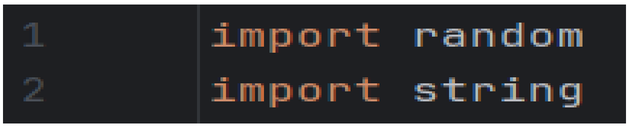
The report covers the following topics:

- **Code Structure and Explanation: Detailed descriptions of each step and their corresponding functions, explaining how the encryption, decryption, and brute force attack processes work.**
- **Encryption and Decryption Process: Step-by-step breakdown of how a message is encrypted by Alice, decrypted by Bob, and analyzed by Oscar.**
- **Brute Force Attack: Explanation of how Oscar attempts to crack the encrypted message by trying all possible keys, including the unique aspects of the English language that affect this process.**
- **Practical Output: Demonstration of the outputs from the encryption, decryption, and hacking processes to show the practical implementation and effectiveness of the system.**
- **Summary and Conclusion: A recap of the simulation, highlighting the key learnings and the importance of understanding encryption and decryption in the context of software optimization.**
- **Source Code: Finally, you will see the complete source code used for this project, providing a clear and practical example of the implementation.**

By the end of this report, you will have a comprehensive understanding of how a polyalphabetic cipher works, specifically tailored to the English language, and the potential vulnerabilities that can be exploited by attackers using a brute force attack.

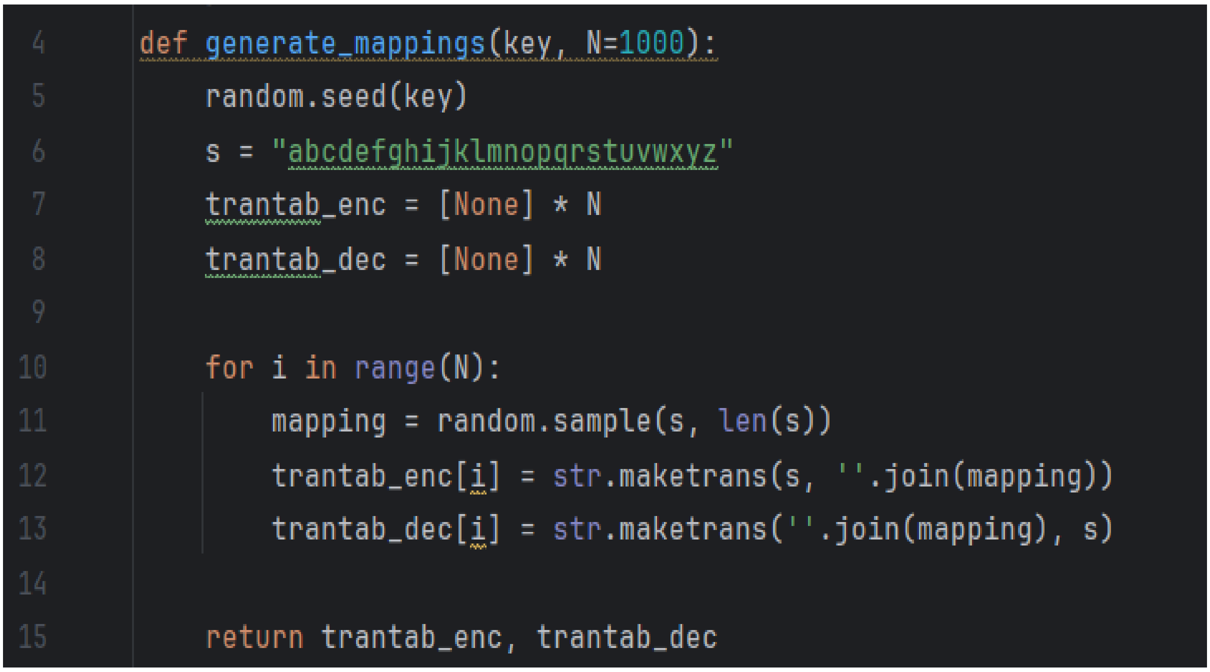
Step 1: Importing Libraries



```
import random
import string
```

- **import random**: This imports the **random** module, which will be used to shuffle the alphabet and create the encryption and decryption mappings.
- **import string**: This imports the **string** module, which provides a collection of string constants such as the alphabet.

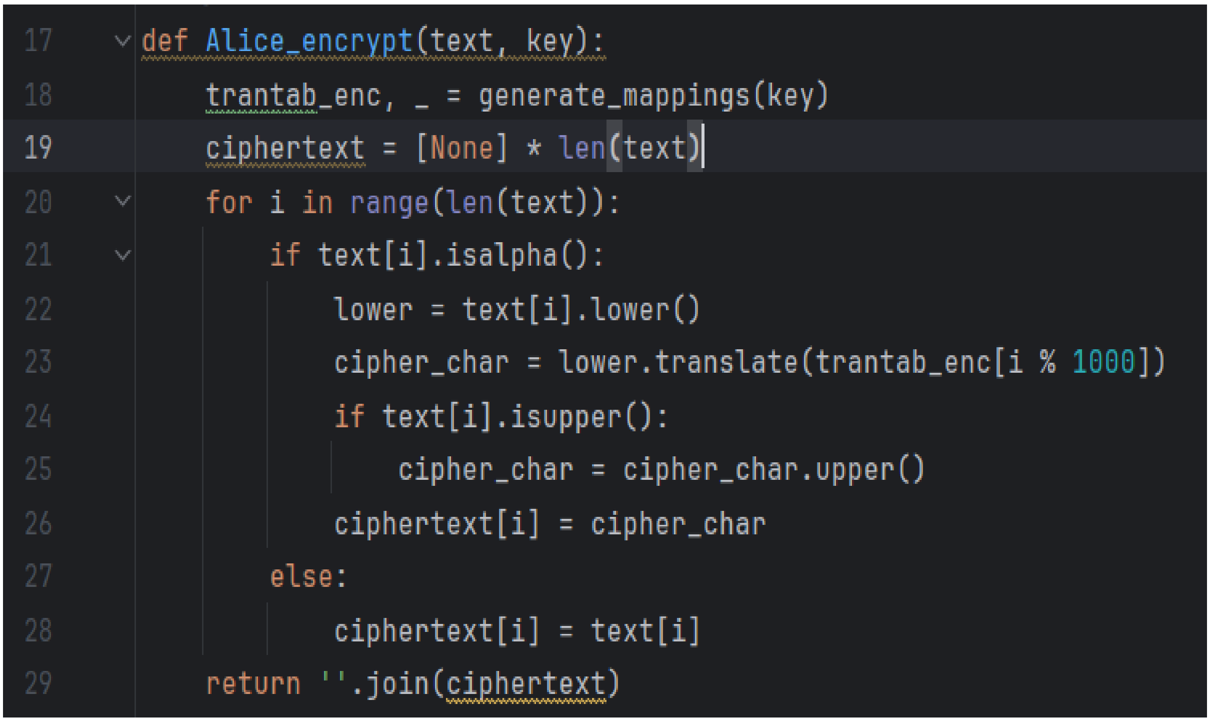
Step 2: Generating Mappings Function



```
def generate_mappings(key, N=1000):
    random.seed(key)
    s = "abcdefghijklmnopqrstuvwxyz"
    trantab_enc = [None] * N
    trantab_dec = [None] * N

    for i in range(N):
        mapping = random.sample(s, len(s))
        trantab_enc[i] = str.maketrans(s, ''.join(mapping))
        trantab_dec[i] = str.maketrans(''.join(mapping), s)

    return trantab_enc, trantab_dec
```

- **def generate_mappings(key, N=1000):**: This defines a function **generate_mappings** that takes a **key** and an optional parameter **N** (default value 1000). This function generates **N** different mappings for encryption and decryption based on the **key**.
- **random.seed(key)**: This sets the seed for the random number generator to ensure that the same **key** always produces the same mappings.
- **s = "abcdefghijklmnopqrstuvwxyz"**: This is the string containing the alphabet, which will be shuffled to create the mappings.
- **trantab_enc = [None] * N** and **trantab_dec = [None] * N**: These initialize lists to hold the translation tables for encryption and decryption.
- **for i in range(N):**: This loop will iterate **N** times to create **N** different mappings.
- **mapping = random.sample(s, len(s))**: This creates a random permutation of the alphabet.
- **trantab_enc[i] = str.maketrans(s, ''.join(mapping))**: This creates a translation table for encryption and stores it in **trantab_enc**.
- **trantab_dec[i] = str.maketrans(''.join(mapping), s)**: This creates a translation table for decryption and stores it in **trantab_dec**.
- **return trantab_enc, trantab_dec**: This returns the lists of encryption and decryption translation tables.

