *o no*. *o*eo*e*, if *o* *a*e a **an an*
*o* *an* *o i***e*en* i* in *o** *ife **en *o*
nee* *i**i**ine. i* *a*e* **in** ea** fo* *o* *o *an**e
an* ***i*a*e** **in* ****e** *o *o** *ife. if *a** a*o**
**e ***e* of *i**i**ine, **en **e* a*e *ene*a*** of **o
***e*. fi*** one i* in***e* *i**i**ine an* **e *e*on* one
i* *e*f- *i**i**ine. in***e* *i**i**ine i* *o*e**in* **a* o**e**
*a****
** o* *e *ea*n ** *eein* o**e**. **i*e *e*f-*i**i**ine *o*e*
f*o* *i**in an* *e *ea*n i* on o** o*n *e*f.
*e*f-*i**i**ine *e**i*e* a *o* of *o*i*a*ion an* ****o** f*o* o**e**.
a*o*e a**, fo**o*in* *o** *ai** ***e***e *i**o** an* *i**a*e i*
a**o *a** of *ein* *i**i**ine*.
```

Reason for the mapping:

- By mapping 's' to 'f' I have formed a valid word **of**. So, the mapping is correct.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l'}

```
Decrypted text:
*i**o** *i**i*line, **e life of a *e**on *ill *e*o*e **ll
an* ina**i*e. al*o, a *i**i*line* *e**on *an *on**ol an* *an*le
**e *i**a*ion of li*in* in a *o**i**i*a*e* *a* **an **o*e
**o *o no*. *o*eo*e*, if *o* *a*e a *lan an*
*o* *an* *o i**le*en* i* in *o** life **en *o*
nee* *i**i*line. i* *a*e* **in** ea** fo* *o* *o *an*le
an* *l*i*a*el* **in* ****e** *o *o** life. if *al* a*o**
**e ***e* of *i**i*line, **en **e* a*e *ene*all* of **o
***e*. fi*** one i* in***e* *i**i*line an* **e *e*on* one
i* *elf- *i**i*line. in***e* *i**i*line i* *o*e**in* **a* o**e**
*a****
** o* *e lea*n ** *eein* o**e**. **ile *elf-*i**i*line *o*e*
f*o* *i**in an* *e lea*n i* on o** o*n *elf.
*elf-*i**i*line *e**i*e* a lo* of *o*i*a*ion an* ****o** f*o* o**e**.
a*o*e all, follo*in* *o** *ail* ***e**le *i**o** an* *i**a*e i*
al*o *a** of *ein* *i**i*line*.
```

Reason for the mapping:

- By mapping 'g' to 'l' I have formed a valid word **life**. So, the mapping is seemed to be correct. So, I will continue with the next letter.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's'}

```
Decrypted text:
*i**o** *is*i*line, **e life of a *e*son *ill *e*o*e **ll
an* ina**i*e. also, a *is*i*line* *e*son *an *on**ol an* *an*le
**e si**a*ion of li*in* in a so**is*i*a*e* *a* **an **ose
**o *o no*. *o*eo*e*, if *o* *a*e a *lan an*
*o* *an* *o i**le*en* i* in *o** life **en *o*
nee* *is*i*line. i* *a*es **in*s eas* fo* *o* *o *an*le
an* *l*i*a*el* **in* s***ess *o *o** life. if *al* a*o**
**e ***es of *is*i*line, **en **e* a*e *ene*all* of **o
***es. fi*s* one is in***e* *is*i*line an* **e se*on* one
is self- *is*i*line. in***e* *is*i*line is so*e**in* **a* o**e*s
*a****
*s o* *e lea*n ** seein* o**e*s. **ile self-*is*i*line *o*es
f*o* *i**in an* *e lea*n i* on o** o*n self.
self-*is*i*line *e**i*es a lo* of *o*i*a*ion an* s***o** f*o* o**e*s.
a*o*e all, follo*in* *o** *ail* s**e**le *i**o** an* *is*a*e is
also *a** of *ein* *is*i*line*.
```

Reason for the mapping:

- By mapping 'w' to 's' I have formed a valid word **also**. So, the mapping is seemed to be correct. So, I will continue with the next letter.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r'}

```
Decrypted text:
*i**o** *is*ipline, **e life of a person *ill *e*o*e **ll
an* ina**i*e. also, a *is*ipline* person *an *on*rol an* *an*le
**e si**a*ion of li*in* in a sop*is*i*a*e* *a* **an **ose
**o *o no*. *oreo*er, if *o* *a*e a plan an*
*o* *an* *o i*ple*en* i* in *o*r life **en *o*
nee* *is*ipline. i* *a*es **in*s eas* for *o* *o *an*le
an* *l*i*a*el* *rin* s***ess *o *o*r life. if *al* a*o**
**e **pes of *is*ipline, **en **e* are *enerall* of **o
**pes. firs* one is in***e* *is*ipline an* **e se*on* one
is self- *is*ipline. in***e* *is*ipline is so*e**in* **a* o**ers
*a****
*s or *e learn ** seein* o**ers. **ile self-*is*ipline *o*es
fro* *i**in an* *e learn i* on o*r o*n self.
self-*is*ipline re**ires a lo* of *o*i*a*ion an* s*ppor* fro* o**ers.
a*o*e all, follo*in* *o*r *ail* s**e**le *i**o** an* *is*a*e is
also par* of *ein* *is*ipline*.
```

Reason for the mapping:

- I have assumed that the word ***e*son** is **person**. So, I have mapped 'a' to 'p' and 't' to 'r'. Let's see if the mapping is correct.

12 **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c'}

```
Decrypted text:
*i**o** discipline, **e life of a person *ill *eco*e d*ll
and inac*i*e. also, a disciplined person can con*rol and *andle
**e si**a*ion of li*in* in a sop*is*ica*ed *a* **an **ose
**o do no*. *oreo*er, if *o* *a*e a plan and
*o* *an* *o i*ple*en* i* in *o*r life **en *o*
need discipline. i* *a*es **in*s eas* for *o* *o *andle
and *l*i*a*el* *rin* s*ccess *o *o*r life. if *al* a*o**
**e **pes of discipline, **en **e* are *enerall* of **o
**pes. firs* one is ind*ced discipline and **e second one
is self- discipline. ind*ced discipline is so*e**in* **a* o**ers
*a****
*s or *e learn ** seein* o**ers. **ile self-discipline co*es
fro* *i**in and *e learn i* on o*r o*n self.
self-discipline re**ires a lo* of *o*i*a*ion and s*ppor* fro* o**ers.
a*o*e all, follo*in* *o*r dail* sc*ed*le *i**o** an* *is*a*e is
also par* of *ein* disciplined.
```

Reason for the mapping:

- By mapping 'k' to 'd' and 'q' to 'c' I have formed a valid word **discipline**. So, the mapping is seemed to be correct. So, I will continue with the next letter.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w'}

```
Decrypted text:
wi**o** discipline, **e life of a person will *eco*e d*ll
and inac*i*e. also, a disciplined person can con*rol and *andle
**e si**a*ion of li*in* in a sop*is*ica*ed wa* **an **ose
w*o do no*. *oreo*er, if *o* *a*e a plan and
*o* wan* *o i*ple*en* i* in *o*r life **en *o*
need discipline. i* *a*es **in*s eas* for *o* *o *andle
and *l*i*a*el* *rin* s*ccess *o *o*r life. if *al* a*o**
**e **pes of discipline, **en **e* are *enerall* of *wo
**pes. firs* one is ind*ced discipline and **e second one
is self- discipline. ind*ced discipline is so*e**in* **a* o**ers
*a****
*s or we learn ** seein* o**ers. w*ile self-discipline co*es
fro* wi**in and we learn i* on o*r own self.
self-discipline re**ires a lo* of *o*i*a*ion and s*ppor* fro* o**ers.
```

```
a*o*e all, followin* *o*r dail* sc*ed*le wi**o** an* *is*a*e is
also par* of *ein* disciplined.
```

Reason for the mapping:

- By mapping 'e' to 'w' I have formed a valid word **will**. So, the mapping is seemed to be correct. So, I will continue with the next letter.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm'}

