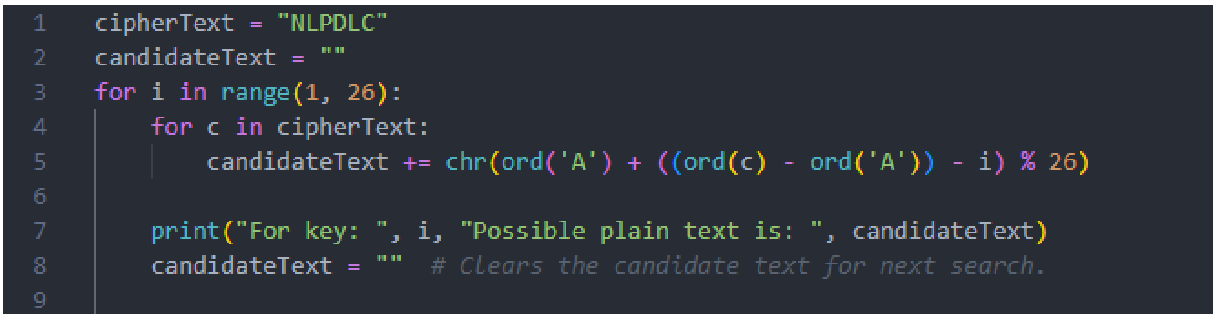
Muhammet Taha ÇAKMAK – 29183

Q1:

- Since this is a classical shift cipher for an 26 letters alphabet, it is very easy to do a brute-force attack.
- By using following python script (Q1.py), we can find the key and message between 25 possible shifts:



```python
cipherText = "NLPDLC"
candidateText = ""
for i in range(1, 26):
    for c in cipherText:
        candidateText += chr(ord('A') + ((ord(c) - ord('A')) - i) % 26)

    print("For key: ", i, "Possible plain text is: ", candidateText)
    candidateText = ""  # Clears the candidate text for next search.

```

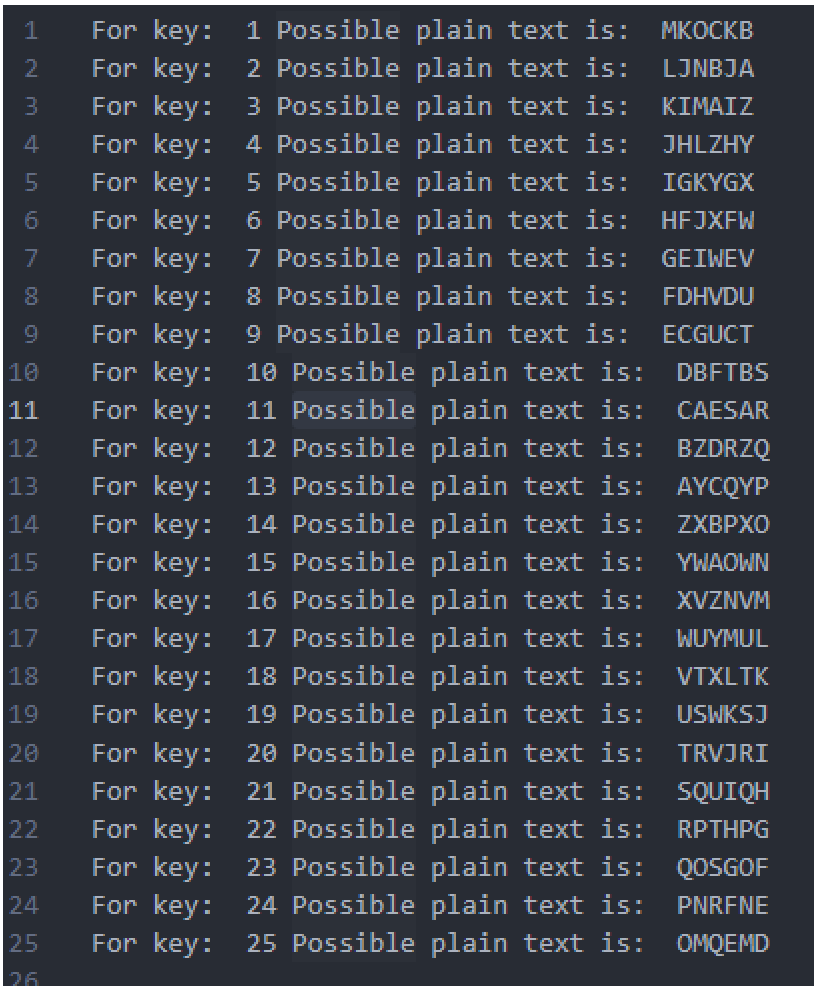
Detailed explanation of the code:

- We first give the cipher text and another string to keep candidate strings.
- The key is between [0,25], by using outer for loop we check every possible key candidate.
- By using second for loop we reversely shift every character in the plaintext. Here by using Ord() method we get ASCII values of the letters.
- ord(c) – ord('A') gets the index of the letter since all the letters are consecutively ordered in the list. After decreasing it by i, we get corresponding char by the same ASCII manipulation and add it to candidate text.
- After completing inner loop, we clear the candidateText variable and repeat the process for the next key (i).

Result:

- For **key = 11** we get the **CAESAR** as the plain text, which is the only meaningful output among all the 25 output.
- All outputs are in the **outputq1.txt**.