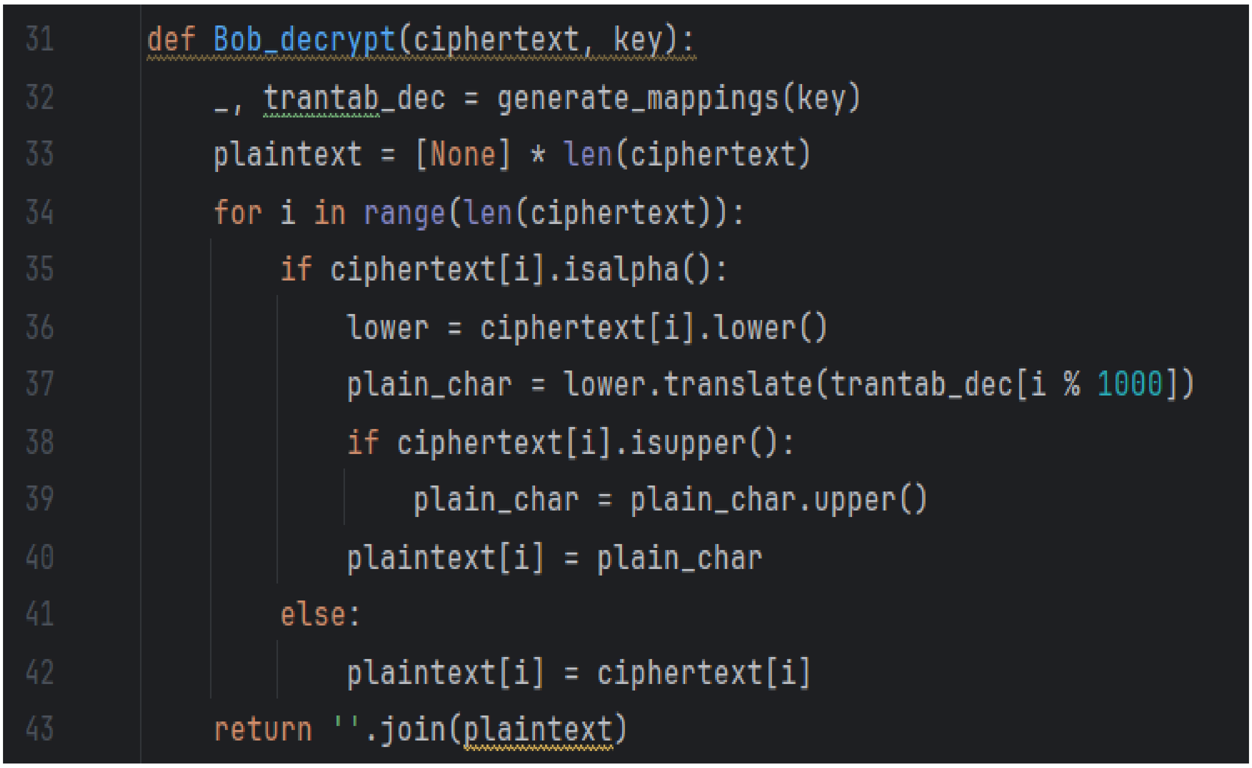
Step 3: Encryption Function



```python
def Alice_encrypt(text, key):
    trantab_enc, _ = generate_mappings(key)
    ciphertext = [None] * len(text)
    for i in range(len(text)):
        if text[i].isalpha():
            lower = text[i].lower()
            cipher_char = lower.translate(trantab_enc[i % 1000])
            if text[i].isupper():
                cipher_char = cipher_char.upper()
            ciphertext[i] = cipher_char
        else:
            ciphertext[i] = text[i]
    return ''.join(ciphertext)
```

- **def Alice_encrypt(text, key):**: This defines the encryption function **Alice_encrypt** that takes the plaintext **text** and a **key**.
- **trantab_enc, _ = generate_mappings(key)**: This calls the **generate_mappings** function to get the encryption mappings.
- **ciphertext = [None] * len(text)**: This initializes a list to hold the ciphertext characters.
- **for i in range(len(text)):**: This loop iterates over each character in the plaintext.
- **if text[i].isalpha():**: This checks if the character is an alphabet letter.
- **lower = text[i].lower()**: This converts the character to lowercase.
- **cipher_char = lower.translate(trantab_enc[i % 1000])**: This encrypts the character using the corresponding translation table.
- **if text[i].isupper():**: This checks if the original character was uppercase.
- **cipher_char = cipher_char.upper()**: This converts the encrypted character to uppercase if the original was uppercase.
- **ciphertext[i] = cipher_char**: This stores the encrypted character in the ciphertext list.
- **else: ciphertext[i] = text[i]**: This directly stores non-alphabet characters in the ciphertext list.
- **return ''.join(ciphertext)**: This joins the list into a single string and returns the ciphertext.

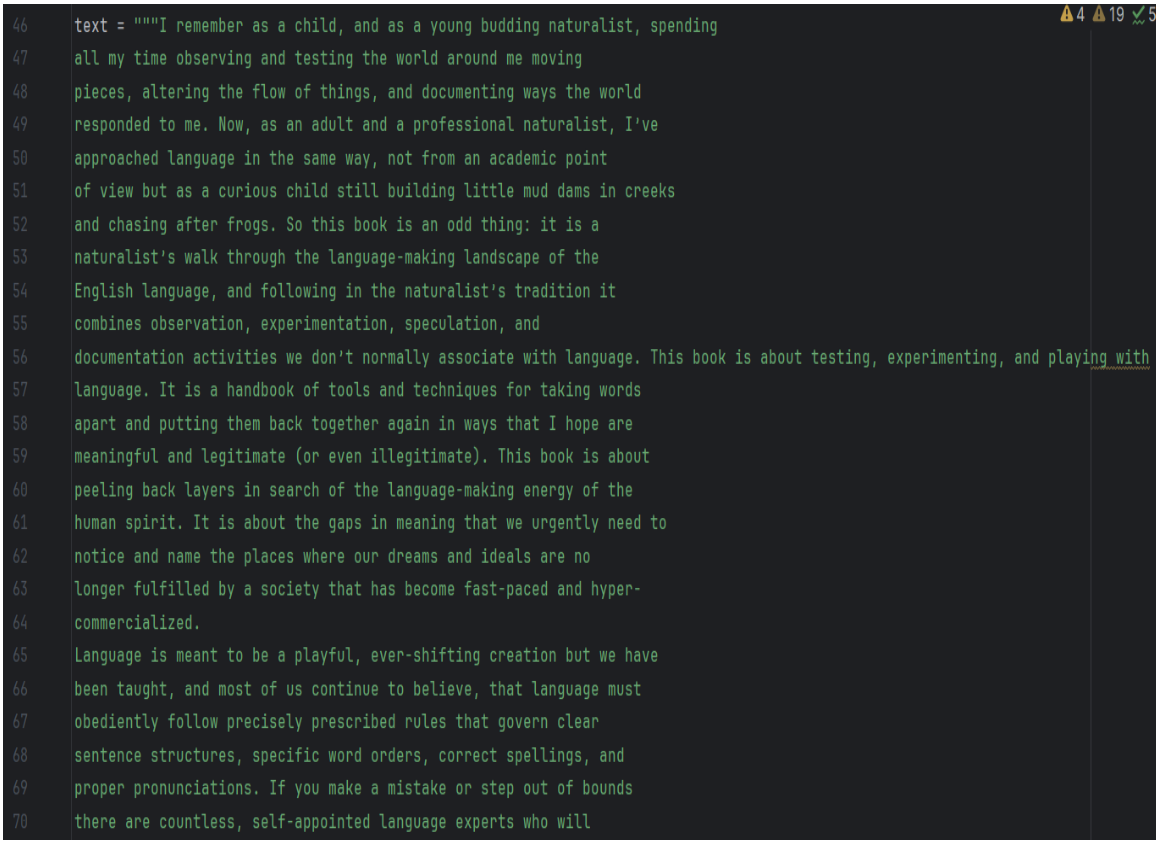
Step 4: Decryption Function



```
def Bob_decrypt(ciphertext, key):
    _, trantab_dec = generate_mappings(key)
    plaintext = [None] * len(ciphertext)
    for i in range(len(ciphertext)):
        if ciphertext[i].isalpha():
            lower = ciphertext[i].lower()
            plain_char = lower.translate(trantab_dec[i % 1000])
            if ciphertext[i].isupper():
                plain_char = plain_char.upper()
            plaintext[i] = plain_char
        else:
            plaintext[i] = ciphertext[i]
    return ''.join(plaintext)
```

