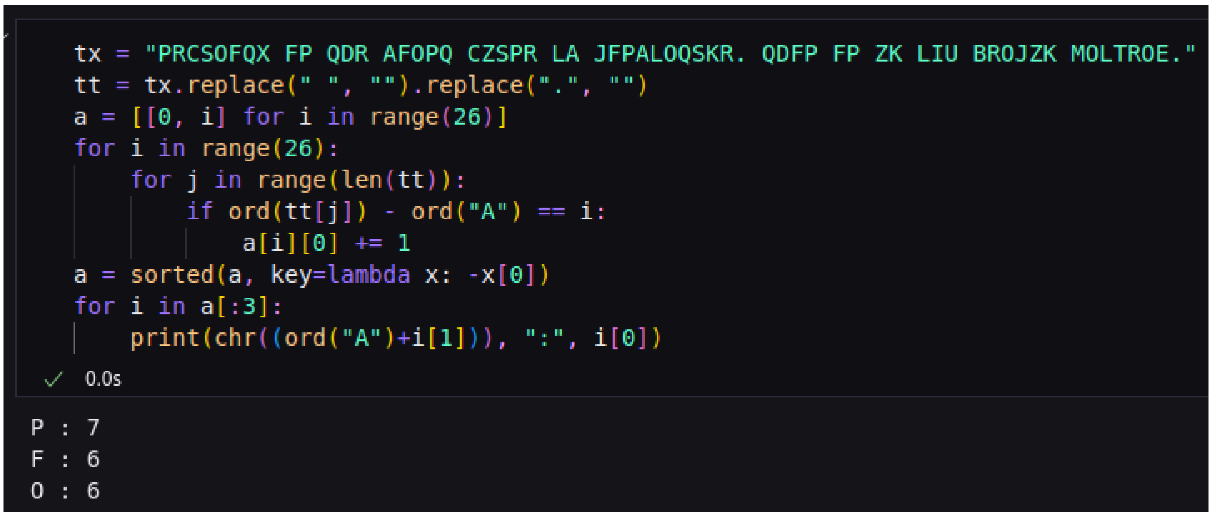
1. Encryption and Statistical Analysis

a.)



```
tx = "PRCSOFQX FP QDR AFOPQ CZSPR LA JFPALOQSKR. QDFP FP ZK LIU BROJZK MOLTROE."
tt = tx.replace(" ", "").replace(".", "")
a = [[0, i] for i in range(26)]
for i in range(26):
    for j in range(len(tt)):
        if ord(tt[j]) - ord("A") == i:
            a[i][0] += 1
a = sorted(a, key=lambda x: -x[0])
for i in a[:3]:
    print(chr((ord("A")+i[1])), ":", i[0])
```

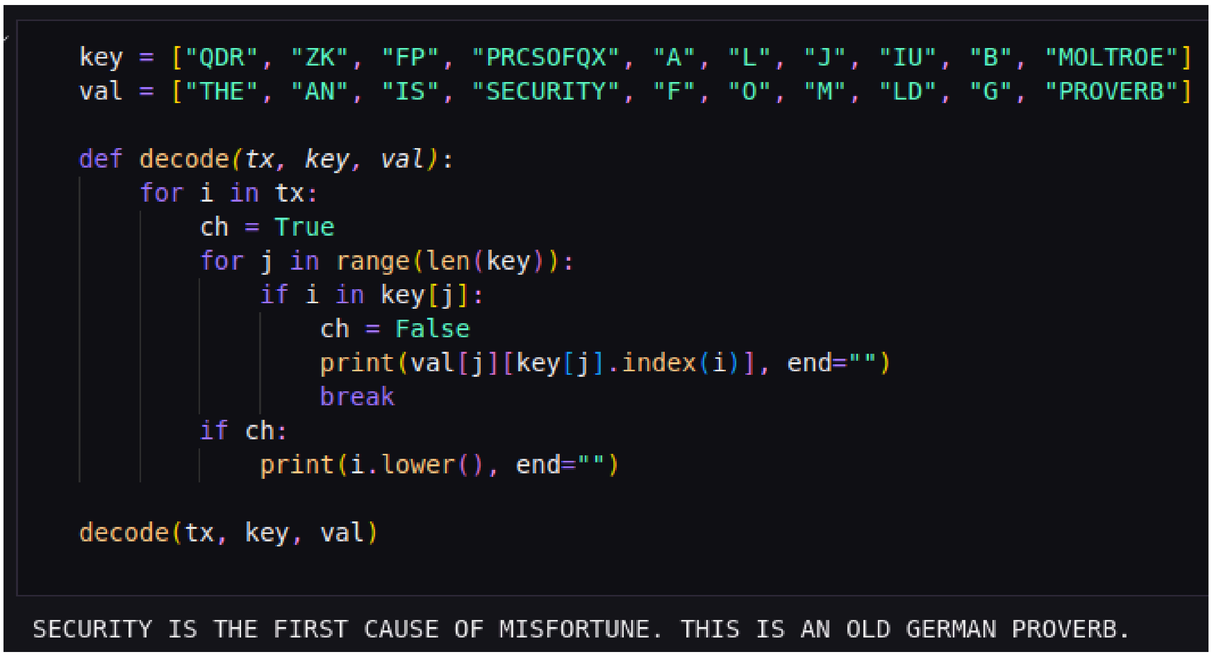
✓ 0.0s

```
P : 7
F : 6
O : 6
```

b.) Three-letter words : the, and, for
Two-letter words : of, to, in

Yes, these give me very clear hints.