```
Decrypted text:
wi**o** discipline, **e life of a person will become d*ll
and inac*i*e. also, a disciplined person can con*rol and *andle
**e si**a*ion of li*in* in a sop*is*ica*ed wa* **an **ose
w*o do no*. moreo*er, if *o* *a*e a plan and
*o* wan* *o implemen* i* in *o*r life **en *o*
need discipline. i* ma*es **in*s eas* for *o* *o *andle
and *l*ima*el* brin* s*ccess *o *o*r life. if *al* abo**
**e **pes of discipline, **en **e* are *enerall* of *wo
**pes. firs* one is ind*ced discipline and **e second one
is self- discipline. ind*ced discipline is some**in* **a* o**ers
*a****
*s or we learn b* seein* o**ers. w*ile self-discipline comes
from wi**in and we learn i* on o*r own self.
self-discipline re**ires a lo* of mo*i*a*ion and s*ppor* from o**ers.
abo*e all, followin* *o*r dail* sc*ed*le wi**o** an* mis*a*e is
also par* of bein* disciplined.
```

Reason for the mapping:

- By mapping 'r' to 'b' and 'f' to 'm' I have formed a valid word **become**. So, the mapping is seemed to be correct. So, I will continue with the next letter.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h'}

```
Decrypted text:
wi*ho** discipline, *he life of a person will become d*ll
and inac*i*e. also, a disciplined person can con*rol and handle
*he si**a*ion of li*in* in a sophis*ica*ed wa* *han *hose
who do no*. moreo*er, if *o* ha*e a plan and
*o* wan* *o implemen* i* in *o*r life *hen *o*
need discipline. i* ma*es *hin*s eas* for *o* *o handle
and *l*ima*el* brin* s*ccess *o *o*r life. if *al* abo**
*he **pes of discipline, *hen *he* are *enerall* of *wo
**pes. firs* one is ind*ced discipline and *he second one
```

```
is self- discipline. ind*ced discipline is some*hin* *ha* o*hers
*a**h*
*s or we learn b* seein* o*hers. while self-discipline comes
from wi*hin and we learn i* on o*r own self.
self-discipline re**ires a lo* of mo*i*a*ion and s*ppor* from o*hers.
abo*e all, followin* *o*r dail* sched*le wi*ho** an* mis*a*e is
also par* of bein* disciplined.
```

Reason for the mapping:

- By mapping 'p' to 'h' I have formed a valid word **the**.
- By mapping 'g' to 'l' I have formed a valid word **handle**.

1. **Current mapping:** {'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h', 'u': 't'}

```
Decrypted text:
witho*t discipline, the life of a person will become d*ll
and inacti*e. also, a disciplined person can control and handle
the sit*ation of li*in* in a sophisticated wa* than those
who do not. moreo*er, if *o* ha*e a plan and
*o* want to implement it in *o*r life then *o*
need discipline. it ma*es thin*s eas* for *o* to handle
and *ltimatel* brin* s*ccess to *o*r life. if tal* abo*t
the t*pes of discipline, then the* are *enerall* of two
t*pes. first one is ind*ced discipline and the second one
is self- discipline. ind*ced discipline is somethin* that others
ta**ht
*s or we learn b* seein* others. while self-discipline comes
from within and we learn it on o*r own self.
self-discipline re**ires a lot of moti*ation and s*pport from others.
abo*e all, followin* *o*r dail* sched*le witho*t an* mista*e is
also part of bein* disciplined.
```

Reason for the mapping:

- After this mapping to u to t, I am sure that I am in the right way. Now, I can move till the end of the text.

17.__Current mapping:__

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h', 'u': 't', 'i': 'u'}

```
Decrypted text:
without discipline, the life of a person will become dull
and inacti*e. also, a disciplined person can control and handle
the situation of li*in* in a sophisticated wa* than those
who do not. moreo*er, if *ou ha*e a plan and
*ou want to implement it in *our life then *ou
need discipline. it ma*es thin*s eas* for *ou to handle
```

```
and ultimatel* brin* success to *our life. if tal* about
the t*pes of discipline, then the* are *enerall* of two
t*pes. first one is induced discipline and the second one
is self- discipline. induced discipline is somethin* that others
tau*ht
us or we learn b* seein* others. while self-discipline comes
from within and we learn it on our own self.
self-discipline re*uires a lot of moti*ation and support from others.
abo*e all, followin* *our dail* schedule without an* mista*e is
also part of bein* disciplined.
```

Reason for the mapping:

- By mapping 'i' to 'u' I have formed a valid word **our**.

18 **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h', 'u': 't', 'i': 'u', 'h': 'g'}

```
Decrypted text:
without discipline, the life of a person will become dull
and inacti*e. also, a disciplined person can control and handle
the situation of li*ing in a sophisticated wa* than those
who do not. moreo*er, if *ou ha*e a plan and
*ou want to implement it in *our life then *ou
need discipline. it ma*es things eas* for *ou to handle
and ultimatel* bring success to *our life. if tal* about
the t*pes of discipline, then the* are generall* of two
t*pes. first one is induced discipline and the second one
is self- discipline. induced discipline is something that others
taught
us or we learn b* seeing others. while self-discipline comes
from within and we learn it on our own self.
self-discipline re*uires a lot of moti*ation and support from others.
abo*e all, following *our dail* schedule without an* mista*e is
also part of being disciplined.
```

Reason for the mapping:

- Here I have mapped 'h' to 'g' and have got more valid words like **following**, **bring**, **something** and so on.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h', 'u': 't', 'i': 'u', 'h': 'g', 'c': 'y'}

```
Decrypted text:
without discipline, the life of a person will become dull
```

```
and inacti*e. also, a disciplined person can control and handle
the situation of li*ing in a sophisticated way than those
who do not. moreo*er, if you ha*e a plan and
you want to implement it in your life then you
need discipline. it ma*es things easy for you to handle
and ultimately bring success to your life. if tal* about
the types of discipline, then they are generally of two
types. first one is induced discipline and the second one
is self- discipline. induced discipline is something that others
taught
us or we learn by seeing others. while self-discipline comes
from within and we learn it on our own self.
self-discipline re*uires a lot of moti*ation and support from others.
abo*e all, following your daily schedule without any mista*e is
also part of being disciplined.
```

Reason for the mapping:

- After mapping 'c' to 'y' I have formed a valid word **way**, **easy**, **daily** and so on. So, the mapping is correct.

21 **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h', 'u': 't', 'i': 'u', 'h': 'g', 'c': 'y', 'j': 'v'}

```
Decrypted text:
without discipline, the life of a person will become dull
and inactive. also, a disciplined person can control and handle
the situation of living in a sophisticated way than those
who do not. moreover, if you have a plan and
you want to implement it in your life then you
need discipline. it ma*es things easy for you to handle
and ultimately bring success to your life. if tal* about
the types of discipline, then they are generally of two
types. first one is induced discipline and the second one
is self- discipline. induced discipline is something that others
taught
us or we learn by seeing others. while self-discipline comes
from within and we learn it on our own self.
self-discipline re*uires a lot of motivation and support from others.
above all, following your daily schedule without any mista*e is
also part of being disciplined.
```

Reason for the mapping:

- By mapping 'j' to 'v' I have formed a valid word **living**, **moreover**, **motivation** and so on. So, the mapping is correct.

1. **Current mapping:**

{'l': 'a', 'x': 'i', 'm': 'e', 'z': 'o', 'o': 'n', 's': 'f', 'g': 'l', 'w': 's', 'a': 'p', 't': 'r', 'k': 'd', 'q': 'c', 'e': 'w', 'r': 'b', 'f': 'm', 'p': 'h', 'u': 't', 'i': 'u', 'h': 'g', 'c': 'y', 'j': 'v', 'n': 'k', 'y': 'q'}