Answer for Q1: Key: 11 & Plain text: CAESAR

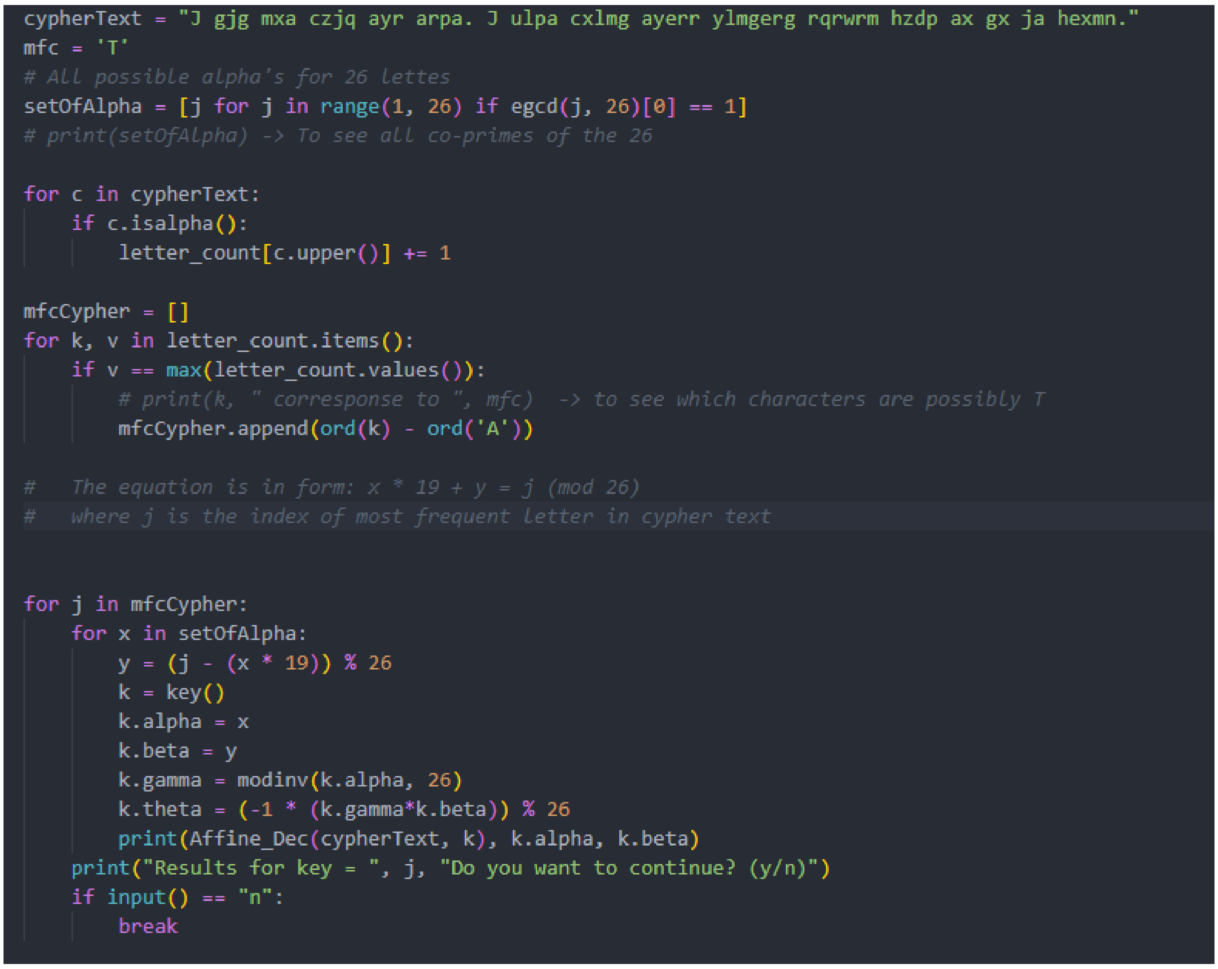


```
For key:  1 Possible plain text is:  MKOCKB
For key:  2 Possible plain text is:  LJNBJA
For key:  3 Possible plain text is:  KIMAIZ
For key:  4 Possible plain text is:  JHLZHY
For key:  5 Possible plain text is:  IGKYGX
For key:  6 Possible plain text is:  HFJXFW
For key:  7 Possible plain text is:  GEIWEV
For key:  8 Possible plain text is:  FDHVDU
For key:  9 Possible plain text is:  ECGUCT
For key:  10 Possible plain text is:  DBFTBS
For key:  11 Possible plain text is:  CAESAR
For key:  12 Possible plain text is:  BZDRZQ
For key:  13 Possible plain text is:  AYCQYP
For key:  14 Possible plain text is:  ZXBPXO
For key:  15 Possible plain text is:  YWAOWN
For key:  16 Possible plain text is:  XVZNVM
For key:  17 Possible plain text is:  WUYMUL
For key:  18 Possible plain text is:  VTXLTK
For key:  19 Possible plain text is:  USWKSJ
For key:  20 Possible plain text is:  TRVJRI
For key:  21 Possible plain text is:  SQUIQH
For key:  22 Possible plain text is:  RPTHPG
For key:  23 Possible plain text is:  QOSGOF
For key:  24 Possible plain text is:  PNRFNE
For key:  25 Possible plain text is:  OMQEMD
```

Outputq1.txt

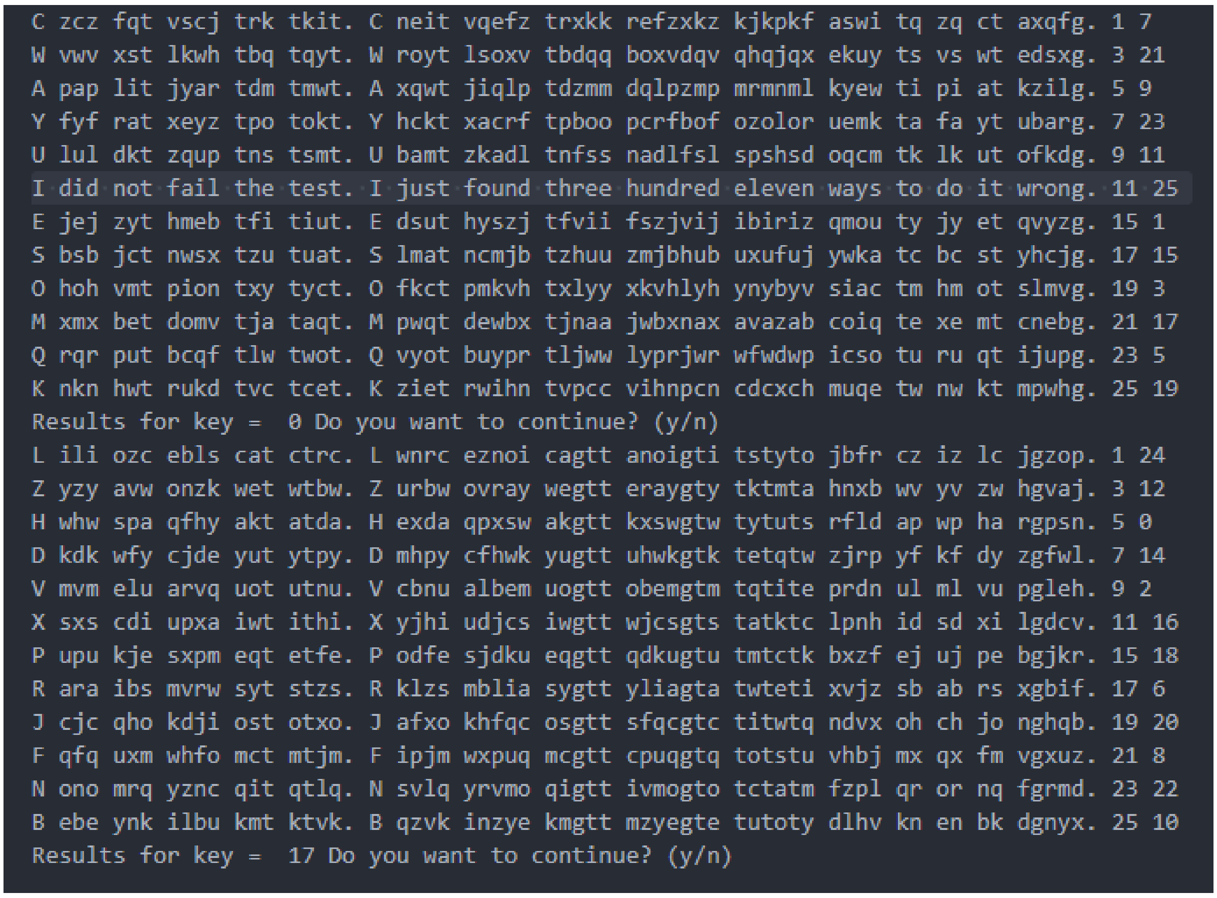
Q2:

- Firstly, we need to detect which character(s) are the most frequent characters in the cipher text by counting them one by one.
- After deciding to the most frequent character, we will use the affine cipher equation to produce all possible keys.
- Since there are 12 different co-primes of the 26, we can produce 12 possible keys for each candidate of most frequent character of the cipher text.
- After solving the affine cipher equation for 12 different keys for each corresponding known text – plaintext couples. We can apply brute force to see which key pair actually work.
- Following piece of code (Q2.py) does this process for us.



```python
cypherText = "J gjg mxa czjq ayr arpa. J ulpa cxlmg ayerr ylmgerg rqrwrm hzdp ax gx ja hexmn."
mfc = 'T'
# All possible alpha's for 26 lettes
setOfAlpha = [j for j in range(1, 26) if egcd(j, 26)[0] == 1]
# print(setOfAlpha) -> To see all co-primes of the 26

for c in cypherText:
    if c.isalpha():
        letter_count[c.upper()] += 1

mfcCypher = []
for k, v in letter_count.items():
    if v == max(letter_count.values()):
        # print(k, " corresponse to ", mfc)  -> to see which characters are possibly T
        mfcCypher.append(ord(k) - ord('A'))

#   The equation is in form: x * 19 + y = j (mod 26)
#   where j is the index of most frequent letter in cypher text


for j in mfcCypher:
    for x in setOfAlpha:
        y = (j - (x * 19)) % 26
        k = key()
        k.alpha = x
        k.beta = y
        k.gamma = modinv(k.alpha, 26)
        k.theta = (-1 * (k.gamma*k.beta)) % 26
        print(Affine_Dec(cypherText, k), k.alpha, k.beta)
    print("Results for key = ", j, "Do you want to continue? (y/n)")
    if input() == "n":
        break
```

Results:



```
C zcz fqt vscj trk tkit. C neit vqefz trxkk refzxkz kjkpkf aswi tq zq ct axqfg. 1 7
W vwv xst lkwh tbq tqyt. W royt lsoxv tbdqq boxvdqv qhqjqx ekuy ts vs wt edsxg. 3 21
A pap lit jyar tdm tmwt. A xqwt jiqlp tdzmm dqlpzmp mrmnml kyew ti pi at kzilg. 5 9
Y fyf rat xeyz tpo tokt. Y hckt xacrf tpboo pcrfbof ozolor uemk ta fa yt ubarg. 7 23
U lul dkt zqup tns tsmt. U bamt zkadl tnfss nadlfsl spshsd oqcm tk lk ut ofkdg. 9 11
I did not fail the test. I just found three hundred eleven ways to do it wrong. 11 25
E jej zyt hmeb tfi tiut. E dsut hyszj tfvii fszjvij ibiriz qmou ty jy et qvyzg. 15 1
S bsb jct nwsx tzu tuat. S lmat ncmjb tzhuu zmjbhub uxufuj ywka tc bc st yhcjg. 17 15
O hoh vmt pion txy tyct. O fkct pmkvh txlyy xkvhlyh ynybyv siac tm hm ot slmvg. 19 3
M xmx bet domv tja taqt. M pwqt dewbx tjnaa jwbxnax avazab coiq te xe mt cnebg. 21 17
Q rqr put bcqf tlw twot. Q vyot buypr tljww lyprjwr wfwdwp icso tu ru qt ijupg. 23 5
K nkn hwt rukd tvc tcet. K ziet rwihn tvpcc vihnpcn cdcxch muqe tw nw kt mpwhg. 25 19
Results for key =  0 Do you want to continue? (y/n)
L ili ozc ebls cat ctrc. L wnrc eznoi cagtt anoigti tstyto jbfr cz iz lc jgzop. 1 24
Z yzy avw onzk wet wtbw. Z urbw ovray wegtt eraygty tktmta hnxb wv yv zw hgvaj. 3 12
H whw spa qfhy akt atda. H exda qpxsw akgtt kxswgtw tytuts rfld ap wp ha rgpsn. 5 0
D kdk wfy cjde yut ytpy. D mhpy cfhwk yugtt uhwkgtk tetqtw zjrp yf kf dy zgfwl. 7 14
V mvm elu arvq uot utnu. V cbnu albem uogtt obemgtm tqtite prdn ul ml vu pgleh. 9 2
X sxs cdi upxa iwt ithi. X yjhi udjcs iwgtt wjcsgts tatktc lpnh id sd xi lgdcv. 11 16
P upu kje sxpm eqt etfe. P odfe sjdku eqgtt qdkugtu tmtctk bxzf ej uj pe bgjkr. 15 18
R ara ibs mvrw syt stzs. R klzs mblia sygtt yliagta twteti xvjz sb ab rs xgbif. 17 6
J cjc qho kdji ost otxo. J afxo khfqc osgtt sfqcgtc titwtq ndvx oh ch jo nghqb. 19 20
F qfq uxm whfo mct mtjm. F ipjm wxpuq mcgtt cpuqgtq totstu vhbj mx qx fm vgxuz. 21 8
N ono mrq yznc qit qtlq. N svlq yrvmo qigtt ivmogto tctatm fzpl qr or nq fgrmd. 23 22
B ebe ynk ilbu kmt ktvk. B qzvk inzye kmgtt mzyegte tutoty dlhv kn en bk dgnyx. 25 10
Results for key =  17 Do you want to continue? (y/n)
```

outputq2.txt

- For Alpha = 11 and Beta = 25 we obtained the message: “I did not fail the test. I just found three hundred eleven ways to do it wrong.” Which is the only meaningful output.
- All the outputs are in **outputq2.txt.**

Answer:

Key : (Alpha: 11, Beta: 25) T

Message: “I did not fail the test. I just found three hundred eleven ways to do it wrong.”

Q3:

- To decide the modulus, we need to know how many possible bigram exist. Our alphabet contains 30 characters. A bigram could take 900 different values, (30 different character for the first letter and 30 for the second one so 30 x 30 = 900). That's why our calculations must be in mod 900.
- Since this is a affine cipher, we need to know how many different alpha and beta are possible.
- Beta is always equal to the number of characters in the alphabet (here number of bigrams) which is 900.
- For a one-to-one correspondence alpha must be co-prime to number of characters in the alphabet. We need to calculate phi(900).
- Phi(900) = phi(9) x phi(4) x phi(25) since the function is multiplicative.
- Phi(9) = 6, Phi(4) = 2 and phi(25) = 20
- 6 x 2 x 20 = 240 different alpha is possible.
- # of possible keys is 240 x 900 = 216.000

  ANSWER: The modulus is 900 and key space has 216000 possible keys.