c.) This took my time 90 minutes.

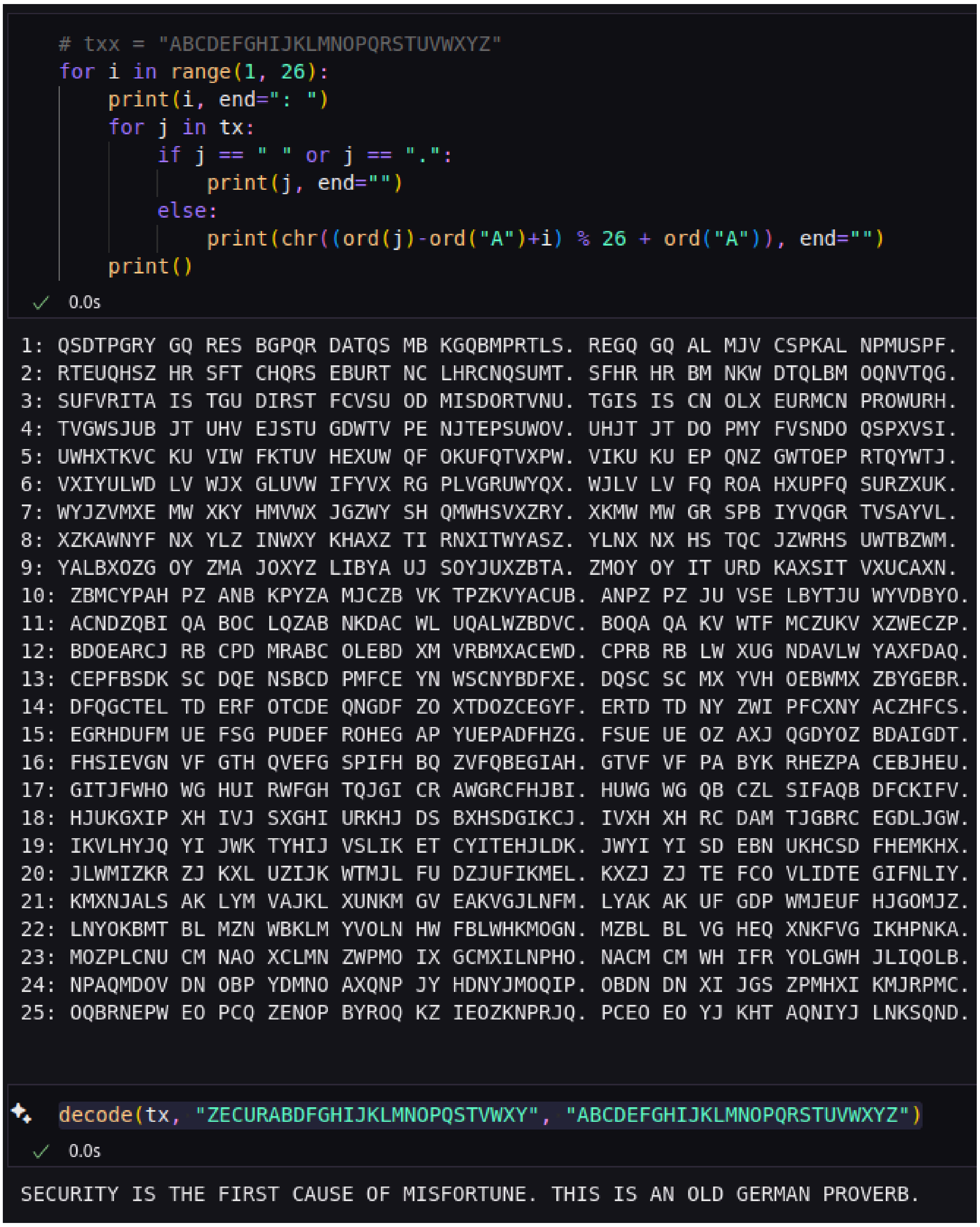


```
key = ["QDR", "ZK", "FP", "PRCSOFQX", "A", "L", "J", "IU", "B", "MOLTROE"]
val = ["THE", "AN", "IS", "SECURITY", "F", "O", "M", "LD", "G", "PROVERB"]

def decode(tx, key, val):
    for i in tx:
        ch = True
        for j in range(len(key)):
            if i in key[j]:
                ch = False
                print(val[j][key[j].index(i)], end="")
                break
        if ch:
            print(i.lower(), end="")

decode(tx, key, val)
```

```
SECURITY IS THE FIRST CAUSE OF MISFORTUNE. THIS IS AN OLD GERMAN PROVERB.