```
Decrypted text:
without discipline, the life of a person will become dull
and inactive. also, a disciplined person can control and handle
the situation of living in a sophisticated way than those
who do not. moreover, if you have a plan and
you want to implement it in your life then you
need discipline. it makes things easy for you to handle
and ultimately bring success to your life. if talk about
the types of discipline, then they are generally of two
types. first one is induced discipline and the second one
is self- discipline. induced discipline is something that others
taught
us or we learn by seeing others. while self-discipline comes
from within and we learn it on our own self.
self-discipline requires a lot of motivation and support from others.
above all, following your daily schedule without any mistake is
also part of being disciplined.
```

Reason for the mapping:

- By mapping 'n' to 'k' and 'y' to 'q' I have formed a valid decrypted text which gramartically and syntactically correct. So, the mapping is correct.

Deciphered Text:

without discipline, the life of a person will become dull and inactive. Also, a disciplined person can control and handle the situation of living in a sophisticated way than those who do not.moreover, if you have a plan and you want to implement it in your life then you need discipline. It makes things easy for you to handle and ultimately bring success to your life. If talk about the types of discipline, then they are generally of two types. First one is induced discipline and the second one is selfdiscipline. induced discipline is something that others taught us or we learn by seeing others. While self-discipline comes from within and we learn it on our own self. Self-discipline requires a lot of motivation and support from others. Above all, following your daily schedule without any mistake is also part of being disciplined.

Final mapping table

Encrypted	Decrypted	Encrypted	Decrypted	Encrypted	Decrypted	Encrypted	Decrypted
l	a	x	i	m	e	z	o
o	n	s	f	g	l	w	s
a	p	t	r	k	d	q	c
e	w	r	b	f	m	p	h
u	t	i	u	h	g	c	y

Lab Task 2(c)

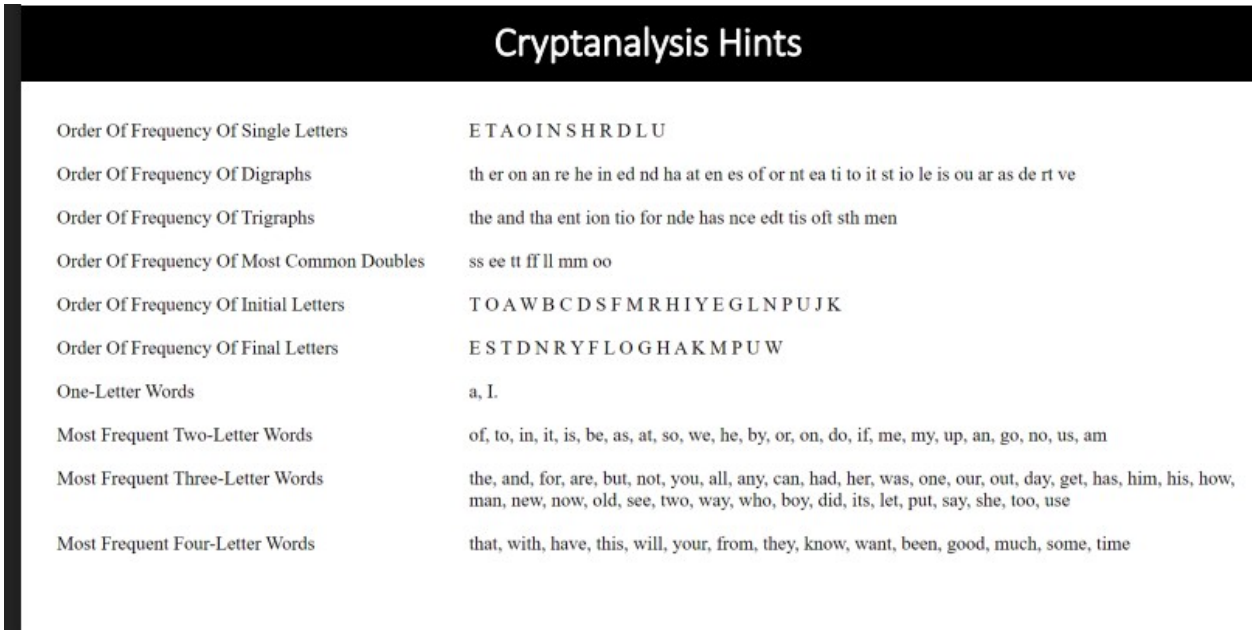
```
from collections import Counter
def print_text_with_spaces(text, words_per_line):
    words = text.split()
    for i in range(0, len(words), words_per_line):
        print(" ".join(words[i:i+words_per_line]))


def decrypt(text, mapping):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            if char in mapping:
                decrypted_text += mapping[char]
            else:
                decrypted_text += "*"
        else:
            decrypted_text += char  # Preserve punctuation
    return decrypted_text

def analyze_frequency(ciphertext):
    # Remove spaces and punctuation
    ciphertext = ''.join(char for char in ciphertext if
char.isalpha())
    # Count the frequency of each letter
    frequency = Counter(ciphertext)
    # Sort the letters by frequency
    sorted_frequency = sorted(frequency.items(), key=lambda x: x[1],
reverse=True)
    return sorted_frequency

def algo(text, mapping, words_per_line):
    while True:
        print("Current mapping:")
        print(mapping)
        reverse_mapping = {}
        found_duplicate = False
        for key, value in mapping.items():
          if value in reverse_mapping:
            print(f"{reverse_mapping[value]} maps to the same value as
{key}: {value}")
            found_duplicate = True
            break
          else:
            reverse_mapping[value] = key
        if found_duplicate:
```

```
            break
        frequency_list = analyze_frequency(text)
        decrypted_text = decrypt(text, mapping)
        print("Decrypted text:")
        print_text_with_spaces(decrypted_text, words_per_line)

        print("Frequency list:")
        for char, freq in frequency_list:
            print(f"{char}: {freq}")

        replacement = input("Enter replacement (or 'q' to quit): ")
        if replacement.lower() == 'q':
            break
        if len(replacement) != 2 or replacement[0] not in
'abcdefghijklmnopqrstuvwxyz' or replacement[1] not in
'abcdefghijklmnopqrstuvwxyz':
            print("Invalid input. Please enter a valid replacement.")
            continue
        if replacement[0] in mapping.values() or replacement[1] in
mapping.values():
            print("Error: One of the replacement characters is already
mapped to another character.")
            continue
        mapping[replacement[0]] = replacement[1]

        # Check if any character is mapped to the same replacement
        duplicate_mapping = [char for char, mapped_char in
mapping.items() if mapped_char == replacement[1] and char !=
replacement[0]]
        if duplicate_mapping:
            print(f"Error: Character '{duplicate_mapping[0]}' is
already mapped to '{replacement[1]}'.")
            del mapping[replacement[0]]  # Rollback the mapping
            continue

text = "AUHC MVKFC V BYZUGC V IZMC CJ GUMBZYAZD UKUVM. VC \
HZZGZB CJ GZ, V HCJJB PD CFZ VYJM KUCZ AZUBVMK \
CJ CFZ BYVWZ UMB OJY U IFVAZ, V TJNAB MJC \
ZMCZY. OJY CFZ IUD, VC IUH PUYYZB CJ GZ."

mapping = {}
words_per_line = 10
algo(text, mapping, words_per_line)
```

Frequency Table

Character	Frequency	Character	Frequency	Character	Frequency	Character	Frequency
Z	19	C	17	U	12	V	12
J	11	M	9	B	9	Y	9

Character	Frequency	Character	Frequency	Character	Frequency	Character	Frequency
A	5	F	5	G	5	H	4
K	4	I	4	D	3	P	2
O	2	W	1	T	1	N	1



I will the use the above hints attached to the lecture slide to solve the problem.

Current mapping:

{'Z': 'E'}

```
Decrypted text:
**** ***** * **E*** * *E** ** ****E**E* *****. **
*EE*E* ** *E, * ***** ** **E **** ***E *E*****
** **E ****E *** *** * ****E, * ***** ***
E**E*. *** **E ***, ** *** ****E* ** *E.
```

Reason for mapping:

- 'Z' is the most frequent character in the cipher text. It is mapped to 'E' in the decrypted text.

Current mapping: {'Z': 'E', 'V': 'I'}

```
Decrypted text:
**** *I*** I **E*** I *E** ** ****E**E* ***I*. I*
*EE*E* ** *E, I ***** ** **E I*** ***E *E**I**
** **E **I*E *** *** * **I*E, I ***** ***
E**E*. *** **E ***, I* *** ****E* ** *E.
```

Reason for mapping:

- 'V' is the single letter word in the cipher text. In English, 'I' and 'A' are two single word letter . The most frequent single letter word in English is 'I'. So, 'V' is mapped to 'I' in the decrypted text.

Current mapping: {'Z': 'E', 'V': 'I', 'U': 'A'}