Q4:

- **This affine cipher is secure against letter frequency analysis** attacks since encoding of the message is not one by one mapped to another character in the cipher text.
- This phenomenon can be explained by **Shannon's diffusion property**. Statistical properties of the single letters in the plain text will not be protected in the cipher text since probability of obtaining a bigram is not related the letters one by one.

- From a more mathematical perspective our claim is that:

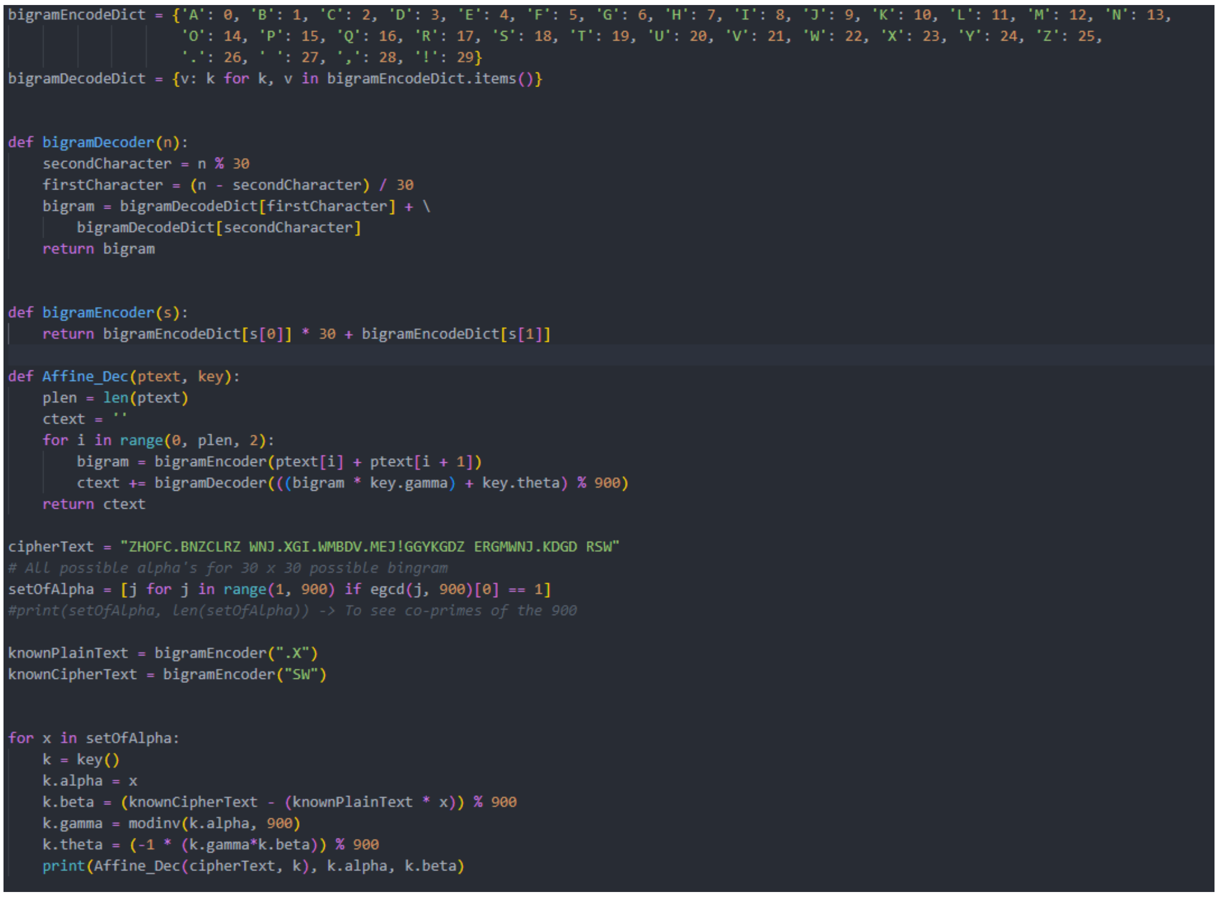
$$P(k_1k_2) \neq P(k_1) * P(k_2)$$

- The reason behind that is totally linguistic. For example, normally finding a letter 's' is lower than finding a letter 'a'. However, if we know that the first character is 'i' than the second character is more likely to be 's' than 'a' since 'is' is a very common bigram in English.
- As a conclusion, probability of first and second letter are **not independent (**First character changes probability of the second character as explained above). That's why by only knowing the single letter frequencies we cannot decide the frequencies of bigrams.
- However, even though our affine cipher is secure against single letter frequency analysis, it is not secure against all the statistical attacks. Someone can produce a new frequency table for 900 possible combinations and can use this table to attack the system.

Q5:

- We start to solve this question by using 2 important information. Firstly, it is stated that our sentence is ending with a dot and it has odd number of characters. That means our plain text ending with a ".X" bigram which corresponded to "SW" in cipher text.
- Since we have 1 equation but 2 unknowns (alpha, beta), it is not possible to achieve a direct solution. However, we know that possible solutions are finite. Basically, it is enough to check for every possible alpha (which are the coprime of the 900) and beta. We have 240 possible alpha beta pair.
- By using all the information above we can write a simple pyhton script to check 240 possibility.
- This question is solved by the following script (Q5.py).
- Firstly, we defined 2 dictionaries one of them to use during the encoding and the other one is for the decoding.
- Then, I defined 2 functions for encoding and decoding. Encoding is simply 30*FirstCharacter + SecondCharacter. For decoding, when we take mod 30 of the encoding number we obtain index of the second character. By subtracting second character and dividing by 30, we obtain index of the first chatacer. (bigramEncoder & bigramDecoder functions are doing these things)

- By using the for loop we checked the all possible alfa values and produced corresponding beta, gamma and theta characters. These characters used to produce the key.
- After producing the gamma and theta pairs encryption becomes trivial. We just multiply by gamma and subtract the theta. That's what Affine_Dec() does.