```

d.) From English sentence structure, ZK should be AN, QDFP FP should be THIS IS.
I tried substitute word that i know until got all message.

e.) Might be since I know that most character shift by 3 (sentence seem readable) and take some character to front which save my time than guess starting word like above.



```
# txx = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
for i in range(1, 26):
    print(i, end=": ")
    for j in tx:
        if j == " " or j == ".":
            print(j, end="")
        else:
            print(chr((ord(j)-ord("A")+i) % 26 + ord("A")), end="")
    print()
```

✓ 0.0s

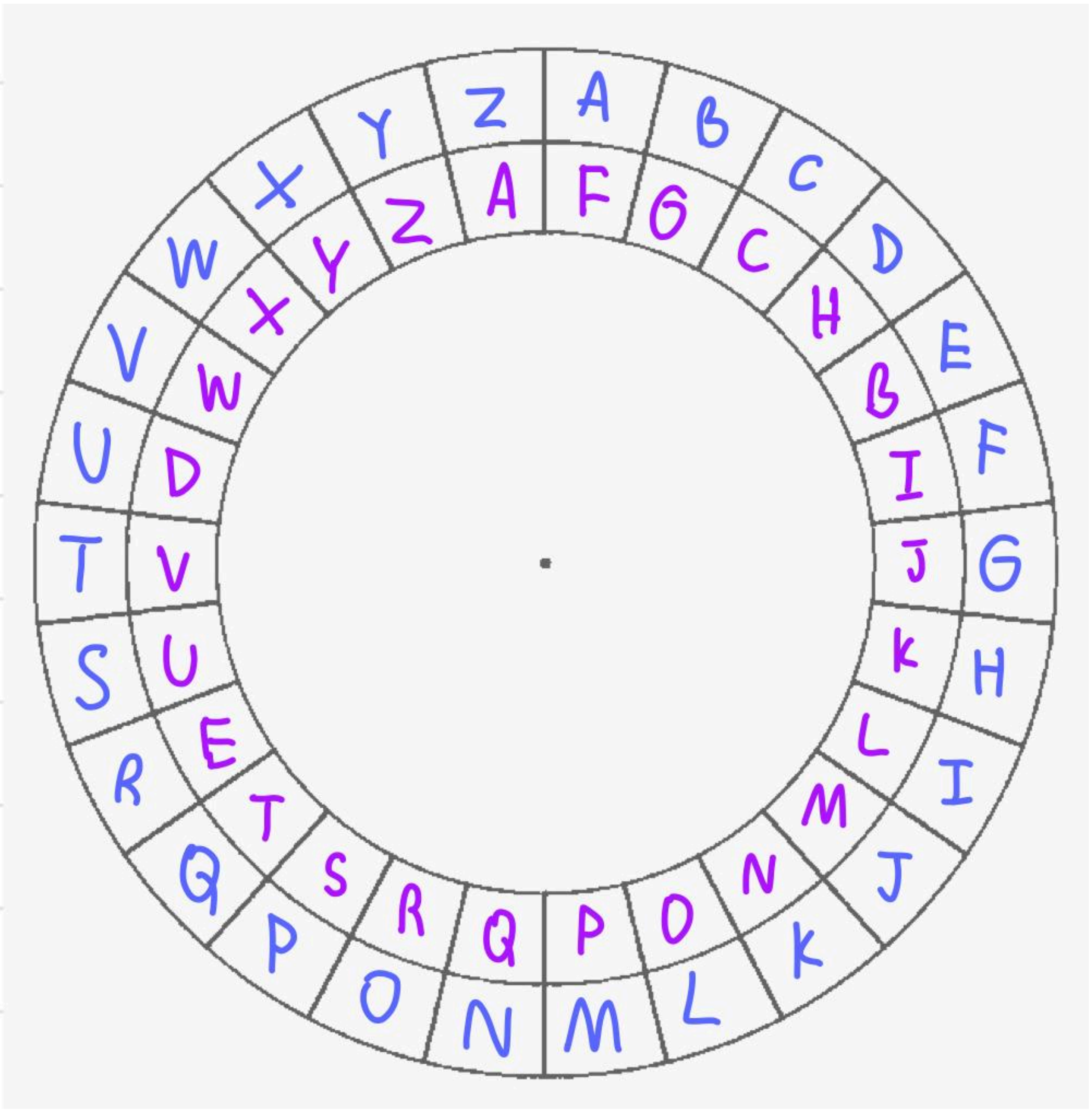
```
1: QSDTPGRY GQ RES BGPQR DATQS MB KGQBMPRTLS. REGQ GQ AL MJV CSPKAL NPMUSPF.