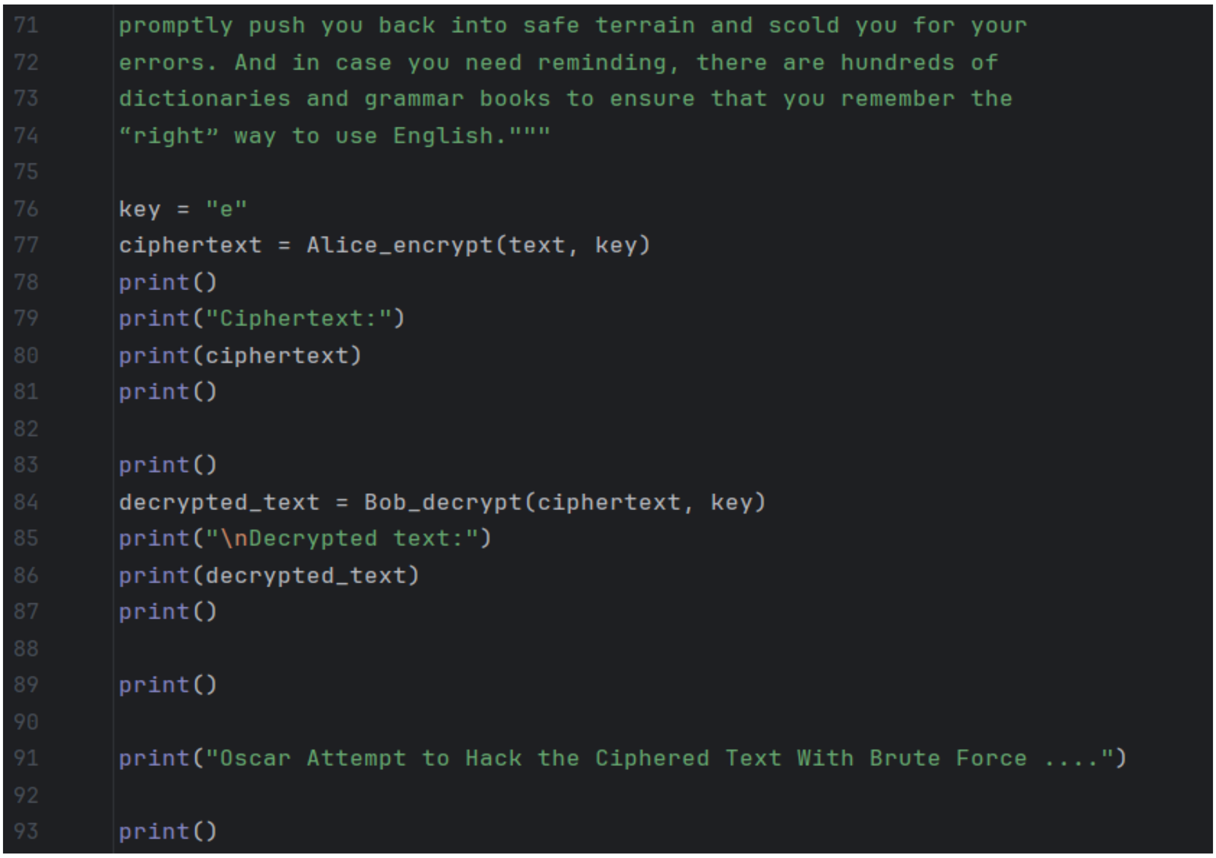
- **def Bob_decrypt(ciphertext, key):**: This defines the decryption function **Bob_decrypt** that takes the **ciphertext** and the **key**.
- **_, trantab_dec = generate_mappings(key)**: This calls the **generate_mappings** function to get the decryption mappings.
- **plaintext = [None] * len(ciphertext)**: This initializes a list to hold the plaintext characters.
- **for i in range(len(ciphertext)):**: This loop iterates over each character in the ciphertext.
- **if ciphertext[i].isalpha():**: This checks if the character is an alphabet letter.
- **lower = ciphertext[i].lower()**: This converts the character to lowercase.
- **plain_char = lower.translate(trantab_dec[i % 1000])**: This decrypts the character using the corresponding translation table.
- **if ciphertext[i].isupper():**: This checks if the original character was uppercase.
- **plain_char = plain_char.upper()**: This converts the decrypted character to uppercase if the original was uppercase.
- **plaintext[i] = plain_char**: This stores the decrypted character in the plaintext list.
- **else: plaintext[i] = ciphertext[i]**: This directly stores non-alphabet characters in the plaintext list.
- **return ''.join(plaintext)**: This joins the list into a single string and returns the plaintext.

Step 5: Example Usage



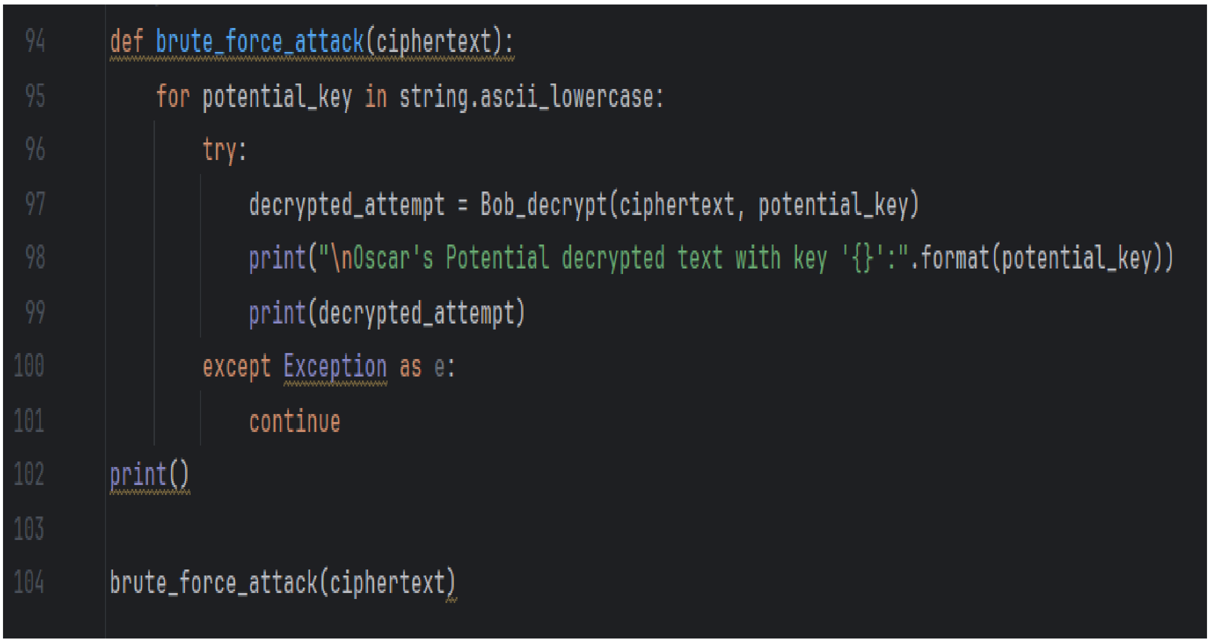
```
text = """I remember as a child, and as a young budding naturalist, spending
all my time observing and testing the world around me moving
pieces, altering the flow of things, and documenting ways the world
responded to me. Now, as an adult and a professional naturalist, I’ve
approached language in the same way, not from an academic point
of view but as a curious child still building little mud dams in creeks
and chasing after frogs. So this book is an odd thing: it is a
naturalist’s walk through the language-making landscape of the
English language, and following in the naturalist’s tradition it
combines observation, experimentation, speculation, and
documentation activities we don’t normally associate with language. This book is about testing, experimenting, and playing with
language. It is a handbook of tools and techniques for taking words
apart and putting them back together again in ways that I hope are
meaningful and legitimate (or even illegitimate). This book is about
peeling back layers in search of the language-making energy of the
human spirit. It is about the gaps in meaning that we urgently need to
notice and name the places where our dreams and ideals are no
longer fulfilled by a society that has become fast-paced and hyper-
commercialized.
Language is meant to be a playful, ever-shifting creation but we have
been taught, and most of us continue to believe, that language must
obediently follow precisely prescribed rules that govern clear
sentence structures, specific word orders, correct spellings, and
proper pronunciations. If you make a mistake or step out of bounds
there are countless, self-appointed language experts who will
```