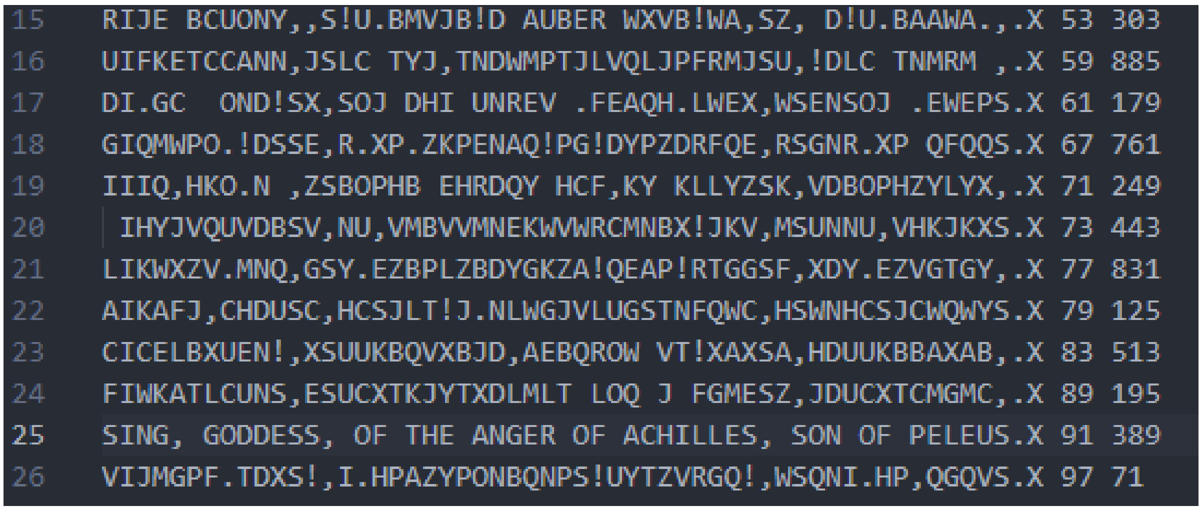


```python
bigramEncodeDict = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13,
                    'O': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25,
                    '.': 26, ' ': 27, ',': 28, '!': 29}
bigramDecodeDict = {v: k for k, v in bigramEncodeDict.items()}


def bigramDecoder(n):
    secondCharacter = n % 30
    firstCharacter = (n - secondCharacter) / 30
    bigram = bigramDecodeDict[firstCharacter] + \
        bigramDecodeDict[secondCharacter]
    return bigram


def bigramEncoder(s):
    return bigramEncodeDict[s[0]] * 30 + bigramEncodeDict[s[1]]

def Affine_Dec(ptext, key):
    plen = len(ptext)
    ctext = ''
    for i in range(0, plen, 2):
        bigram = bigramEncoder(ptext[i] + ptext[i + 1])
        ctext += bigramDecoder(((bigram * key.gamma) + key.theta) % 900)
    return ctext

cipherText = "ZHOFC.BNZCLRZ WNJ.XGI.WMBDV.MEJ!GGYKGDZ ERGMWNJ.KDGD RSW"
# All possible alpha's for 30 x 30 possible bingram
setOfAlpha = [j for j in range(1, 900) if egcd(j, 900)[0] == 1]
#print(setOfAlpha, len(setOfAlpha)) -> To see co-primes of the 900

knownPlainText = bigramEncoder(".X")
knownCipherText = bigramEncoder("SW")


for x in setOfAlpha:
    k = key()
    k.alpha = x
    k.beta = (knownCipherText - (knownPlainText * x)) % 900
    k.gamma = modinv(k.alpha, 900)
    k.theta = (-1 * (k.gamma*k.beta)) % 900
    print(Affine_Dec(cipherText, k), k.alpha, k.beta)
```

Q5.py

- After trying all possible keys, we will get the only meaningful output for alpha = 91 beta = 389 which is "SING, GODDESS, OF THE ANGER OF ACHILLES, SON OF PELEUS.X "
- All possible outputs are listed in **outputq5.txt**

Answer: SING, GODDESS, OF THE ANGER OF ACHILLES, SON OF PELEUS. (Alpha = 91, Beta = 389)



```
RIJE BCUONY,,S!U.BMVJB!D AUBER WXVB!WA,SZ, D!U.BAAWA.,.X 53 303
UIFKETCCANN,JSLC TYJ,TNDWMPTJLVQLJPFRMJSU,!DLC TNMRM ,.X 59 885
DI.GC  OND!SX,SOJ DHI UNREV .FEAQH.LWEX,WSENSOJ .EWEPS.X 61 179
GIQMWPO.!DSSE,R.XP.ZKPENAQ!PG!DYPZDRFQE,RSGNR.XP QFQQS.X 67 761
IIIQ,HKO.N ,ZSBOPHB EHRDQY HCF,KY KLLYZSK,VDBOPHZYLYX,.X 71 249
 IHYJVQUVDBSV,NU,VMBVVMNEKWVWRCMNBX!JKV,MSUNNU,VHKJKXS.X 73 443
LIKWXZV.MNQ,GSY.EZBPLZBDYGKZA!QEAP!RTGGSF,XDY.EZVGTGY,.X 77 831
AIKAFJ,CHDUSC,HCSJLT!J.NLWGJVLUGSTNFQWC,HSWNHCSJCWQWYS.X 79 125
CICELBXUEN!,XSUUKBQVXBJD,AEBQROW VT!XAXSA,HDUUKBBAXAB,.X 83 513
FIWKATLCUNS,ESUCXTKJYTXDLMLT LOQ J FGMESZ,JDUCXTCMGMC,.X 89 195
SING, GODDESS, OF THE ANGER OF ACHILLES, SON OF PELEUS.X 91 389
VIJMGPF.TDXS!,I.HPAZYPONBQNPS!UYTZVRGQ!,WSQNI.HP,QGQVS.X 97 71
```