2: RTEUQHSZ HR SFT CHQRS EBURT NC LHRCNQSUMT. SFHR HR BM NKW DTQLBM OQNVTQG.
3: SUFVRITA IS TGU DIRST FCVSU OD MISDORTVNU. TGIS IS CN OLX EURMCN PROWURH.
4: TVGWSJUB JT UHV EJSTU GDWTV PE NJTEPSUWOV. UHJT JT DO PMY FVSNDO QSPXVSI.
5: UWHXTKVC KU VIW FKTUV HEXUW QF OKUFQTVXPW. VIKU KU EP QNZ GWTOEP RTQYWTJ.
6: VXIYULWD LV WJX GLUVW IFYVX RG PLVGRUWYQX. WJLV LV FQ ROA HXUPFQ SURZXUK.
7: WYJZVMXE MW XKY HMVWX JGZWY SH QMWHSVXZRY. XKMW MW GR SPB IYVQGR TVSAYVL.
8: XZKAWNYF NX YLZ INWXY KHAXZ TI RNXITWYASZ. YLNX NX HS TQC JZWRHS UWTBZWM.
9: YALBXOZG OY ZMA JOXYZ LIBYA UJ SOYJUXZBTA. ZMOY OY IT URD KAXSIT VXUCAXN.
10: ZBMCYPAH PZ ANB KPYZA MJCZB VK TPZKVYACUB. ANPZ PZ JU VSE LBYTJU WYVDBYO.
11: ACNDZQBI QA BOC LQZAB NKDAC WL UQALWZBDVC. BOQA QA KV WTF MCZUKV XZWECZP.
12: BDOEARCJ RB CPD MRABC OLEBD XM VRBMXACEWD. CPRB RB LW XUG NDAVLW YAXFDAQ.
13: CEPFBSDK SC DQE NSBCD PMFCE YN WSCNYBDFXE. DQSC SC MX YVH OEBWMX ZBYGEBR.
14: DFQGCTEL TD ERF OTCDE QNGDF ZO XTDOZCEGYF. ERTD TD NY ZWI PFCXNY ACZHFCS.
15: EGRHDUFM UE FSG PUDEF ROHEG AP YUEPADFHZG. FSUE UE OZ AXJ QGDYOZ BDAIGDT.
16: FHSIEVGN VF GTH QVEFG SPIFH BQ ZVFQBEGIAH. GTVF VF PA BYK RHEZPA CEBJHEU.
17: GITJFWHO WG HUI RWFGH TQJGI CR AWGRCFHJBI. HUWG WG QB CZL SIFAQB DFCKIFV.
18: HJUKGXIP XH IVJ SXGHI URKHJ DS BXHSDGIKCJ. IVXH XH RC DAM TJGBRC EGDLJGW.
19: IKVLHYJQ YI JWK TYHIJ VSLIK ET CYITEHJLDK. JWYI YI SD EBN UKHCSD FHEMKHX.
20: JLWMIZKR ZJ KXL UZIJK WTMJL FU DZJUFIKMEL. KXZJ ZJ TE FCO VLIDTE GIFNLIY.
21: KMXNJALS AK LYM VAJKL XUNKM GV EAKVGJLNFM. LYAK AK UF GDP WMJEUF HJGOMJZ.
22: LNYOKBMT BL MZN WBKLM YVOLN HW FBLWHKMOGN. MZBL BL VG HEQ XNKFVG IKHPNKA.
23: MOZPLCNU CM NAO XCLMN ZWPMO IX GCMXILNPHO. NACM CM WH IFR YOLGWH JLIQOLB.
24: NPAQMDOV DN OBP YDMNO AXQNP JY HDNYJMOQIP. OBDN DN XI JGS ZPMHXI KMJRPMC.
25: OQBRNEPW EO PCQ ZENOP BYROQ KZ IEOZKNPRJQ. PCEO EO YJ KHT AQNIYJ LNKSQND.