```
Decrypted text:
*A** *I*** I **EA** I *E** ** *A**E**E* A*AI*. I*
*EE*E* ** *E, I ***** ** **E I*** *A*E *EA*I**
** **E **I*E A** *** A **I*E, I ***** ***
E**E*. *** **E *A*, I* *A* *A**E* ** *E.
```

Reason for mapping:

- 'U' is another single letter word. As I mentioned earlier, 'I' and 'A' are two single word letter in English. So, 'U' is mapped to 'A' in the decrypted text.

Current mapping: {'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T'}

```
Decrypted text:
*A*T *I**T I **EA*T I *E*T T* *A**E**E* A*AI*. IT
*EE*E* T* *E, I *T*** ** T*E I*** *ATE *EA*I**
T* T*E **I*E A** *** A **I*E, I ***** **T
E*TE*. *** T*E *A*, IT *A* *A**E* T* *E.
```

Reason for mapping:

- 'C' is the second most frequent character in the cipher text. It is mapped to 'T' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O'}

```
Decrypted text:
*A*T *I**T I **EA*T I *E*T TO *A**E**E* A*AI*. IT
*EE*E* TO *E, I *TO0* ** T*E I*O* *ATE *EA*I**
TO T*E **I*E A** *O* A **I*E, I *O*** *OT
E*TE*. *O* T*E *A*, IT *A* *A**E* TO *E.
```

Reason for mapping:

- If we look at the decrypted text, we can see that the word 'TO' is repeated many times. So, 'J' is mapped to 'O' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M'}

```
Decrypted text:
*A*T *I**T I **EAMT I *E*T TO MA**E**E* A*AI*. IT
*EEME* TO ME, I *TOO* ** T*E I*O* *ATE *EA*I**
TO T*E **I*E A** *O* A **I*E, I *O*** *OT
E*TE*. *O* T*E *A*, IT *A* *A**E* TO ME.
```

Reason for mapping:

- If I map 'G' to 'M', the word **ME** will be formed after **TO** in the last line and which is grammatically correct. So, 'G' is mapped to 'M' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H'}

```
Decrypted text:
*A*T *I*HT I **EAMT I *E*T TO MA**E**E* A*AI*. IT
*EEME* TO ME, I *TOO* ** THE I*O* *ATE *EA*I**
TO THE **I*E A** *O* A *HI*E, I *O*** *OT
E*TE*. *O* THE *A*, IT *A* *A**E* TO ME.
```

Reason for mapping:

- If I map 'F' to 'H', the word **THE** will be formed .

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G'}

```
Decrypted text:
*A*T NIGHT I **EAMT I *ENT TO MAN*E**E* AGAIN. IT
*EEME* TO ME, I *TOO* ** THE I*ON GATE *EA*ING
TO THE **I*E AN* *O* A *HI*E, I *O*** NOT
ENTE*. *O* THE *A*, IT *A* *A**E* TO ME.
```

Reason for mapping:

- I assume that the second the word in the decrypted text is **NIGHT** and the word **GATE** is formed after **THE** in the second line. So, 'M' is mapped to 'N' and 'K' is mapped to 'G' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S'}

```
Decrypted text:
LAST NIGHT I **EAMT I *ENT TO MAN*E*LE* AGAIN. IT
SEEME* TO ME, I STOO* ** THE I*ON GATE LEA*ING
```

```
TO THE **I*E AN* *O* A *HILE, I *O*L* NOT
ENTE*. *O* THE *A*, IT *AS *A**E* TO ME.
```

Reason for mapping:

- Before **NIGHT** in the decrypted text, the word **LAST** is formed. So, 'A' is mapped to 'L' and 'H' is mapped to 'S' in the decrypted text. Let's see if the decrypted text makes sense.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D'}

```
Decrypted text:
LAST NIGHT I D*EAMT I *ENT TO MANDE*LE* AGAIN. IT
SEEMED TO ME, I STOOD ** THE I*ON GATE LEADING
TO THE D*I*E AND *O* A *HILE, I *O*LD NOT
ENTE*. *O* THE *A*, IT *AS *A*'*ED TO ME.
```

Reason for mapping:

- If I map 'B' to 'D' the valid words **SEEMED**, **STOOD**, **LEADING** is formed. So, 'B' is mapped to 'D' in the decrypted text.

Current mapping: {'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R'}

```
Decrypted text:
LAST NIGHT I DREAMT I *ENT TO MANDERLE* AGAIN. IT
SEEMED TO ME, I STOOD ** THE IRON GATE LEADING
TO THE DRI*E AND *OR A *HILE, I *O*LD NOT
ENTER. *OR THE *A*, IT *AS *ARRED TO ME.
```

Reason for mapping:

- More correct word like
  - **DREAMT** is formed after **NIGHT**
  - **IRON** is formed after **THE**
  - __DRI*E__ is formed after **TO**
  - **ENTER** is formed after **NOT**
  - **ARRIVED** is formed after **IT**

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R', 'O': 'F'}

```
Decrypted text:
LAST NIGHT I DREAMT I *ENT TO MANDERLE* AGAIN. IT
SEEMED TO ME, I STOOD ** THE IRON GATE LEADING
```

```
TO THE DRI*E AND FOR A *HILE, I *O*LD NOT
ENTER. FOR THE *A*, IT *AS *ARRED TO ME.
```

Reason for mapping:

- I assume that the *OR may be FOR. So, 'O' is mapped to 'F' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R', 'O': 'F', 'T': 'C', 'N': 'U'}

```
Decrypted text:
LAST NIGHT I DREAMT I *ENT TO MANDERLE* AGAIN. IT
SEEMED TO ME, I STOOD ** THE IRON GATE LEADING
TO THE DRI*E AND FOR A *HILE, I COULD NOT
ENTER. FOR THE *A*, IT *AS *ARRED TO ME.
```

Reason for mapping:

- The word phrase "*O*LD NOT" may be **COULD NOT**. So, 'N' is mapped to 'U' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R', 'O': 'F', 'T': 'C', 'N': 'U', 'I': 'W'}

```
Decrypted text:
LAST NIGHT I DREAMT I WENT TO MANDERLE* AGAIN. IT
SEEMED TO ME, I STOOD ** THE IRON GATE LEADING
TO THE DRI*E AND FOR A WHILE, I COULD NOT
ENTER. FOR THE WA*, IT WAS *ARRED TO ME.
```

Reason for mapping:

- The words such as:
  - WHILE
  - WENT are formed if I map 'I' to 'W' in the decrypted text.

Current mapping: {'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R', 'O': 'F', 'T': 'C', 'N': 'U', 'I': 'W', 'D': 'Y'}

```
Decrypted text:
LAST NIGHT I DREAMT I WENT TO MANDERLEY AGAIN. IT
SEEMED TO ME, I STOOD *Y THE IRON GATE LEADING
TO THE DRI*E AND FOR A WHILE, I COULD NOT
ENTER. FOR THE WAY, IT WAS *ARRED TO ME.
```

Reason for mapping:

- The word **WAY** is formed after **FOR** in the decrypted text. So, 'D' is mapped to 'Y' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R', 'O': 'F', 'T': 'C', 'N': 'U', 'I': 'W', 'D': 'Y', 'P': 'B'}

```
Decrypted text:
LAST NIGHT I DREAMT I WENT TO MANDERLEY AGAIN. IT
SEEMED TO ME, I STOOD BY THE IRON GATE LEADING
TO THE DRI*E AND FOR A WHILE, I COULD NOT
ENTER. FOR THE WAY, IT WAS BARRED TO ME.
```

Reason for mapping:

- The word **BARRED** is formed after **IT** in the decrypted text. So, 'P' is mapped to 'B' in the decrypted text.

Current mapping:

{'Z': 'E', 'V': 'I', 'U': 'A', 'C': 'T', 'J': 'O', 'G': 'M', 'F': 'H', 'M': 'N', 'K': 'G', 'A': 'L', 'H': 'S', 'B': 'D', 'Y': 'R', 'O': 'F', 'T': 'C', 'N': 'U', 'I': 'W', 'D': 'Y', 'P': 'B', 'W': 'V'}