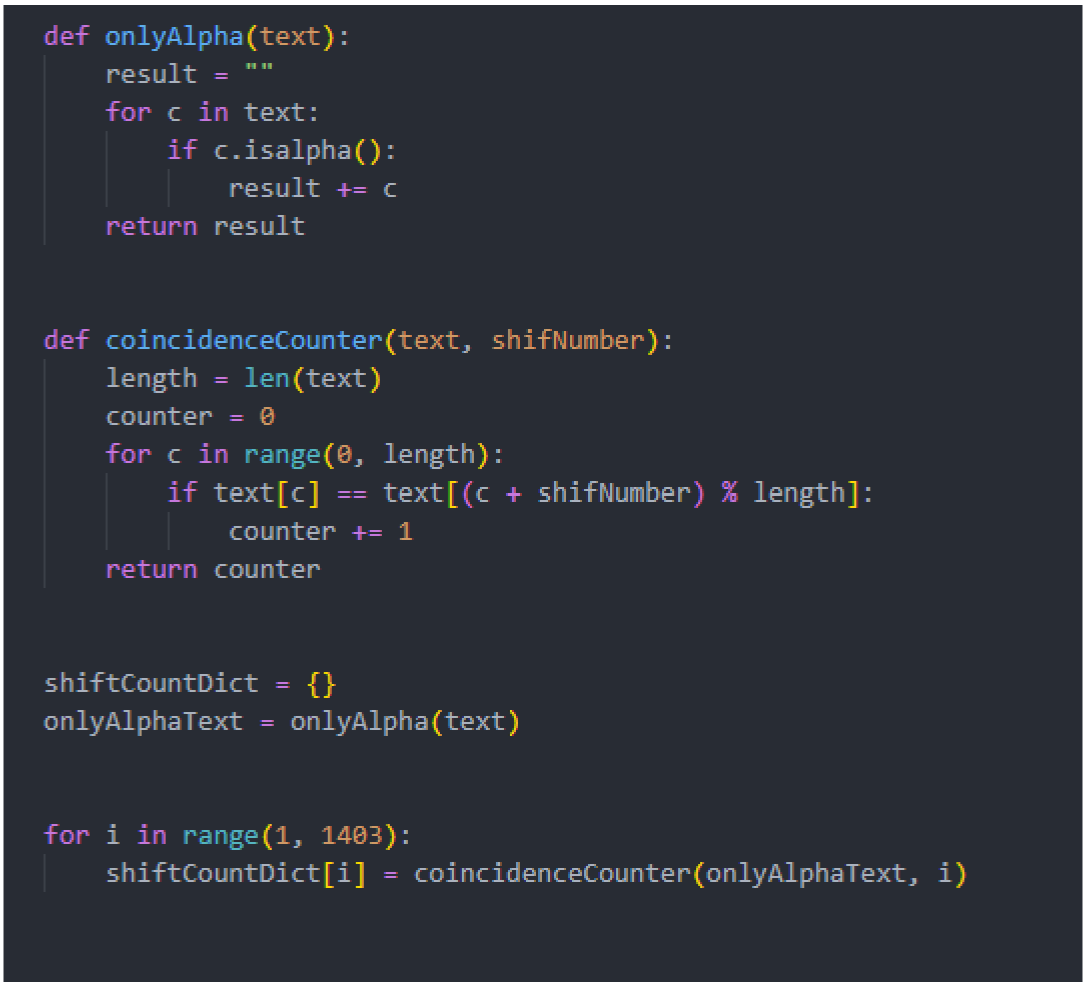
From outputq5.txt

Q6:

- Probability of getting any letter in plain text is a statistical property.
- $\boldsymbol{P_\alpha(a) + P_\alpha(b) + P_\alpha(c) + \cdots + P_\alpha(z) = 1}$ ➔ Even we don't know every individual probability we know that their sum will be one.
- Since every shift number is selected from uniformly distributed shift numbers in 29 letters alphabet, probability of obtaining any shift is equally likely and it is 1 / 29.
- $P_\beta(\gamma) = \frac{1}{29}\boldsymbol{P_\alpha(a)} + \frac{1}{29}\boldsymbol{P_\alpha(b)} + \frac{1}{29}\boldsymbol{P_\alpha(c)} + \cdots + \frac{1}{29}\boldsymbol{P_\alpha(z)}$
- The probability can be interpreted as follows, probability of obtaining any character (gamma) is equal to obtain correct shift for each possible letter in the plaintext times finding probability of that letter. Since all shifts coming from a uniformly distributed probability distribution. Doing the correct shift probability is always 1/29
- $P_\beta(\gamma) = \frac{1}{29}(\boldsymbol{P_\alpha(a) + P_\alpha(b) + P_\alpha(c) + \cdots + P_\alpha(z)}) = \mathbf{1\ /29}$

BONUS Question:

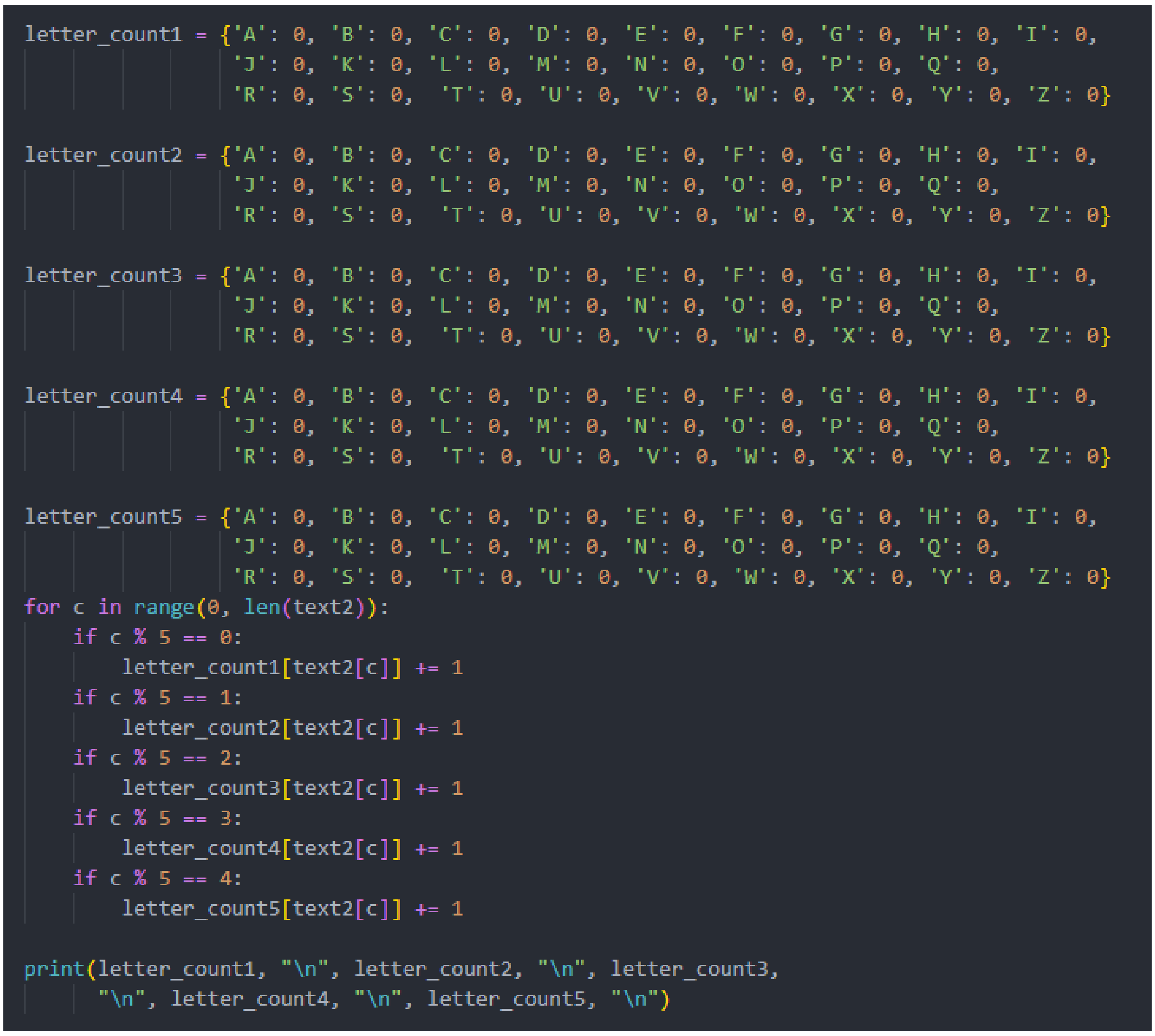
- To find key length we will check the coincidences for n shifted versions of the text. We will do this by using Q7Part1.py.



```python
def onlyAlpha(text):
    result = ""
    for c in text:
        if c.isalpha():
            result += c
    return result


def coincidenceCounter(text, shifNumber):
    length = len(text)
    counter = 0
    for c in range(0, length):
        if text[c] == text[(c + shifNumber) % length]:
            counter += 1
    return counter


shiftCountDict = {}
onlyAlphaText = onlyAlpha(text)


for i in range(1, 1403):
    shiftCountDict[i] = coincidenceCounter(onlyAlphaText, i)
```