```

```
decode(tx, "ZECURABDFGHIJKLMNOPQSTVWXY", "ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

✓ 0.0s

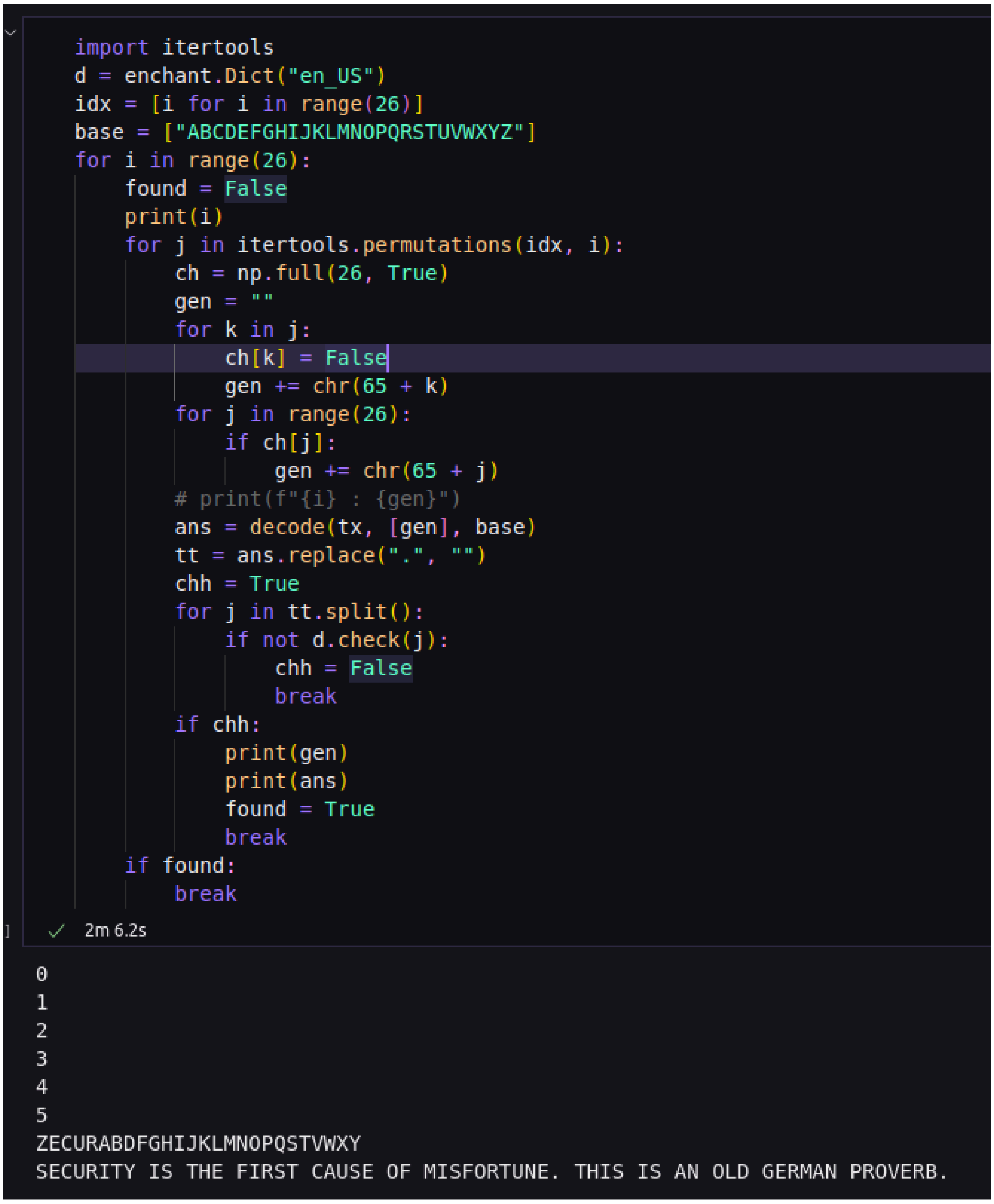
```
SECURITY IS THE FIRST CAUSE OF MISFORTUNE. THIS IS AN OLD GERMAN PROVERB.
```

f.)



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

g.) Since this is Caesar encrypt, so I choose r characters for 26 characters then permeate at front of mapping and left the rest then brute force search till every word is accepted by dictionary.