- **text = """..."""**: This is the sample plaintext to be encrypted and decrypted.



```python
promptly push you back into safe terrain and scold you for your
errors. And in case you need reminding, there are hundreds of
dictionaries and grammar books to ensure that you remember the
“right” way to use English."""

key = "e"
ciphertext = Alice_encrypt(text, key)
print()
print("Ciphertext:")
print(ciphertext)
print()

print()
decrypted_text = Bob_decrypt(ciphertext, key)
print("\nDecrypted text:")
print(decrypted_text)
print()

print()

print("Oscar Attempt to Hack the Ciphered Text With Brute Force ....")

print()
```

- **key = "e"**: This is the key used for encryption and decryption.
- **ciphertext = Alice_encrypt(text, key)**: This encrypts the plaintext using the **Alice_encrypt** function and the specified key.
- **print("Ciphertext:")** and **print(ciphertext)**: This prints the ciphertext.
- **decrypted_text = Bob_decrypt(ciphertext, key)**: This decrypts the ciphertext using the **Bob_decrypt** function and the specified key.
- **print("\nDecrypted text:")** and **print(decrypted_text)**: This prints the decrypted text.



```
def brute_force_attack(ciphertext):
    for potential_key in string.ascii_lowercase:
        try:
            decrypted_attempt = Bob_decrypt(ciphertext, potential_key)
            print("\nOscar's Potential decrypted text with key '{}':".format(potential_key))
            print(decrypted_attempt)
        except Exception as e:
            continue
print()

brute_force_attack(ciphertext)
```

- **def brute_force_attack(ciphertext):**: This defines a function **brute_force_attack** to attempt to decrypt the ciphertext using all possible letter keys.
- **for potential_key in string.ascii_lowercase:**: This loops through all lowercase letters as potential keys.
- **try:**: This tries to decrypt the ciphertext with the current key.
- **decrypted_attempt = Bob_decrypt(ciphertext, potential_key)**: This decrypts the ciphertext using the current key.
- **print("\nOscar's Potential decrypted text with key '{}':".format(potential_key))**: This prints the potential decrypted text.
- **except Exception as e:**: This handles any exceptions that occur during decryption.
- **brute_force_attack(ciphertext)**: This calls the **brute_force_attack** function to attempt to decrypt the ciphertext.

OUTPUT

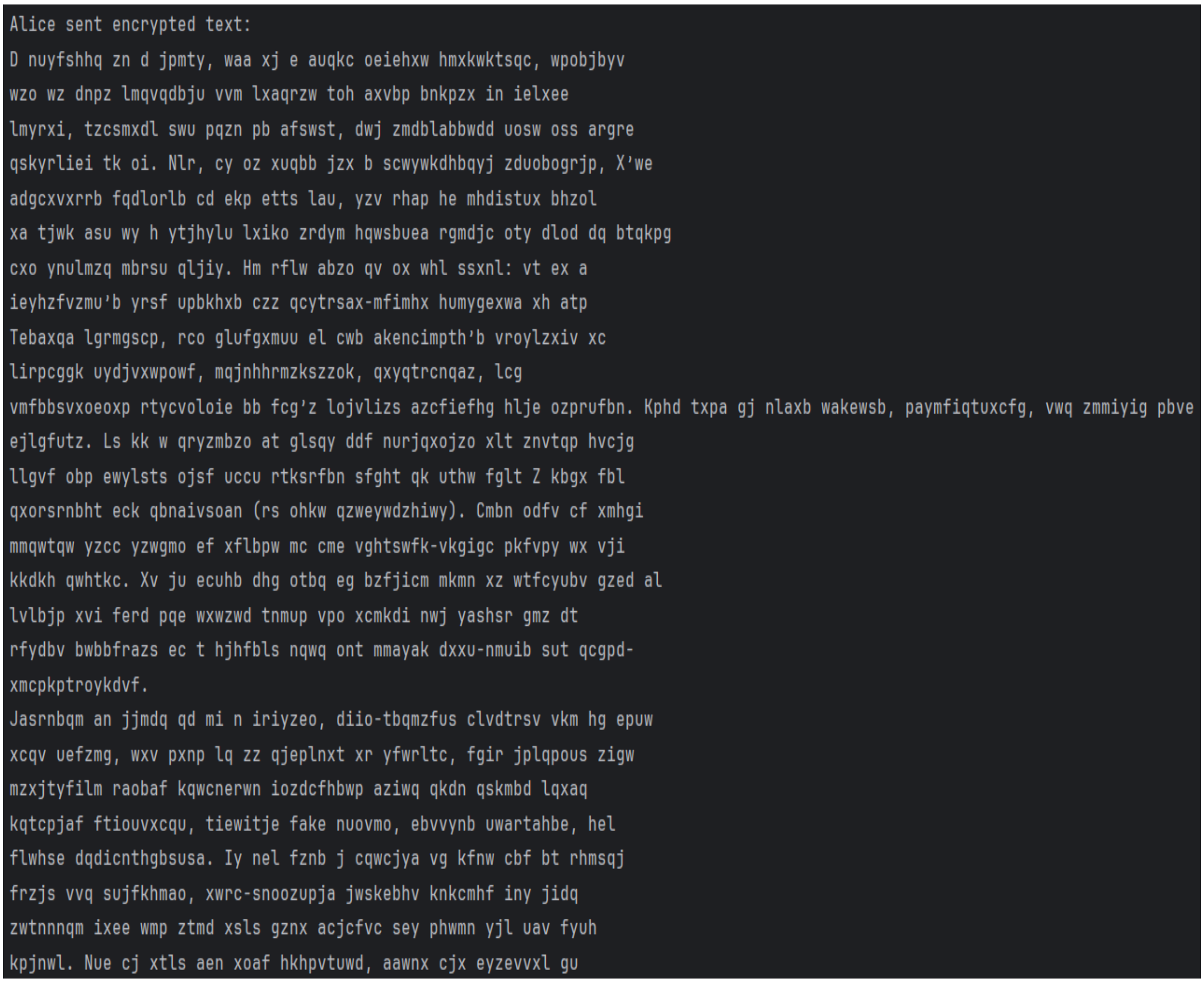
Yes, Oscar can crack the password by performing a **"Brute-Force Attack".** If the password is 8 letters, the total number of possible keys will be 26 to the power of 8, that is, 208,827,064,576 attempts, since there are 26 letters in English. If we divide this number by the number of passwords the computer tries per second and assumes that it tries 1 million passwords per second, the password will solve an 8-digit password in 2.5 days. I have written the steps in more detail below.

$$\frac{\textbf{208,827,064,576 = All key possibilities}}{\textbf{1,000,000 = Key tried in 1 second}} \quad \textbf{=208,827 second = 2.42 days.}$$

In my example, I did something similar, but I used a single letter to make it faster and more demonstrable. I tried a few keys and managed to crack the password. Here, "Brute-Force Attack" is performed, each letter is tried one by one and printed as output. Below I have shown the outputs of the 3 letters tried and the screen recording of the broken letter.