```
Decrypted text:
LAST NIGHT I DREAMT I WENT TO MANDERLEY AGAIN. IT
SEEMED TO ME, I STOOD BY THE IRON GATE LEADING
TO THE DRIVE AND FOR A WHILE, I COULD NOT
ENTER. FOR THE WAY, IT WAS BARRED TO ME.
```

Reason for mapping:

- The word **DRIVE** is formed after **TO** in the decrypted text. So, 'W' is mapped to 'V' in the decrypted text. And thus our decrypted text is formed.

Decrypted text:

```
LAST NIGHT I DREAMT I WENT TO MANDERLEY AGAIN. IT
SEEMED TO ME, I STOOD BY THE IRON GATE LEADING
TO THE DRIVE AND FOR A WHILE, I COULD NOT
ENTER. FOR THE WAY, IT WAS BARRED TO ME.
```

Final mapping:

Encrypted	Decrypted
Z	E
V	I
U	A
C	T
J	O

Encrypted	Decrypted
G	M
F	H
M	N
K	G
A	L
H	S
B	D
Y	R
O	F
T	C
N	U
I	W
D	Y
P	B
W	V

Lab Task 2(d)

```
from collections import Counter
def print_text_with_spaces(text, words_per_line):
    words = text.split()
    for i in range(0, len(words), words_per_line):
        print(" ".join(words[i:i+words_per_line]))


def decrypt(text, mapping):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            if char in mapping:
                decrypted_text += mapping[char]
            else:
                decrypted_text += "*"
        else:
            decrypted_text += char  # Preserve punctuation
    return decrypted_text

def analyze_frequency(ciphertext):
    # Remove spaces and punctuation
    ciphertext = ''.join(char for char in ciphertext if
char.isalpha())
    # Count the frequency of each letter
    frequency = Counter(ciphertext)
    # Sort the letters by frequency
    sorted_frequency = sorted(frequency.items(), key=lambda x: x[1],
reverse=True)
    return sorted_frequency

def algo(text, mapping, words_per_line):
    while True:
        print("Current mapping:")
        print(mapping)
        reverse_mapping = {}
        found_duplicate = False
        for key, value in mapping.items():
          if value in reverse_mapping:
            print(f"{reverse_mapping[value]} maps to the same value as
{key}: {value}")
            found_duplicate = True
            break
          else:
            reverse_mapping[value] = key
        if found_duplicate:
```

```
            break
        frequency_list = analyze_frequency(text)
        decrypted_text = decrypt(text, mapping)
        print("Decrypted text:")
        print_text_with_spaces(decrypted_text, words_per_line)

        print("Frequency list:")
        for char, freq in frequency_list:
            print(f"{char}: {freq}")

        replacement = input("Enter replacement (or 'q' to quit): ")
        if replacement.lower() == 'q':
            break
        if len(replacement) != 2 or replacement[0] not in
'abcdefghijklmnopqrstuvwxyz' or replacement[1] not in
'abcdefghijklmnopqrstuvwxyz':
            print("Invalid input. Please enter a valid replacement.")
            continue
        if replacement[0] in mapping.values() or replacement[1] in
mapping.values():
            print("Error: One of the replacement characters is already
mapped to another character.")
            continue
        mapping[replacement[0]] = replacement[1]

        # Check if any character is mapped to the same replacement
        duplicate_mapping = [char for char, mapped_char in
mapping.items() if mapped_char == replacement[1] and char !=
replacement[0]]
        if duplicate_mapping:
            print(f"Error: Character '{duplicate_mapping[0]}' is
already mapped to '{replacement[1]}'.")
            del mapping[replacement[0]]  # Rollback the mapping
            continue

text = "JGRMQOYGHMVBJ WRWQFPW HGF FDQGFPFZR KBEEBJIZQ QO CIBZK.
LFAFGQVFZFWW, EOG WOPF \
GFHWOL PHLR LOLFDMFGQW BLWBWQ OL KFWBYLBLY LFS FLJGRMQBOL WJVFPFW QVHQ
\
WFFP QO QVFP QO CF POGF WFJIGF QVHL HLR OQVFG \
WJVFPF OL FHGQV. QVF ILEOGQILHQF QGIQV VOSFAFG BW QVHQ WIJV \
WJVFPFW HGF IWIHZZR QGBABHZ QO CGFHX"

mapping = {}
words_per_line = 10
algo(text, mapping, words_per_line)
```

Frequency Analysis

Character	Frequency	Character	Frequency	Character	Frequency	Character	Frequency
F	37	Q	26	W	21	G	19
L	17	O	16	V	15	H	14
B	12	P	10	J	9	I	9
R	7	Z	7	M	4	E	4
Y	3	K	3	C	3	A	3
D	2	S	2	X	1		

Cryptanalysis Hints

Order Of Frequency Of Single Letters	E T A O I N S H R D L U
Order Of Frequency Of Digraphs	th er on an re he in ed nd ha at en es of or nt ea ti to it st io le is ou ar as de rt ve
Order Of Frequency Of Trigraphs	the and tha ent ion tio for nde has nce edt tis oft sth men
Order Of Frequency Of Most Common Doubles	ss ee tt ff ll mm oo
Order Of Frequency Of Initial Letters	T O A W B C D S F M R H I Y E G L N P U J K
Order Of Frequency Of Final Letters	E S T D N R Y F L O G H A K M P U W
One-Letter Words	a, I.
Most Frequent Two-Letter Words	of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am
Most Frequent Three-Letter Words	the, and, for, are, but, not, you, all, any, can, had, her, was, one, our, out, day, get, has, him, his, how, man, new, now, old, see, two, way, who, boy, did, its, let, put, say, she, too, use
Most Frequent Four-Letter Words	that, with, have, this, will, your, from, they, know, want, been, good, much, some, time

Frequency Analysis of the given text shows that the most frequent character is 'F' and the least frequent character is 'X'. The frequency of the characters is shown in the table above and the bar graph shows the frequency of the characters. I am going to use this frequency analysis to solve the problem.

Current mapping: {'F': 'E'}

```
Decrypted text:
************* ****E** **E E***E*E** ********* ** *****. *E*E***E*E**,
*** ***E
*E**** **** ***E**E*** ****** ** *E******* *E* E********* ***E*E* ****
*EE* ** **E* ** *E ***E *E***E **** *** ***E*
***E*E ** E****. **E **********E ***** ***E*E* ** **** ****
***E*E* **E ******* ******* ** **E**
```

Reason for the mapping:

- The most frequent character in the encrypted text is 'F' and the most frequent character in the English language is 'E'. So, I am going to map 'F' to 'E'.

Current mapping:

{'F': 'E', 'Q': 'T'}

```
Decrypted text:
****T******** ***TE** **E E*T*E*E** ********T T* *****. *E*E*T*E*E**,
*** ***E
*E**** **** ***E**E*T* *****T ** *E******* *E* E*****T*** ***E*E* T**T
*EE* T* T*E* T* *E ***E *E***E T*** *** *T*E*
***E*E ** E**T*. T*E *****T***TE T**T* ***E*E* ** T**T ****
***E*E* **E ******* T****** T* **E**
```

Reason for the mapping:

- The second most frequent character in the encrypted text is 'Q' and the second most frequent character in the English language is 'T'. So, I am going to map 'Q' to 'T'.

Current mapping: {'F': 'E', 'Q': 'T', 'O': 'O'}

```
Decrypted text:
****TO******* ***TE** **E E*T*E*E** ********T TO *****. *E*E*T*E*E**,
*O* *O*E
*E**O* **** *O*E**E*T* *****T O* *E******* *E* E*****T*O* ***E*E* T**T
*EE* TO T*E* TO *E *O*E *E***E T*** *** OT*E*
***E*E O* E**T*. T*E ***O*T***TE T**T* *O*E*E* ** T**T ****
***E*E* **E ******* T****** TO **E**
```

Reason for the mapping:

- There are many two letter words starting with 'T' and the most common two letter word starting with 'T' is 'TO'. So, I am going to map 'O' to 'O'.

Current mapping: {'F': 'E', 'Q': 'T', 'O': 'O', 'W': 'A'}