Q7Part1.py

- onlyAlpha method removes the non-alphabetical characters from the string and creates a string only contains the letters in the cipher text.
- CoincidenceCounter counts the number of coincidences for decided shift number.
- Since our text has 1404 character for a full analysis I tried all possible shifts.
- When we analyze the outputs (available at Q7\outputq7_1.txt) we will observe a coincidence pick in every multiple of 5. That's why we have a strong evidence key length is 5.
- Now we will divide the characters into 5 different groups and count the letter frequencies for every individual group by using following piece of script.



```
letter_count1 = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0, 'H': 0, 'I': 0,
                 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O': 0, 'P': 0, 'Q': 0,
                 'R': 0, 'S': 0,  'T': 0, 'U': 0, 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0}

letter_count2 = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0, 'H': 0, 'I': 0,
                 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O': 0, 'P': 0, 'Q': 0,
                 'R': 0, 'S': 0,  'T': 0, 'U': 0, 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0}

letter_count3 = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0, 'H': 0, 'I': 0,
                 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O': 0, 'P': 0, 'Q': 0,
                 'R': 0, 'S': 0,  'T': 0, 'U': 0, 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0}

letter_count4 = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0, 'H': 0, 'I': 0,
                 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O': 0, 'P': 0, 'Q': 0,
                 'R': 0, 'S': 0,  'T': 0, 'U': 0, 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0}

letter_count5 = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0, 'H': 0, 'I': 0,
                 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O': 0, 'P': 0, 'Q': 0,
                 'R': 0, 'S': 0,  'T': 0, 'U': 0, 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0}
for c in range(0, len(text2)):
    if c % 5 == 0:
        letter_count1[text2[c]] += 1
    if c % 5 == 1:
        letter_count2[text2[c]] += 1
    if c % 5 == 2:
        letter_count3[text2[c]] += 1
    if c % 5 == 3:
        letter_count4[text2[c]] += 1
    if c % 5 == 4:
        letter_count5[text2[c]] += 1

print(letter_count1, "\n", letter_count2, "\n", letter_count3,
      "\n", letter_count4, "\n", letter_count5, "\n")
```

From Q7Part2.py

- Results from the script above shows that most frequent characters can be listed as follow:
  - mostFrequentCharacters1 = ['A', 'Q', 'F'] (Q is the most frequent)
  - mostFrequentCharacters2 = ['K', 'O', 'Z'] (K is the most frequent)
  - mostFrequentCharacters3 = ['O', 'Z',’D’] (Z is the most frequent)
  - mostFrequentCharacters4 = ['H', 'W'] (H is the most frequent)
  - mostFrequentCharacters5 = ['F', 'U'] (U is the most frequent)
- Firstly, let’s map every most frequent letter to ‘E’ and see the result:

THE CPNTRIAETAL QORCE ZN OUR ALANEE IS STTLL FELRFULWY STRZNG, ALJOSHA. T HAVE L LONGTNG FOC LIFE, LND I GZ ON LIGING IY SPITP OF LORIC. THZUGH I XAY NOE BELIPVE IN EHE OROER OF EHE UNTVERSP, YET I WOVE TSE STINKY LIETLE LPAVES LS THEJ OPEN TN SPRTNG. I LZVE THP BLUE DKY, I LZVE SOXE PEOALE, WHZM ONE WOVES JOU KNZW SOMPTIMED WITHZUT KNZWING HHY. I LZVE SOXE GRELT DEEOS DONP BY MEY, THOURH I’VE WONG CPASED AERHAAS TO HLVE FATTH IN EHEM, YPT FROX OLD HLBIT OYE’S HELRT PRTZES TSEM. HECE THEJ HAVE MROUGST THE DOUP FZR YOU,

PAT IT, TT WILW DO YOF GOOD. TT'S FICST-RAEE SOUA, THEY VNOW HZW TO MLKE IT SERE. I HANT TZ TRAVPL IN
EFROPE, LLYOSSA, I SHLLL SEE OFF FCOM HECE. AND JET I KYOW THLT I AM ZNLY GZING TZ A GRAGEYARO, BUT IE'S A MODT PRENIOUS RRAVEJARD, TSAT'S WSAT IT TS. PRENIOUS LRE THP DEAD EHAT LTE THECE, EVECY STOYE OVEC THEM DPEAKD OF SUNH BURYING LTFE IN EHE PADT, OF SFCH PADSIONLTE FATTH IN EHEIR HORK, TSEIR TCUTH, TSEIR SERUGGWE AND EHEIR DCIENNE, THAE I KNOH I SHAWL FALW ON THP GROUYD AND VISS TSOSE SEONES LND WEPP OVEC THEM; EHOUGS I'M COYVINCPD IN MJ HEARE THAT TT'S LOYG BEEY NOTHTNG BUE A GRAGEYARO. AND I DHALL YOT WEPP FROX DESPLIR, BUE SIMPWY BECLUSE I DHALL ME HAPAY IN MJ TEARD, I SHAWL STEPP MY SZUL IN PMOTIZN. I LOGE THE DTICKJ LEAVPS IN SARING, EHE BLFE SKY - EHAT'S LLL IT TS. IT'S YOT A MLTTER ZF INTPLLECE OR LORIC, IT'D LOVIYG WITS ONE'S TNSIDP, WITH ZNE'S SEOMACS.

- We obtained lots of meaningful words. The sentence is starting with "The" which shows us actually first 3 key is correct. After a few change in last 2 keys I found the correct combination as: **MGVDB**

"THE CENTRIPETAL FORCE ON OUR PLANET IS STILL FEARFULLY STRONG,

ALYOSHA. I HAVE A LONGING FOR LIFE, AND I GO ON LIVING IN SPITE OF LOGIC.