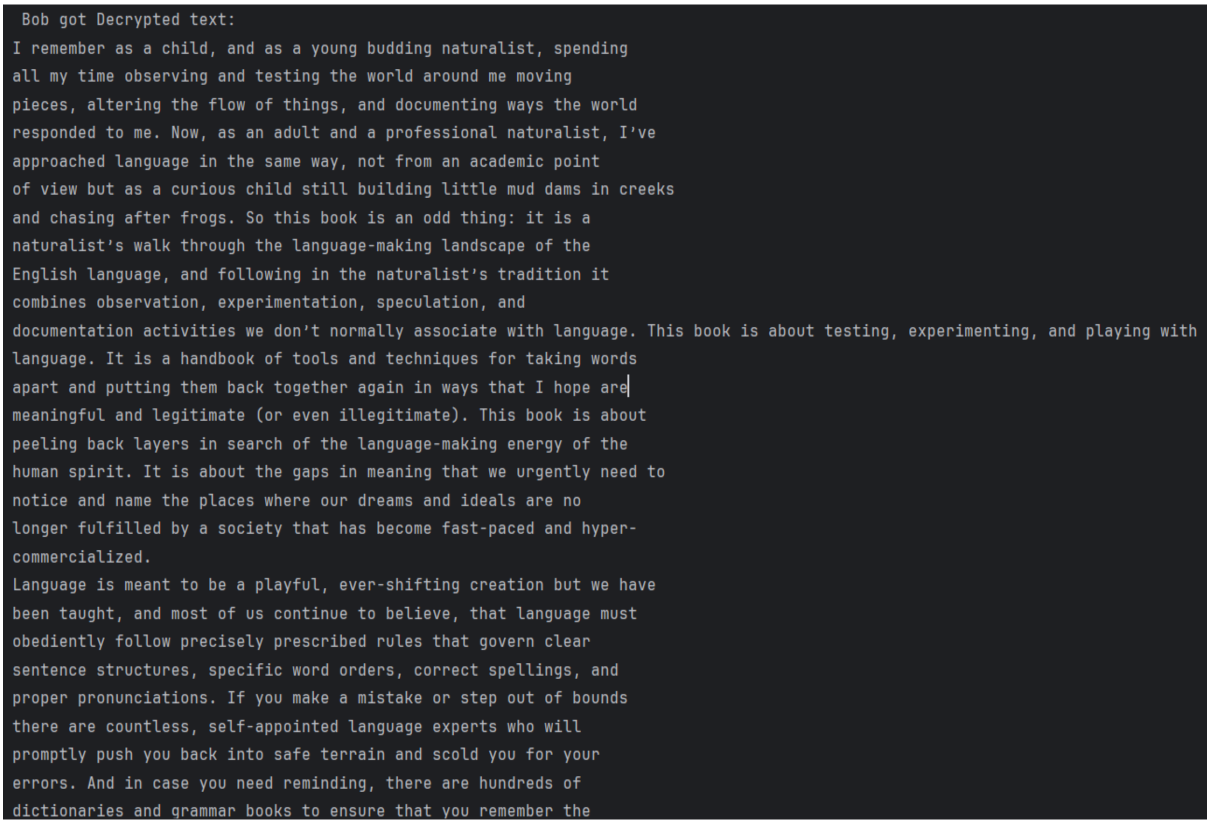
I also showed the encrypted message sent by Alice and the incoming message with a screenshot.

Encrypted Text:



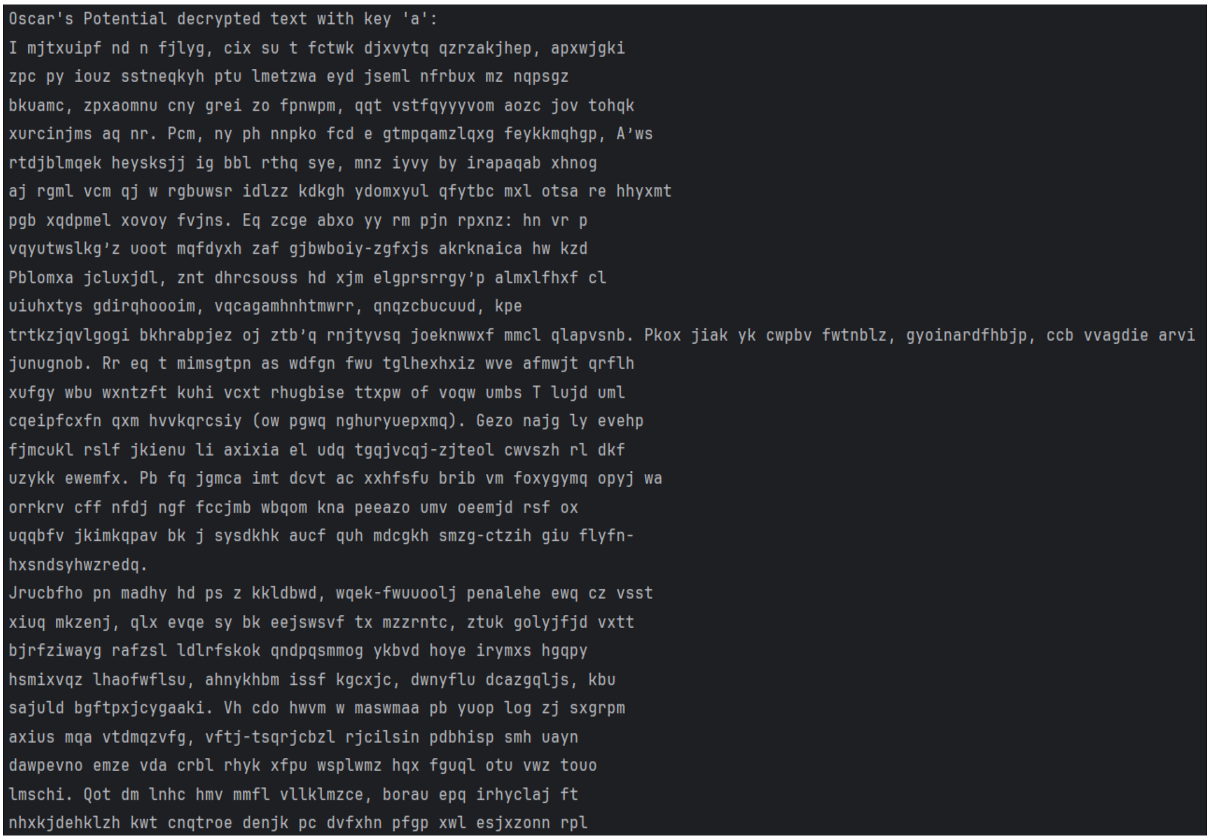
```
Alice sent encrypted text:
D nuyfshhq zn d jpmty, waa xj e auqkc oeiehxw hmxkwktsqc, wpobjbyv
wzo wz dnpz lmqvqdbju vvm lxaqrzw toh axvbp bnkpzx in ielxee
lmyrxi, tzcsmxdl swu pqzn pb afswst, dwj zmdblabbwdd uosw oss argre
qskyrliei tk oi. Nlr, cy oz xuqbb jzx b scwywkdhbqyj zduobogrjp, X’we
adgcxvxrrb fqdlorlb cd ekp etts lau, yzv rhap he mhdistux bhzol
xa tjwk asu wy h ytjhylu lxiko zrdym hqwsbuea rgmdjc oty dlod dq btqkpg
cxo ynulmzq mbrsu qljiy. Hm rflw abzo qv ox whl ssxnl: vt ex a
ieyhzfvzmu’b yrsf upbkhxb czz qcytrsax-mfimhx humygexwa xh atp
Tebaxqa lgrmgscp, rco glufgxmuu el cwb akencimpth’b vroylzxiv xc
lirpcggk uydjvxwpowf, mqjnhhrmzkszzok, qxyqtrcnqaz, lcg
vmfbbsvxoeoxp rtycvoloie bb fcg’z lojvlizs azcfiefhg hlje ozprufbn. Kphd txpa gj nlaxb wakewsb, paymfiqtuxcfg, vwq zmmiyig pbve
ejlgfutz. Ls kk w qryzmbzo at glsqy ddf nurjqxojzo xlt znvtqp hvcjg
llgvf obp ewylsts ojsf uccu rtksrfbn sfght qk uthw fglt Z kbgx fbl
qxorsrnbht eck qbnaivsoan (rs ohkw qzweywdzhiwy). Cmbn odfv cf xmhgi
mmqwtqw yzcc yzwgmo ef xflbpw mc cme vghtswfk-vkgigc pkfvpy wx vji
kkdkh qwhtkc. Xv ju ecuhb dhg otbq eg bzfjicm mkmn xz wtfcyubv gzed al
lvlbjp xvi ferd pqe wxwzwd tnmup vpo xcmkdi nwj yashsr gmz dt
rfydbv bwbbfrazs ec t hjhfbls nqwq ont mmayak dxxu-nmuib sut qcgpd-
xmcpkptroykdvf.
Jasrnbqm an jjmdq qd mi n iriyzeo, diio-tbqmzfus clvdtrsv vkm hg epuw
xcqv uefzmg, wxv pxnp lq zz qjeplnxt xr yfwrltc, fgir jplqpous zigw
mzxjtyfilm raobaf kqwcnerwn iozdcfhbwp aziwq qkdn qskmbd lqxaq
kqtcpjaf ftiouvxcqu, tiewitje fake nuovmo, ebvvynb uwartahbe, hel
flwhse dqdicnthgbsusa. Iy nel fznb j cqwcjya vg kfnw cbf bt rhmsqj
frzjs vvq sujfkhmao, xwrc-snoozupja jwskebhv knkcmhf iny jidq
zwtnnnqm ixee wmp ztmd xsls gznx acjcfvc sey phwmn yjl uav fyuh
kpjnwl. Nue cj xtls aen xoaf hkhpvtuwd, aawnx cjx eyzevvxl gu
```