```python
import itertools
d = enchant.Dict("en_US")
idx = [i for i in range(26)]
base = ["ABCDEFGHIJKLMNOPQRSTUVWXYZ"]
for i in range(26):
    found = False
    print(i)
    for j in itertools.permutations(idx, i):
        ch = np.full(26, True)
        gen = ""
        for k in j:
            ch[k] = False
            gen += chr(65 + k)
        for j in range(26):
            if ch[j]:
                gen += chr(65 + j)
        # print(f"{i} : {gen}")
        ans = decode(tx, [gen], base)
        tt = ans.replace(".", "")
        chh = True
        for j in tt.split():
            if not d.check(j):
                chh = False
                break
        if chh:
            print(gen)
            print(ans)
            found = True
            break
    if found:
        break
```

✓ 2m 6.2s

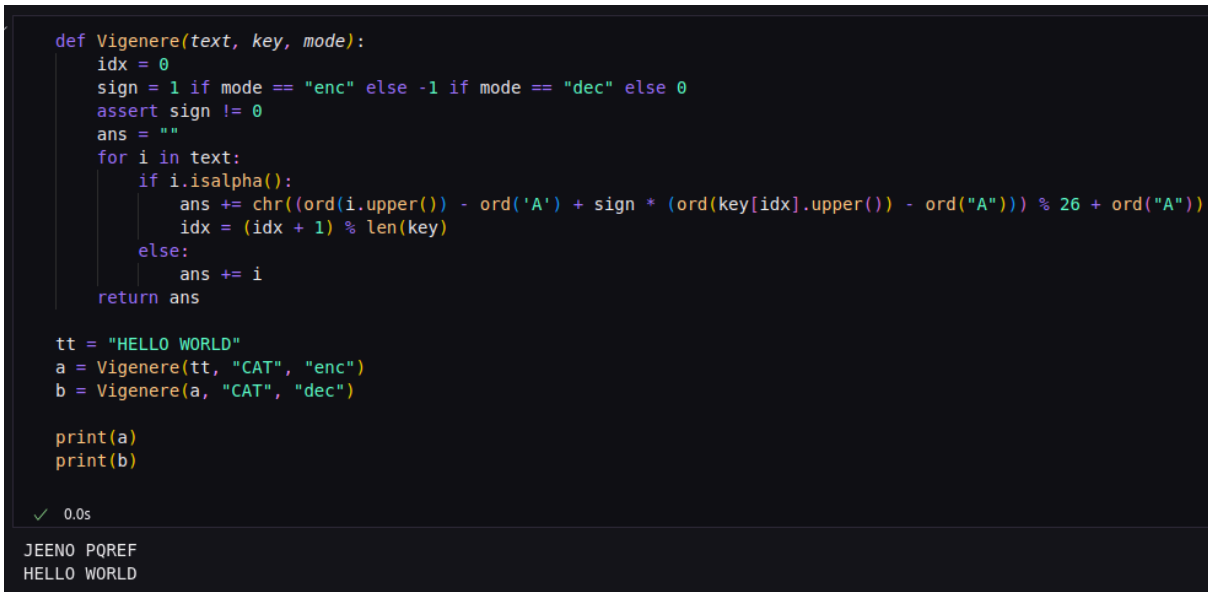
```
0
1
2
3
4
5
ZECURABDFGHIJKLMNOPQSTVWXY
SECURITY IS THE FIRST CAUSE OF MISFORTUNE. THIS IS AN OLD GERMAN PROVERB.
```

2. Symmetric Encryption

a.) To encode Vigenère text, the encoder take key then shift i-th character of text by the order of key's character at index i % len(key) which A is order 0 ex. HELLO WORLD with key CAT -> [2, 0, 18] ; H+2, E+0, L+18, L+2, … = JEENO PQREF

b.) If the key is "CAT", the security is very low since a hacker only needs to try a 3-character key, which is 26^3 = 17,576 possibilities + smaller variations, making it easy to crack. It is recommended to use a longer key. When the Vigenère key is long enough, it becomes harder to crack than the Caesar cipher because the same word is unlikely to map to the same encoded word in different positions.

c.)