```
Decrypted text:
****TO******* A*ATE*A **E E*T*E*E** ********T TO *****. *E*E*T*E*EAA,
*O* AO*E
*E*AO* **** *O*E**E*TA **A*AT O* *EA****** *E* E*****T*O* A**E*EA T**T
AEE* TO T*E* TO *E *O*E AE***E T*** *** OT*E*
A**E*E O* E**T*. T*E ***O*T***TE T**T* *O*E*E* *A T**T A***
A**E*EA **E *A***** T****** TO **E**
```

Reason for the mapping:

- This mapping is not seemed to be correct as the decrypted text is not making any sense there is not two letter word that ends with 'A'. So, I am going to change the mapping of 'W' to 'A'.

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H'}

```
Decrypted text:
****TO**A*H** ***TE** A*E E*T*E*E** ********T TO *****. *E*E*THE*E**,
*O* *O*E
*EA*O* *A** *O*E**E*T* *****T O* *E******* *E* E*****T*O* **HE*E* THAT
*EE* TO THE* TO *E *O*E *E***E THA* A** OTHE*
**HE*E O* EA*TH. THE ***O*T**ATE T**TH HO*E*E* ** THAT ***H
**HE*E* A*E ***A*** T****A* TO **EA*
```

Reason for the mapping:

- I assume that the word **T--T** is **THAT** . So, I am going to map 'V' to 'H' and 'H' to 'A'. Let's see if the decrypted text makes sense.

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R'}

```
Decrypted text:
*R**TO*RA*H** ***TE** ARE E*TRE*E** ********T TO *****. *E*ERTHE*E**,
*OR *O*E
REA*O* *A** *O*E**ERT* *****T O* *E******* *E* E**R**T*O* **HE*E* THAT
*EE* TO THE* TO *E *ORE *E**RE THA* A** OTHER
**HE*E O* EARTH. THE ***ORT**ATE TR*TH HO*E*ER ** THAT ***H
**HE*E* ARE ***A*** TR***A* TO *REA*
```

Reason for the mapping:

- The decrypted text makes sense. So, I am going to map 'G' to 'R'. Because by mapping so I get the word **OTHER**.

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V'}

```
Decrypted text:
*R**TO*RA*H** ***TE** ARE E*TRE*E** ********T TO *****. *EVERTHE*E**,
*OR *O*E
REA*O* *A** *O*E**ERT* *****T O* *E******* *EW E**R**T*O* **HE*E* THAT
*EE* TO THE* TO *E *ORE *E**RE THA* A** OTHER
**HE*E O* EARTH. THE ***ORT**ATE TR*TH HOWEVER ** THAT ***H
**HE*E* ARE ***A*** TR*V*A* TO *REA*
```

Reason for the mapping:

- I assume that the word **HO-E-ER** is **HOWEVER**. So, I am going to map 'S' to 'W' and 'A' to 'V'. Let's see if the decrypted text makes sense.

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M'}

```
Decrypted text:
*R**TO*RA*H** ***TEM* ARE E*TREME** ********T TO *****. *EVERTHE*E**,
*OR *OME
REA*O* MA** *O*E**ERT* *****T O* *E******* *EW E**R**T*O* **HEME* THAT
*EEM TO THEM TO *E MORE *E**RE THA* A** OTHER
**HEME O* EARTH. THE ***ORT**ATE TR*TH HOWEVER ** THAT ***H
**HEME* ARE ***A*** TR*V*A* TO *REA*
```

Reason for the mapping:

- I can get the words
  - **THEM** by mapping 'P' to 'M'
  - **MORE**

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y'}

```
Decrypted text:
*RY*TO*RA*H** *Y*TEM* ARE EXTREMELY *******LT TO ***L*. *EVERTHELE**,
*OR *OME
REA*O* MA*Y *O*EX*ERT* *****T O* *E******* *EW E**RY*T*O* **HEME* THAT
*EEM TO THEM TO *E MORE *E**RE THA* A*Y OTHER
**HEME O* EARTH. THE ***ORT**ATE TR*TH HOWEVER ** THAT ***H
**HEME* ARE ***ALLY TR*V*AL TO *REA*
```

Reason for the mapping:

- I can get the words
  - **EXTREMELY**

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S'}

```
Decrypted text:
*RY*TO*RA*H** SYSTEMS ARE EXTREMELY *******LT TO ***L*. *EVERTHELESS,
*OR SOME
REASO* MA*Y *O*EX*ERTS **S*ST O* *ES****** *EW E**RY*T*O* S*HEMES THAT
SEEM TO THEM TO *E MORE SE**RE THA* A*Y OTHER
S*HEME O* EARTH. THE ***ORT**ATE TR*TH HOWEVER *S THAT S**H
S*HEMES ARE *S*ALLY TR*V*AL TO *REA*
```

Reason for the mapping:

- I got the following valid words:
  - **SYSTEMS**
  - **NEVERTHELESS**
  - **EXPERTS**

- **SEEM** by mapping 'W' to 'S'

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F'}

```
Decrypted text:
*RY*TO*RA*H** SYSTEMS ARE EXTREMELY **FF***LT TO ***L*. *EVERTHELESS,
FOR SOME
REASO* MA*Y *O*EX*ERTS **S*ST O* *ES****** *EW E**RY*T*O* S*HEMES THAT
SEEM TO THEM TO *E MORE SE**RE THA* A*Y OTHER
S*HEME O* EARTH. THE **FORT**ATE TR*TH HOWEVER *S THAT S**H
S*HEMES ARE *S*ALLY TR*V*AL TO *REA*
```

Reason for the mapping:

- I got the following valid words:
  - **FOR** by mapping 'E' to 'F'

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N'}

```
Decrypted text:
*RY*TO*RA*H** SYSTEMS ARE EXTREMELY **FF***LT TO ***L*. NEVERTHELESS,
FOR SOME
REASON MANY NONEX*ERTS *NS*ST ON *ES**N*N* NEW EN*RY*T*ON S*HEMES THAT
SEEM TO THEM TO *E MORE SE**RE THAN ANY OTHER
S*HEME ON EARTH. THE *NFORT*NATE TR*TH HOWEVER *S THAT S**H
S*HEMES ARE *S*ALLY TR*V*AL TO *REA*
```

Reason for the mapping:

- I assume that there may be a word **NONEXPERTS** and I am going to map 'L' to 'N'

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C'}

```
Decrypted text:
CRY*TO*RA*H*C SYSTEMS ARE EXTREMELY **FF*C*LT TO ***L*. NEVERTHELESS,
FOR SOME
REASON MANY NONEX*ERTS *NS*ST ON *ES**N*N* NEW ENCRY*T*ON SCHEMES THAT
SEEM TO THEM TO *E MORE SEC*RE THAN ANY OTHER
SCHEME ON EARTH. THE *NFORT*NATE TR*TH HOWEVER *S THAT S*CH
SCHEMES ARE *S*ALLY TR*V*AL TO *REA*
```

Reason for the mapping:

- I assume that there may be a word **SCHEME** and I am going to map 'J' to 'C'

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C', 'M': 'P'}

```
Decrypted text:
CRYPTO*RAPH*C SYSTEMS ARE EXTREMELY **FF*C*LT TO ***L*. NEVERTHELESS,
FOR SOME
REASON MANY NONEXPERTS *NS*ST ON *ES**N*N* NEW ENCRYPT*ON SCHEMES THAT
SEEM TO THEM TO *E MORE SEC*RE THAN ANY OTHER
SCHEME ON EARTH. THE *NFORT*NATE TR*TH HOWEVER *S THAT S*CH
SCHEMES ARE *S*ALLY TR*V*AL TO *REA*
```

Reason for the mapping:

- I assume that there may be a word **NONEXPERTS** and I am going to map 'M' to 'P'

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C', 'M': 'P', 'B': 'I'}