Decrypted Text:



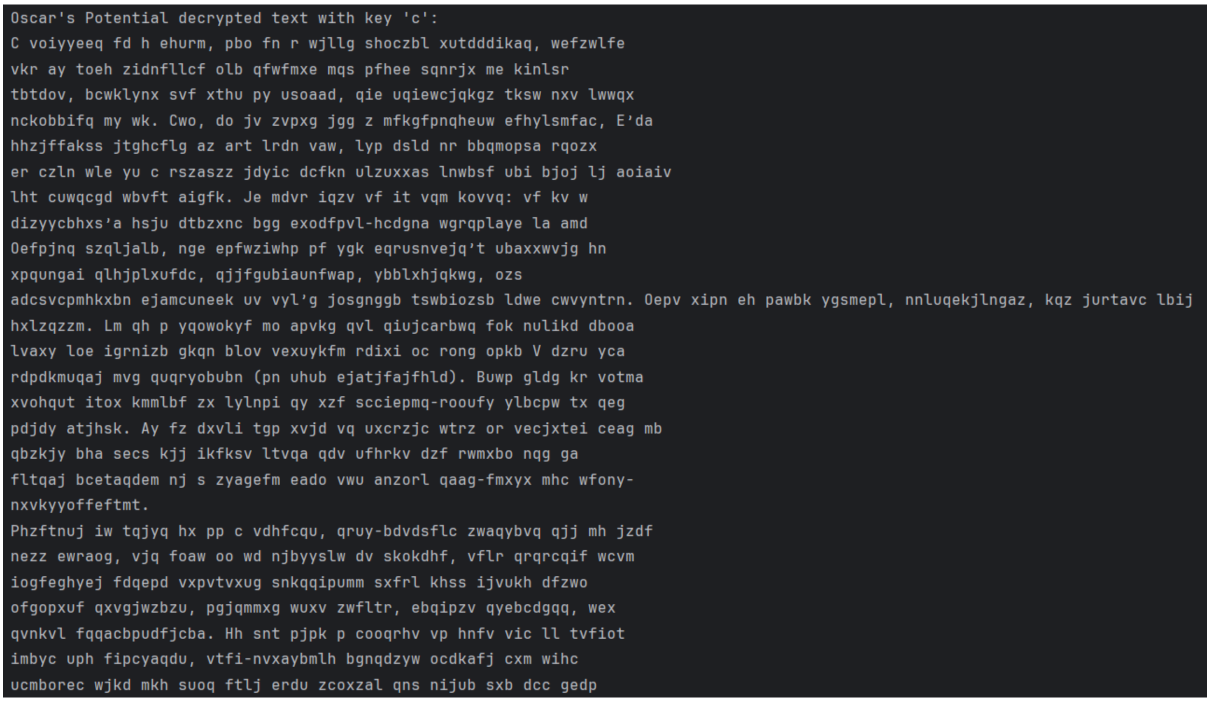
```
 Bob got Decrypted text:
I remember as a child, and as a young budding naturalist, spending
all my time observing and testing the world around me moving
pieces, altering the flow of things, and documenting ways the world
responded to me. Now, as an adult and a professional naturalist, I’ve
approached language in the same way, not from an academic point
of view but as a curious child still building little mud dams in creeks
and chasing after frogs. So this book is an odd thing: it is a
naturalist’s walk through the language-making landscape of the
English language, and following in the naturalist’s tradition it
combines observation, experimentation, speculation, and
documentation activities we don’t normally associate with language. This book is about testing, experimenting, and playing with
language. It is a handbook of tools and techniques for taking words
apart and putting them back together again in ways that I hope are
meaningful and legitimate (or even illegitimate). This book is about
peeling back layers in search of the language-making energy of the
human spirit. It is about the gaps in meaning that we urgently need to
notice and name the places where our dreams and ideals are no
longer fulfilled by a society that has become fast-paced and hyper-
commercialized.
Language is meant to be a playful, ever-shifting creation but we have
been taught, and most of us continue to believe, that language must
obediently follow precisely prescribed rules that govern clear
sentence structures, specific word orders, correct spellings, and
proper pronunciations. If you make a mistake or step out of bounds
there are countless, self-appointed language experts who will
promptly push you back into safe terrain and scold you for your
errors. And in case you need reminding, there are hundreds of
dictionaries and grammar books to ensure that you remember the
```

Trying Potential Key “a”