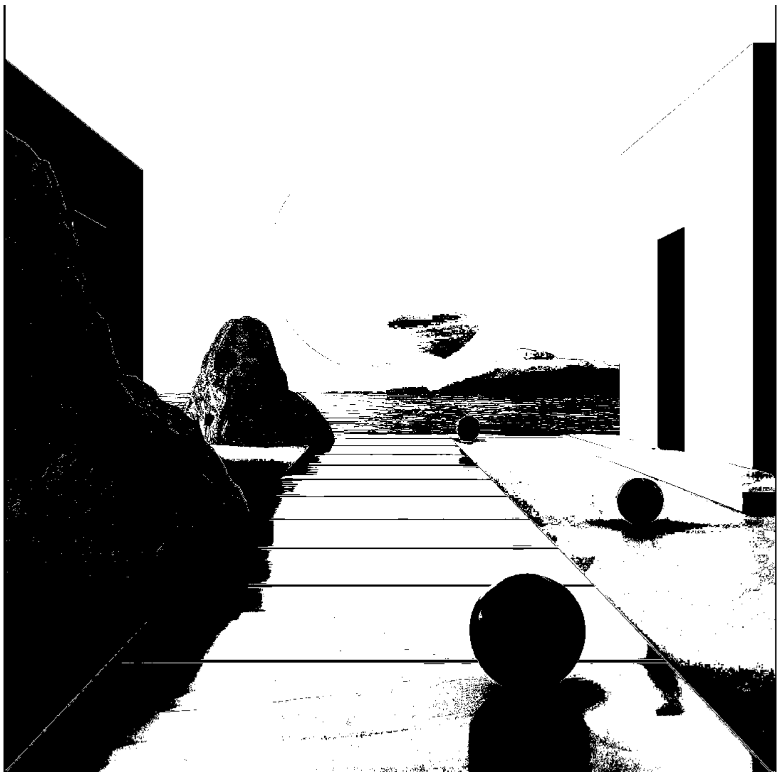
```python
def Vigenere(text, key, mode):
    idx = 0
    sign = 1 if mode == "enc" else -1 if mode == "dec" else 0
    assert sign != 0
    ans = ""
    for i in text:
        if i.isalpha():
            ans += chr((ord(i.upper()) - ord('A') + sign * (ord(key[idx].upper()) - ord("A"))) % 26 + ord("A"))
            idx = (idx + 1) % len(key)
        else:
            ans += i
    return ans

tt = "HELLO WORLD"
a = Vigenere(tt, "CAT", "enc")
b = Vigenere(a, "CAT", "dec")

print(a)
print(b)
```

✓ 0.0s

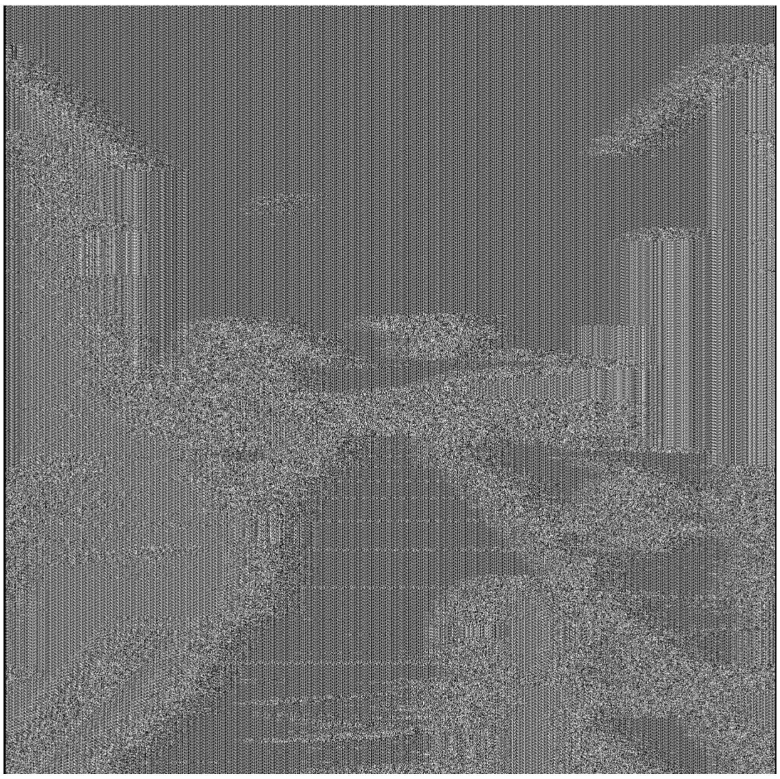
```
JEENO PQREF
HELLO WORLD
```

3. Mode in Block Cipher

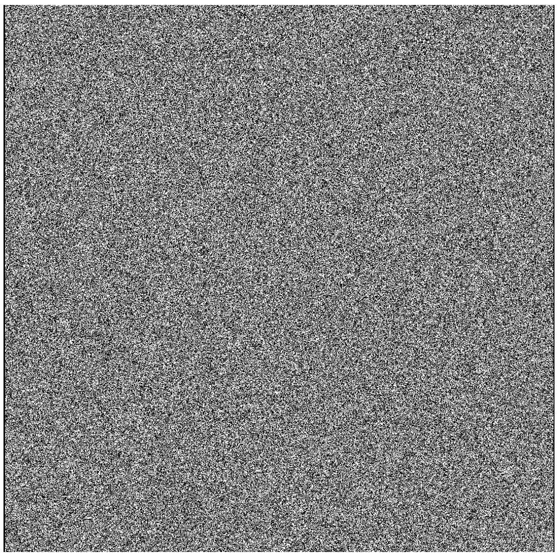
a.) Original



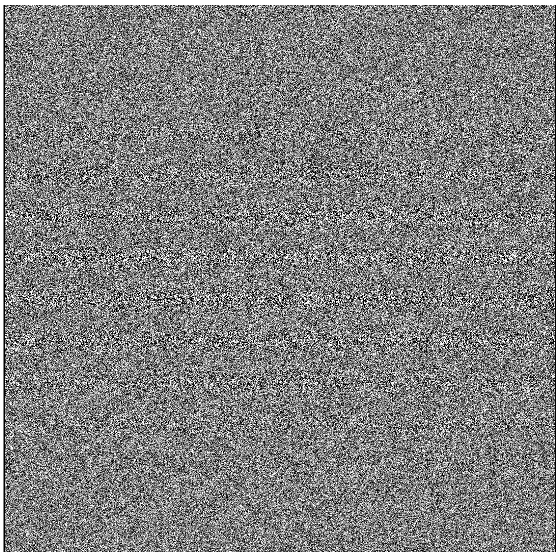
d.) ECB



e.) Chaining



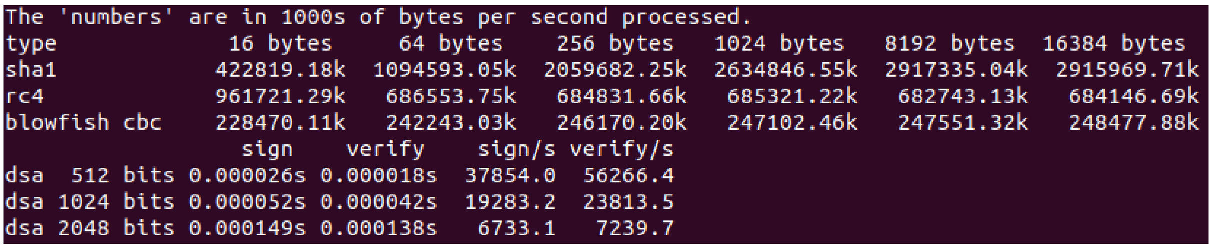
Cipher Feedback



f.) If we consider blocks of data, there are 3 main types: fully black, fully white, and mixed black/white (since this is a bitmap image). ECB will encode every block with the same key, making patterns noticeable. In contrast, CBC and CFB take context from the previous result, though they differ in the order and what they encode. This ensures that the same data pattern in different positions produces different results.

4. Encryption Protocol - Digital Signature

a.) I used the speed command of OpenSSL to test SHA1, RC4, BF, and DSA.



```
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes  16384 bytes
sha1            422819.18k  1094593.05k  2059682.25k  2634846.55k  2917335.04k  2915969.71k
rc4             961721.29k   686553.75k   684831.66k   685321.22k   682743.13k   684146.69k
blowfish cbc    228470.11k   242243.03k   246170.20k   247102.46k   247551.32k   248477.88k
                  sign    verify    sign/s verify/s