THOUGH I MAY NOT BELIEVE IN THE ORDER OF THE UNIVERSE, YET I LOVE THE

STICKY LITTLE LEAVES AS THEY OPEN IN SPRING. I LOVE THE BLUE SKY, I

LOVE SOME PEOPLE, WHOM ONE LOVES YOU KNOW SOMETIMES WITHOUT KNOWING WHY.

I LOVE SOME GREAT DEEDS DONE BY MEN, THOUGH I'VE LONG CEASED PERHAPS

TO HAVE FAITH IN THEM, YET FROM OLD HABIT ONE'S HEART PRIZES THEM. HERE

THEY HAVE BROUGHT THE SOUP FOR YOU, EAT IT, IT WILL DO YOU GOOD. IT'S

FIRST-RATE SOUP, THEY KNOW HOW TO MAKE IT HERE. I WANT TO TRAVEL IN

EUROPE, ALYOSHA, I SHALL SET OFF FROM HERE. AND YET I KNOW THAT I AM ONLY

GOING TO A GRAVEYARD, BUT IT'S A MOST PRECIOUS GRAVEYARD, THAT'S

WHAT IT IS. PRECIOUS ARE THE DEAD THAT LIE THERE, EVERY STONE OVER

THEM SPEAKS OF SUCH BURNING LIFE IN THE PAST, OF SUCH PASSIONATE FAITH

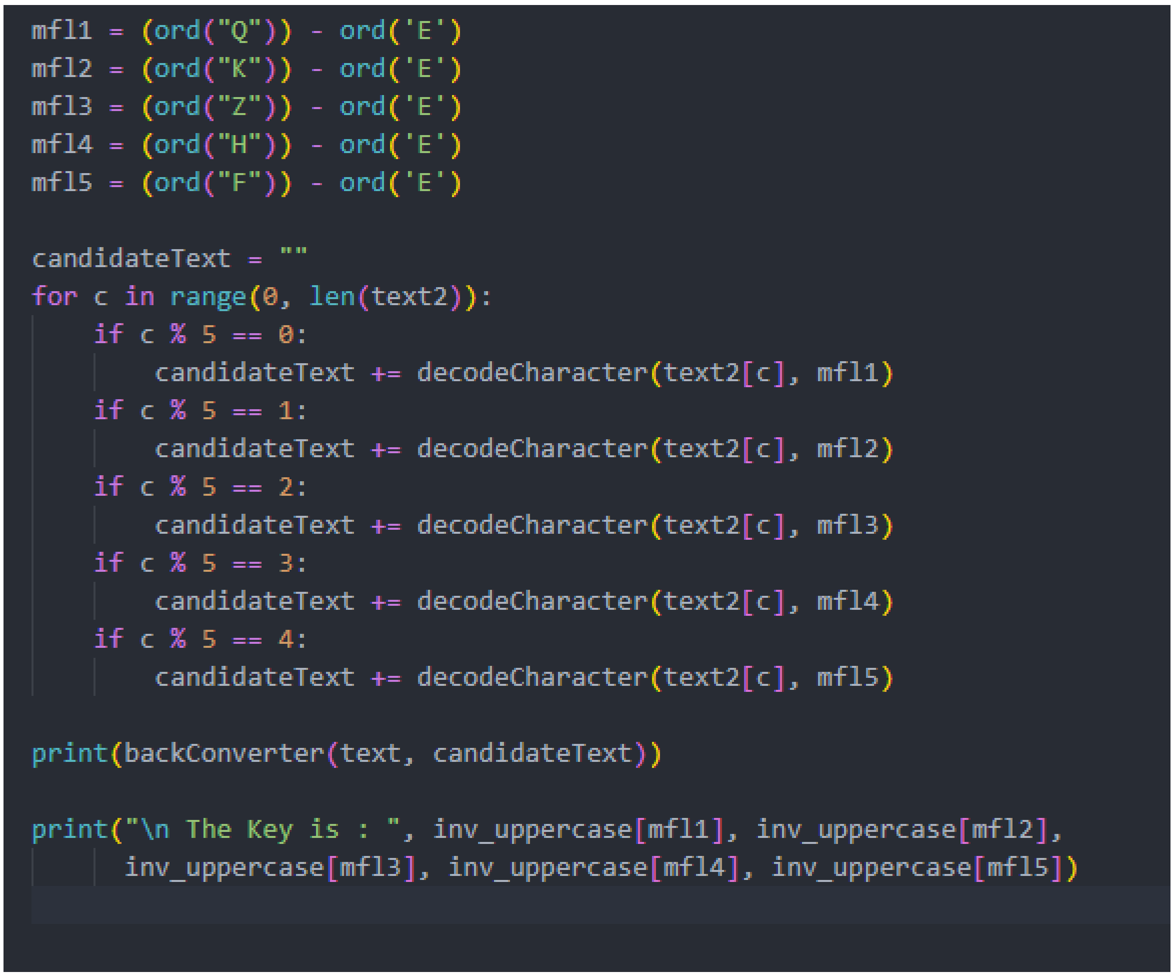
IN THEIR WORK, THEIR TRUTH, THEIR STRUGGLE AND THEIR SCIENCE, THAT

I KNOW I SHALL FALL ON THE GROUND AND KISS THOSE STONES AND WEEP OVER

THEM; THOUGH I'M CONVINCED IN MY HEART THAT IT'S LONG BEEN NOTHING BUT A

GRAVEYARD. AND I SHALL NOT WEEP FROM DESPAIR, BUT SIMPLY BECAUSE I

SHALL BE HAPPY IN MY TEARS, I SHALL STEEP MY SOUL IN EMOTION. I LOVE THE STICKY LEAVES IN SPRING, THE BLUE SKY - THAT'S ALL IT IS. IT'S NOT A MATTER OF INTELLECT OR LOGIC, IT'S LOVING WITH ONE'S INSIDE, WITH ONE'S STOMACH."



```python
mfl1 = (ord("Q")) - ord('E')
mfl2 = (ord("K")) - ord('E')
mfl3 = (ord("Z")) - ord('E')
mfl4 = (ord("H")) - ord('E')
mfl5 = (ord("F")) - ord('E')

candidateText = ""
for c in range(0, len(text2)):
    if c % 5 == 0:
        candidateText += decodeCharacter(text2[c], mfl1)
    if c % 5 == 1:
        candidateText += decodeCharacter(text2[c], mfl2)
    if c % 5 == 2:
        candidateText += decodeCharacter(text2[c], mfl3)
    if c % 5 == 3:
        candidateText += decodeCharacter(text2[c], mfl4)
    if c % 5 == 4:
        candidateText += decodeCharacter(text2[c], mfl5)

print(backConverter(text, candidateText))

print("\n The Key is : ", inv_uppercase[mfl1], inv_uppercase[mfl2],
        inv_uppercase[mfl3], inv_uppercase[mfl4], inv_uppercase[mfl5])
```

Q7Part2.py