```
Decrypted text:
CRYPTO*RAPHIC SYSTEMS ARE EXTREMELY *IFFIC*LT TO **IL*. NEVERTHELESS,
FOR SOME
REASON MANY NONEXPERTS INSIST ON *ESI*NIN* NEW ENCRYPTION SCHEMES THAT
SEEM TO THEM TO *E MORE SEC*RE THAN ANY OTHER
SCHEME ON EARTH. THE *NFORT*NATE TR*TH HOWEVER IS THAT S*CH
SCHEMES ARE *S*ALLY TRIVIAL TO *REA*
```

Reason for the mapping:

- I assume that there may be a word **CRYPTOGRAPHIC** and I am going to map 'B' to 'I'
- Again I get another valid word **ENCRIPTION** by mapping 'B' to 'I'

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C', 'M': 'P', 'B': 'I', 'I': 'U'}

```
Decrypted text:
CRYPTO*RAPHIC SYSTEMS ARE EXTREMELY *IFFICULT TO *UIL*. NEVERTHELESS,
FOR SOME
REASON MANY NONEXPERTS INSIST ON *ESI*NIN* NEW ENCRYPTION SCHEMES THAT
SEEM TO THEM TO *E MORE SECURE THAN ANY OTHER
SCHEME ON EARTH. THE UNFORTUNATE TRUTH HOWEVER IS THAT SUCH
SCHEMES ARE USUALLY TRIVIAL TO *REA*
```

Reason for the mapping:

- By mapping 'I' to 'U' I get the word **USUALLY**
- **UNFORTUNATE**
- **TRUTH**
- **SECURE**
- **SUCH**

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C', 'M': 'P', 'B': 'I', 'I': 'U', 'C': 'B', 'K': 'D'}

```
Decrypted text:
CRYPTO*RAPHIC SYSTEMS ARE EXTREMELY DIFFICULT TO BUILD. NEVERTHELESS,
FOR SOME
REASON MANY NONEXPERTS INSIST ON DESI*NIN* NEW ENCRYPTION SCHEMES THAT
SEEM TO THEM TO BE MORE SECURE THAN ANY OTHER
SCHEME ON EARTH. THE UNFORTUNATE TRUTH HOWEVER IS THAT SUCH
SCHEMES ARE USUALLY TRIVIAL TO BREA*
```

Reason for the mapping:

- By mapping 'C' to 'B' I get the word **BUILD**. I am going to map 'K' to 'D' and see if the decrypted text makes sense.

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C', 'M': 'P', 'B': 'I', 'I': 'U', 'C': 'B', 'K': 'D', 'Y': 'G'}

```
Decrypted text:
CRYPTOGRAPHIC SYSTEMS ARE EXTREMELY DIFFICULT TO BUILD. NEVERTHELESS,
FOR SOME
REASON MANY NONEXPERTS INSIST ON DESIGNING NEW ENCRYPTION SCHEMES THAT
SEEM TO THEM TO BE MORE SECURE THAN ANY OTHER
SCHEME ON EARTH. THE UNFORTUNATE TRUTH HOWEVER IS THAT SUCH
SCHEMES ARE USUALLY TRIVIAL TO BREA*
```

Current mapping:

{'F': 'E', 'Q': 'T', 'O': 'O', 'H': 'A', 'V': 'H', 'G': 'R', 'S': 'W', 'A': 'V', 'P': 'M', 'D': 'X', 'Z': 'L', 'R': 'Y', 'W': 'S', 'E': 'F', 'L': 'N', 'J': 'C', 'M': 'P', 'B': 'I', 'I': 'U', 'C': 'B', 'K': 'D', 'Y': 'G', 'X': 'K'}

```
Decrypted text:
CRYPTOGRAPHIC SYSTEMS ARE EXTREMELY DIFFICULT TO BUILD. NEVERTHELESS,
FOR SOME
REASON MANY NONEXPERTS INSIST ON DESIGNING NEW ENCRYPTION SCHEMES THAT
SEEM TO THEM TO BE MORE SECURE THAN ANY OTHER
SCHEME ON EARTH. THE UNFORTUNATE TRUTH HOWEVER IS THAT SUCH
SCHEMES ARE USUALLY TRIVIAL TO BREAK
```

Reason for the mapping:

- I assume that the word __BREA*__ is **BREAK** and I am going to map 'X' to 'K'. And thus our decrypted text is complete.

Final decrypted text:

CRYPTOGRAPHIC SYSTEMS ARE EXTREMELY DIFFICULT TO BUILD. NEVERTHELESS, FOR SOME REASON MANY NONEXPERTS INSIST ON DESIGNING NEW ENCRYPTION SCHEMES THAT SEEM TO THEM TO BE MORE SECURE THAN ANY OTHER SCHEME ON EARTH. THE UNFORTUNATE TRUTH HOWEVER IS THAT SUCH SCHEMES ARE USUALLY TRIVIAL TO BREAK

Final Mapping

Encrypted	Decrypted
F	E
Q	T
O	O
H	A
V	H
G	R
S	W
A	V
P	M
D	X
Z	L
R	Y
W	S
E	F
L	N
J	C
M	P
B	I
I	U
C	B
K	D
Y	G
X	K

Lab Task 3

```
import string

def vigenere_encrypt(plaintext, key):

    ciphertext = ""
    index = 0
    keyLength = len(key)

    for char in plaintext.upper():
        if char not in string.ascii_uppercase:
            ciphertext += char
            continue

        k_i = ord(key[index % keyLength]) - ord('A')
        p_i=ord(char) - ord('A')
        encrypted_char = chr((p_i + k_i) % 26 + ord('A'))
        ciphertext += encrypted_char

        index += 1

    return ciphertext

def vigenere_decrypt(ciphertext, key):

    plaintext = ""
    index = 0
    keyLength = len(key)

    for char in ciphertext.upper():
        if char not in string.ascii_uppercase:
            plaintext += char
            continue

        p_i = ord(key[index % keyLength]) - ord('A')
        k_i=ord(char) - ord('A')
        decrypted_char = chr((k_i - p_i) % 26 + ord('A'))
        plaintext += decrypted_char

        index += 1

    return plaintext

if __name__ == "__main__":
    while True:
        print("\nVigenere Cipher")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
```

```
        choice = input("\nEnter your choice: ")

        if choice == '1':
            plaintext = input("\nEnter plaintext: ")
            key = input("\nEnter key: ")
            ciphertext = vigenere_encrypt(plaintext, key)
            print("\nEncrypted ciphertext:", ciphertext)
        elif choice == '2':
            ciphertext = input("\nEnter ciphertext: ")
            key = input("\nEnter key: ")
            plaintext = vigenere_decrypt(ciphertext, key)
            print("\nDecrypted plaintext:", plaintext)
        elif choice == '3':
            print("\nExiting...")
            break
        else:
            print("\nInvalid choice. Please try again.")


Vigenere Cipher
1. Encrypt
2. Decrypt
3. Exit

Enter your choice: 2

Enter ciphertext: UMW YKFED SWTT LLWIJR EGYJMW BPLVGXMVVASF NG VQETMYJ
LGUYJCLR CFH XIJXPKMUM

Enter key: SUSTCSE

Decrypted plaintext: CSE FINAL YEAR THEORY COURSE INTRODUCTION TO
COMPUER SECURITY AND FORENSICS

Vigenere Cipher
1. Encrypt
2. Decrypt
3. Exit

Enter your choice: 3

Exiting...

import string

def vigenere_encrypt(plaintext, key):
    ciphertext = ""
    index = 0
    keyLength = len(key)
    table = [["Plaintext", "Keyword", "Encrypted"]]
```