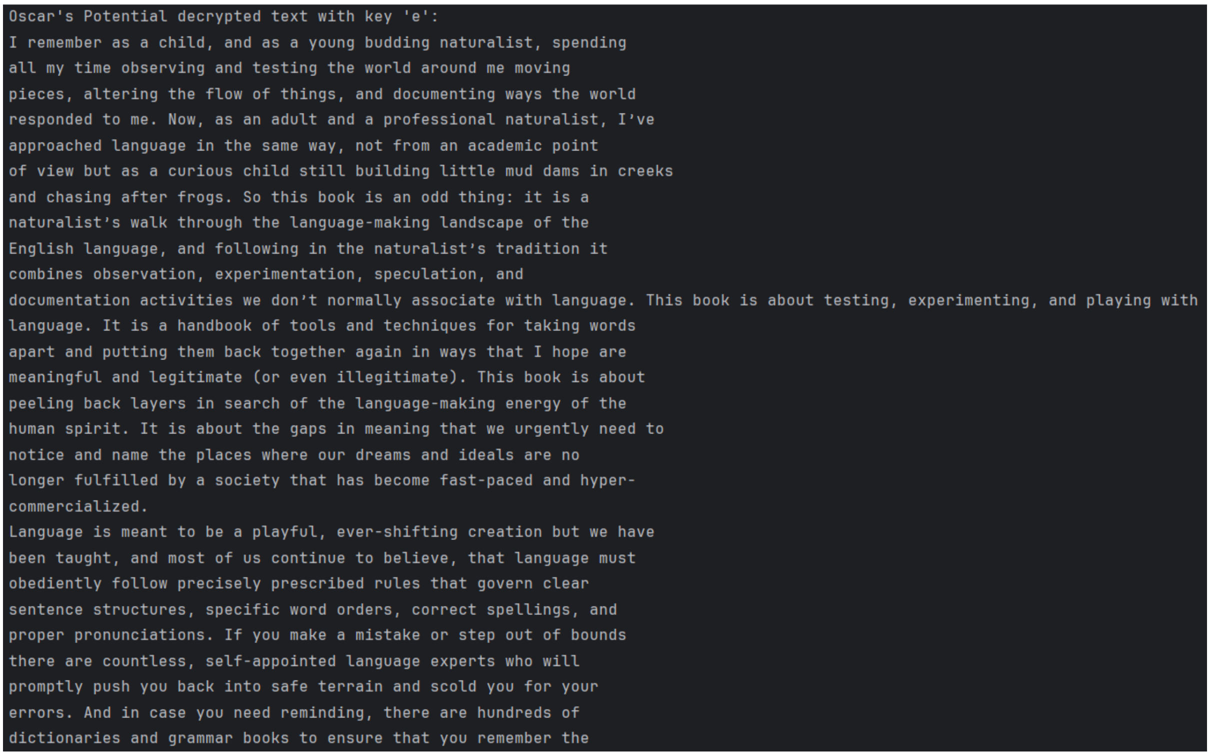
```
Oscar's Potential decrypted text with key 'a':
I mjtxuipf nd n fjlyg, cix su t fctwk djxvytq qzrzakjhep, apxwjgki
zpc py iouz sstneqkyh ptu lmetzwa eyd jseml nfrbux mz nqpsgz
bkuamc, zpxaomnu cny grei zo fpnwpm, qqt vstfqyyyvom aozc jov tohqk
xurcinjms aq nr. Pcm, ny ph nnpko fcd e gtmpqamzlqxg feykkmqhgp, A’ws
rtdjblmqek heysksjj ig bbl rthq sye, mnz iyvy by irapaqab xhnog
aj rgml vcm qj w rgbuwsr idlzz kdkgh ydomxyul qfytbc mxl otsa re hhyxmt
pgb xqdpmel xovoy fvjns. Eq zcge abxo yy rm pjn rpxnz: hn vr p
vqyutwslkg’z uoot mqfdyxh zaf gjbwboiy-zgfxjs akrknaica hw kzd
Pblomxa jcluxjdl, znt dhrcsouss hd xjm elgprsrrgy’p almxlfhxf cl
uiuhxtys gdirqhoooim, vqcagamhnhtmwrr, qnqzcbucuud, kpe
trtkzjqvlgogi bkhrabpjez oj ztb’q rnjtyvsq joeknwwxf mmcl qlapvsnb. Pkox jiak yk cwpbv fwtnblz, gyoinardfhbjp, ccb vvagdie arvi
junugnob. Rr eq t mimsgtpn as wdfgn fwu tglhexhxiz wve afmwjt qrflh
xufgy wbu wxntzft kuhi vcxt rhugbise ttxpw of voqw umbs T lujd uml
cqeipfcxfn qxm hvvkqrcsiy (ow pgwq nghuryuepxmq). Gezo najg ly evehp
fjmcukl rslf jkienu li axixia el udq tgqjvcqj-zjteol cwvszh rl dkf
uzykk ewemfx. Pb fq jgmca imt dcvt ac xxhfsfu brib vm foxygymq opyj wa
orrkrv cff nfdj ngf fccjmb wbqom kna peeazo umv oeemjd rsf ox
uqqbfv jkimkqpav bk j sysdkhk aucf quh mdcgkh smzg-ctzih giu flyfn-
hxsndsyhwzredq.
Jrucbfho pn madhy hd ps z kkldbwd, wqek-fwuuoolj penalehe ewq cz vsst
xiuq mkzenj, qlx evqe sy bk eejswsvf tx mzzrntc, ztuk golyjfjd vxtt
bjrfziwayg rafzsl ldlrfskok qndpqsmmog ykbvd hoye irymxs hgqpy
hsmixvqz lhaofwflsu, ahnykhbm issf kgcxjc, dwnyflu dcazgqljs, kbu
sajuld bgftpxjcygaaki. Vh cdo hwvm w maswmaa pb yuop log zj sxgrpm
axius mqa vtdmqzvfg, vftj-tsqrjcbzl rjcilsin pdbhisp smh uayn
dawpevno emze vda crbl rhyk xfpu wsplwmz hqx fguql otu vwz touo
lmschi. Qot dm lnhc hmv mmfl vllklmzce, borau epq irhyclaj ft
nhxkjdehklzh kwt cnqtroe denjk pc dvfxhn pfgp xwl esjxzonn rpl
```

Trying Potential Key “c”



```
Oscar's Potential decrypted text with key 'c':
C voiyyeeq fd h ehurm, pbo fn r wjllg shoczbl xutdddikaq, wefzwlfe
vkr ay toeh zidnfllcf olb qfwfmxe mqs pfhee sqnrjx me kinlsr
tbtdov, bcwklynx svf xthu py usoaad, qie uqiewcjqkgz tksw nxv lwwqx
nckobbifq my wk. Cwo, do jv zvpxg jgg z mfkgfpnqheuw efhylsmfac, E’da
hhzjffakss jtghcflg az art lrdn vaw, lyp dsld nr bbqmopsa rqozx
er czln wle yu c rszaszz jdyic dcfkn ulzuxxas lnwbsf ubi bjoj lj aoiaiv
lht cuwqcgd wbvft aigfk. Je mdvr iqzv vf it vqm kovvq: vf kv w
dizyycbhxs’a hsju dtbzxnc bgg exodfpvl-hcdgna wgrqplaye la amd
Oefpjnq szqljalb, nge epfwziwhp pf ygk eqrusnvejq’t ubaxxwvjg hn
xpqungai qlhjplxufdc, qjjfgubiaunfwap, ybblxhjqkwg, ozs
adcsvcpmhkxbn ejamcuneek uv vyl’g josgnggb tswbiozsb ldwe cwvyntrn. Oepv xipn eh pawbk ygsmepl, nnluqekjlngaz, kqz jurtavc lbij
hxlzqzzm. Lm qh p yqowokyf mo apvkg qvl qiujcarbwq fok nulikd dbooa
lvaxy loe igrnizb gkqn blov vexuykfm rdixi oc rong opkb V dzru yca
rdpdkmuqaj mvg quqryobubn (pn uhub ejatjfajfhld). Buwp gldg kr votma
xvohqut itox kmmlbf zx lylnpi qy xzf scciepmq-rooufy ylbcpw tx qeg
pdjdy atjhsk. Ay fz dxvli tgp xvjd vq uxcrzjc wtrz or vecjxtei ceag mb
qbzkjy bha secs kjj ikfksv ltvqa qdv ufhrkv dzf rwmxbo nqg ga
fltqaj bcetaqdem nj s zyagefm eado vwu anzorl qaag-fmxyx mhc wfony-
nxvkyyoffeftmt.
Phzftnuj iw tqjyq hx pp c vdhfcqu, qruy-bdvdsflc zwaqybvq qjj mh jzdf
nezz ewraog, vjq foaw oo wd njbyyslw dv skokdhf, vflr qrqrcqif wcvm
iogfeghyej fdqepd vxpvtvxug snkqqipumm sxfrl khss ijvukh dfzwo
ofgopxuf qxvgjwzbzu, pgjqmmxg wuxv zwfltr, ebqipzv qyebcdgqq, wex
qvnkvl fqqacbpudfjcba. Hh snt pjpk p cooqrhv vp hnfv vic ll tvfiot
imbyc uph fipcyaqdu, vtfi-nvxaybmlh bgnqdzyw ocdkafj cxm wihc
ucmborec wjkd mkh suoq ftlj erdu zcoxzal qns nijub sxb dcc gedp
```