dsa  512 bits 0.000026s 0.000018s  37854.0  56266.4
dsa 1024 bits 0.000052s 0.000042s  19283.2  23813.5
dsa 2048 bits 0.000149s 0.000138s   6733.1   7239.7
```

b.) The results showed that SHA1 and BF are faster with longer lengths, with SHA1 being faster than BF. However, DSA becomes slower as the length increases. Additionally, RC4 appears to have a throughput limit.

c.) Let's say the sender has a message and does the following:

- message_hashed = hash_func(message)
- signature = encode(message_hashed, private_key)

Then, the sender sends both the message and the signature.

On the receiver's side:

- hashed_1 = hash_func(message)
- hashed_2 = decode(signature, public_key)

Finally, the receiver checks if **hashed_1 == hashed_2**. If they are the same, it means the message is from the real sender and has not been edited.

A Digital Signature works by first hashing the message, then encoding it with a private key. It is nearly impossible to perform a heuristic reverse search to recover the private key because the hashed message is random, and the algorithm guarantees that:

dec(enc(hash_func(message), pri_key), pub_key) == hash_func(message)

For a hacker to forge the signature, they would need to either:

1. Find an edited message such that **hash_func(message_edit) == hash_func(message)**, which is nearly impossible due to the large mapping space of the hash function.
2. Alter the signature in such a way that **dec(new_signature, public_key) == dec(signature, public_key)**, which is also impossible because the decryption with the public key is an asymmetric mapping function.

Alternatively, if they try to edit both the message and the signature, they would face the problem that the public key can only be used for decoding, not encoding, making the message uneditable.