```
    for char in plaintext.upper():
        if char not in string.ascii_uppercase:
            table.append([char, "-", char])
            ciphertext += char
            continue

        k_i = ord(key[index % keyLength]) - ord('A')
        p_i = ord(char) - ord('A')
        encrypted_char = chr((p_i + k_i) % 26 + ord('A'))
        table.append([char, key[index % keyLength], encrypted_char])
        ciphertext += encrypted_char

        index += 1

    print_table(table)
    return ciphertext

def vigenere_decrypt(ciphertext, key):
    plaintext = ""
    index = 0
    keyLength = len(key)
    table = [["Ciphertext", "Keyword", "Decrypted"]]

    for char in ciphertext.upper():
        if char not in string.ascii_uppercase:
            table.append([char, "-", char])
            plaintext += char
            continue

        p_i = ord(key[index % keyLength]) - ord('A')
        k_i = ord(char) - ord('A')
        decrypted_char = chr((k_i - p_i) % 26 + ord('A'))
        table.append([char, key[index % keyLength], decrypted_char])
        plaintext += decrypted_char

        index += 1

    print_table(table)
    return plaintext

def print_table(table):
    col_width = [max(len(str(x)) for x in col) for col in zip(*table)]
    for row in table:
        print("".join(str(val).ljust(width + 2) for val, width in
zip(row, col_width)))

if __name__ == "__main__":
    while True:
        print("\nVigenere Cipher")
```

```
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")

        choice = input("\nEnter your choice: ")

        if choice == '1':
            plaintext = input("\nEnter plaintext: ")
            key = input("\nEnter key: ")
            ciphertext = vigenere_encrypt(plaintext, key)
            print("\nEncrypted ciphertext:", ciphertext)
        elif choice == '2':
            ciphertext = input("\nEnter ciphertext: ")
            key = input("\nEnter key: ")
            plaintext = vigenere_decrypt(ciphertext, key)
            print("\nDecrypted plaintext:", plaintext)
        elif choice == '3':
            print("\nExiting...")
            break
        else:
            print("\nInvalid choice. Please try again.")


Vigenere Cipher
1. Encrypt
2. Decrypt
3. Exit
```

Lab Task 4

Decryption

```
import numpy as np
import math

# Function to remove spaces from a sentence
def remove_spaces(sentence):
    return sentence.replace(" ", "")

# Function to calculate the modular inverse of a number
def mod_inv(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

# Function to calculate the modular inverse of a matrix
def matrix_mod_inv(matrix, modulus):
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = mod_inv(det, modulus)
    if det_inv is None:
        raise ValueError("Modular inverse does not exist.")
    matrix_modulus_inv = (det_inv * np.round(det *
np.linalg.inv(matrix)).astype(int) % modulus)
    return matrix_modulus_inv

# Function to generate a key matrix from the given key string
def generate_key_matrix(key):
    keylen = len(key)
    n = math.ceil(math.sqrt(keylen))
    key_matrix = np.array([], dtype=int)
    for i in range(n, keylen + n, n):
        temp = np.array([])
        string = key[(i - n):i]
        for x in string:
            temp = np.append(temp, (ord(x) - ord('A')))
        if len(key_matrix):
            key_matrix = np.vstack([key_matrix, temp])
        else:
            key_matrix = temp
    return key_matrix

# Function to generate a text matrix from the given text
def generate_text_matrix(text):
    tlen = len(text)
    text_matrix = np.array([], dtype=int)
```

```
    for i in range(tlen):
        text_matrix = np.append(text_matrix, (ord(text[i]) -
ord('A')))
    text_matrix = np.resize(text_matrix, (tlen, 1))
    return text_matrix

# Function to encrypt text using the Hill Cipher
def hill_encryption(text, key):
    text_matrix = generate_text_matrix(text)
    key_matrix = generate_key_matrix(key)
    enc = np.dot(key_matrix, text_matrix) % 26
    non_space_chars = sum(1 for char in text if char != ' ')
    enc = np.resize(enc, (1, non_space_chars))
    ec_text = ''
    space_positions = []
    for i, char in enumerate(text):
        if char == ' ':
            space_positions.append(i)
    for i in enc[0]:
        ec_text += chr(ord('A') + int(i))
    return ec_text

# Function to decrypt text using the Hill Cipher
def hill_decryption(text, key):
    keylen = len(key)
    tlen = len(text)
    n = math.ceil(math.sqrt(keylen))
    text_matrix = generate_text_matrix(text)
    key_matrix = generate_key_matrix(key)
    key_matrix_inv = matrix_mod_inv(key_matrix, 26)
    dec = np.dot(key_matrix_inv, text_matrix) % 26
    non_space_chars = sum(1 for char in text if char != ' ')
    dec = np.resize(dec, (1, non_space_chars))
    dec_text = ''
    for i in dec[0]:
        dec_text += chr(ord('A') + int(i))
    return dec_text

# User input
text = input("Enter text: ").upper()
key = input("Enter key: ").upper()
option = input("Choose operation (encryption/decryption): ").lower()

if option == "encryption":
    ciphertext = hill_encryption(text, key)
    print("Encrypted ciphertext:", ciphertext)
elif option == "decryption":
    decrypted_text = hill_decryption(text, key)
    print("Decrypted plaintext:", decrypted_text)
else:
```

```
    print("Invalid option. Please choose either encryption or
decryption.")


Enter text: POH
Enter key: GYBNQKURP
Choose operation (encryption/decryption): decryption
Decrypted plaintext: ACT
```

Encryption

```
import numpy as np
import math

# Function to remove spaces from a sentence
def remove_spaces(sentence):
    return sentence.replace(" ", "")

# Function to calculate the modular inverse of a number
def mod_inv(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

# Function to calculate the modular inverse of a matrix
def matrix_mod_inv(matrix, modulus):
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = mod_inv(det, modulus)
    if det_inv is None:
        raise ValueError("Modular inverse does not exist.")
    matrix_modulus_inv = (det_inv * np.round(det *
np.linalg.inv(matrix)).astype(int) % modulus)
    return matrix_modulus_inv

# Function to generate a key matrix from the given key string
def generate_key_matrix(key):
    keylen = len(key)
    n = math.ceil(math.sqrt(keylen))
    key_matrix = np.array([], dtype=int)
    for i in range(n, keylen + n, n):
        temp = np.array([])
        string = key[(i - n):i]
        for x in string:
            temp = np.append(temp, (ord(x) - ord('A')))
        if len(key_matrix):
            key_matrix = np.vstack([key_matrix, temp])
        else:
            key_matrix = temp
    return key_matrix
```

```
# Function to generate a text matrix from the given text
def generate_text_matrix(text):
    tlen = len(text)
    text_matrix = np.array([], dtype=int)
    for i in range(tlen):
        text_matrix = np.append(text_matrix, (ord(text[i]) -
ord('A')))
    text_matrix = np.resize(text_matrix, (tlen, 1))
    return text_matrix

# Function to encrypt text using the Hill Cipher
def hill_encryption(text, key):
    text_matrix = generate_text_matrix(text)
    key_matrix = generate_key_matrix(key)
    enc = np.dot(key_matrix, text_matrix) % 26
    non_space_chars = sum(1 for char in text if char != ' ')
    enc = np.resize(enc, (1, non_space_chars))
    ec_text = ''
    space_positions = []
    for i, char in enumerate(text):
        if char == ' ':
            space_positions.append(i)
    for i in enc[0]:
        ec_text += chr(ord('A') + int(i))
    return ec_text

# Function to decrypt text using the Hill Cipher
def hill_decryption(text, key):
    keylen = len(key)
    tlen = len(text)
    n = math.ceil(math.sqrt(keylen))
    text_matrix = generate_text_matrix(text)
    key_matrix = generate_key_matrix(key)
    key_matrix_inv = matrix_mod_inv(key_matrix, 26)
    dec = np.dot(key_matrix_inv, text_matrix) % 26
    non_space_chars = sum(1 for char in text if char != ' ')
    dec = np.resize(dec, (1, non_space_chars))
    dec_text = ''
    for i in dec[0]:
        dec_text += chr(ord('A') + int(i))
    return dec_text

# User input
text = input("Enter text: ").upper()
key = input("Enter key: ").upper()
option = input("Choose operation (encryption/decryption): ").lower()

if option == "encryption":
    ciphertext = hill_encryption(text, key)
```

```
    print("Encrypted ciphertext:", ciphertext)
elif option == "decryption":
    decrypted_text = hill_decryption(text, key)
    print("Decrypted plaintext:", decrypted_text)
else:
    print("Invalid option. Please choose either encryption or
decryption.")
```

```
Enter text: ACT
Enter key: GYBNQKURP
Choose operation (encryption/decryption): encryption
Encrypted ciphertext: POH
```

This Testcase is taken from slide