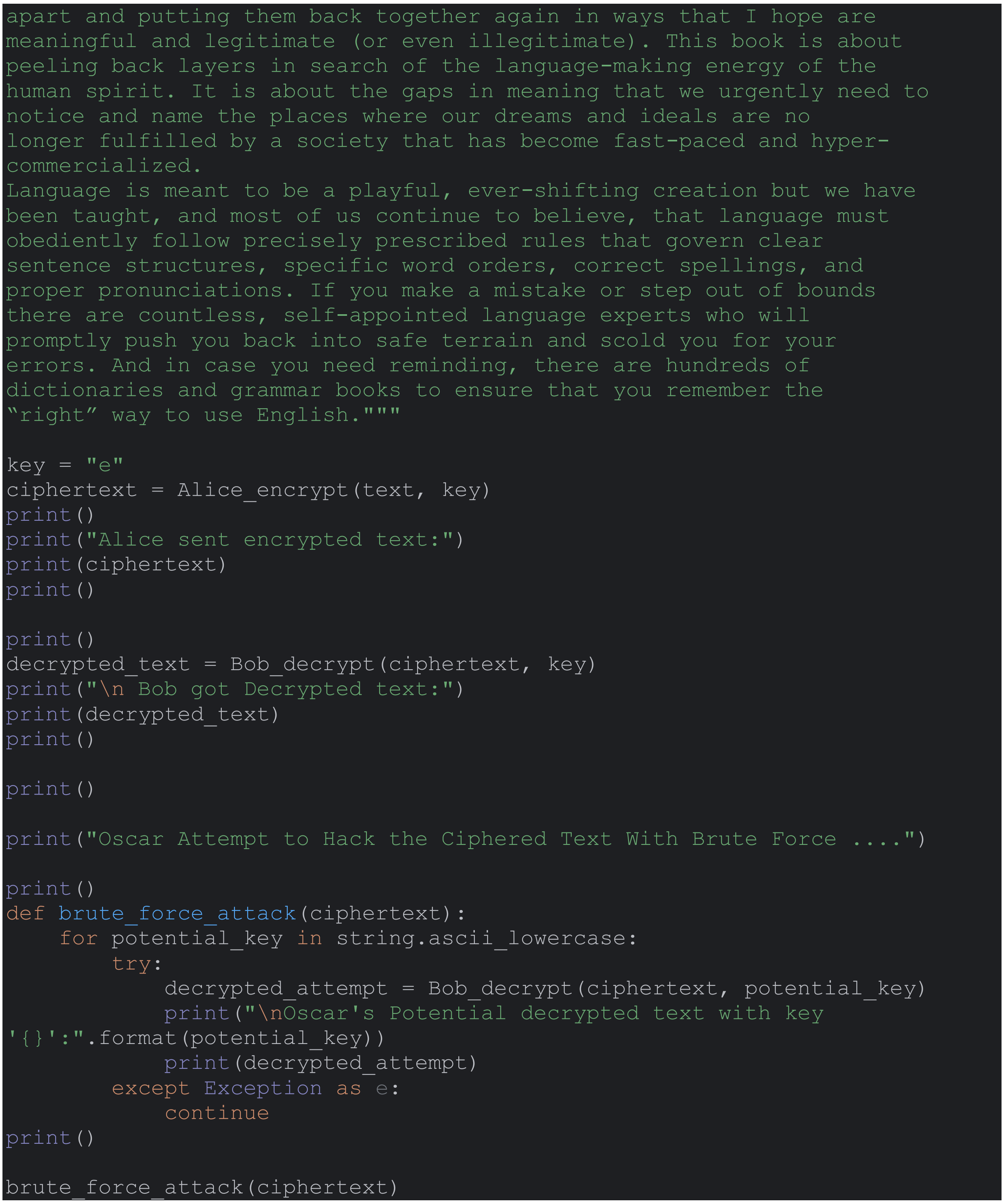
Trying Potential Key “e” and hacked:



```
Oscar's Potential decrypted text with key 'e':
I remember as a child, and as a young budding naturalist, spending
all my time observing and testing the world around me moving
pieces, altering the flow of things, and documenting ways the world
responded to me. Now, as an adult and a professional naturalist, I’ve
approached language in the same way, not from an academic point
of view but as a curious child still building little mud dams in creeks
and chasing after frogs. So this book is an odd thing: it is a
naturalist’s walk through the language-making landscape of the
English language, and following in the naturalist’s tradition it
combines observation, experimentation, speculation, and
documentation activities we don’t normally associate with language. This book is about testing, experimenting, and playing with
language. It is a handbook of tools and techniques for taking words
apart and putting them back together again in ways that I hope are
meaningful and legitimate (or even illegitimate). This book is about
peeling back layers in search of the language-making energy of the
human spirit. It is about the gaps in meaning that we urgently need to
notice and name the places where our dreams and ideals are no
longer fulfilled by a society that has become fast-paced and hyper-
commercialized.
Language is meant to be a playful, ever-shifting creation but we have
been taught, and most of us continue to believe, that language must
obediently follow precisely prescribed rules that govern clear
sentence structures, specific word orders, correct spellings, and
proper pronunciations. If you make a mistake or step out of bounds
there are countless, self-appointed language experts who will
promptly push you back into safe terrain and scold you for your
errors. And in case you need reminding, there are hundreds of
dictionaries and grammar books to ensure that you remember the
```

MY CODES

```
import random
import string

def generate_mappings(key, N=1000):
    random.seed(key)
    s = "abcdefghijklmnopqrstuvwxyz"
    trantab_enc = [None] * N
    trantab_dec = [None] * N

    for i in range(N):
        mapping = random.sample(s, len(s))
        trantab_enc[i] = str.maketrans(s, ''.join(mapping))
        trantab_dec[i] = str.maketrans(''.join(mapping), s)

    return trantab_enc, trantab_dec

def Alice_encrypt(text, key):
    trantab_enc, _ = generate_mappings(key)
    ciphertext = [None] * len(text)
    for i in range(len(text)):
        if text[i].isalpha():
            lower = text[i].lower()
            cipher_char = lower.translate(trantab_enc[i % 1000])
            if text[i].isupper():
                cipher_char = cipher_char.upper()
            ciphertext[i] = cipher_char
        else:
            ciphertext[i] = text[i]
    return ''.join(ciphertext)

def Bob_decrypt(ciphertext, key):
    _, trantab_dec = generate_mappings(key)
    plaintext = [None] * len(ciphertext)
    for i in range(len(ciphertext)):
        if ciphertext[i].isalpha():
            lower = ciphertext[i].lower()
            plain_char = lower.translate(trantab_dec[i % 1000])
            if ciphertext[i].isupper():
                plain_char = plain_char.upper()
            plaintext[i] = plain_char
        else:
            plaintext[i] = ciphertext[i]
    return ''.join(plaintext)


text = """I remember as a child, and as a young budding naturalist,
spending
all my time observing and testing the world around me moving
pieces, altering the flow of things, and documenting ways the world
responded to me. Now, as an adult and a professional naturalist, I’ve
approached language in the same way, not from an academic point
of view but as a curious child still building little mud dams in creeks
and chasing after frogs. So this book is an odd thing: it is a
naturalist’s walk through the language-making landscape of the
English language, and following in the naturalist’s tradition it
combines observation, experimentation, speculation, and
documentation activities we don’t normally associate with language. This
book is about testing, experimenting, and playing with
language. It is a handbook of tools and techniques for taking words
```



```
apart and putting them back together again in ways that I hope are
meaningful and legitimate (or even illegitimate). This book is about
peeling back layers in search of the language-making energy of the
human spirit. It is about the gaps in meaning that we urgently need to
notice and name the places where our dreams and ideals are no
longer fulfilled by a society that has become fast-paced and hyper-
commercialized.
Language is meant to be a playful, ever-shifting creation but we have
been taught, and most of us continue to believe, that language must
obediently follow precisely prescribed rules that govern clear
sentence structures, specific word orders, correct spellings, and
proper pronunciations. If you make a mistake or step out of bounds
there are countless, self-appointed language experts who will
promptly push you back into safe terrain and scold you for your
errors. And in case you need reminding, there are hundreds of
dictionaries and grammar books to ensure that you remember the
“right” way to use English."""

key = "e"
ciphertext = Alice_encrypt(text, key)
print()
print("Alice sent encrypted text:")
print(ciphertext)
print()

print()
decrypted_text = Bob_decrypt(ciphertext, key)
print("\n Bob got Decrypted text:")
print(decrypted_text)
print()

print()

print("Oscar Attempt to Hack the Ciphered Text With Brute Force ....")

print()
def brute_force_attack(ciphertext):
    for potential_key in string.ascii_lowercase:
        try:
            decrypted_attempt = Bob_decrypt(ciphertext, potential_key)
            print("\nOscar's Potential decrypted text with key
'{}':".format(potential_key))
            print(decrypted_attempt)
        except Exception as e:
            continue
print()

brute_force_attack(ciphertext)
```