Drew Brown

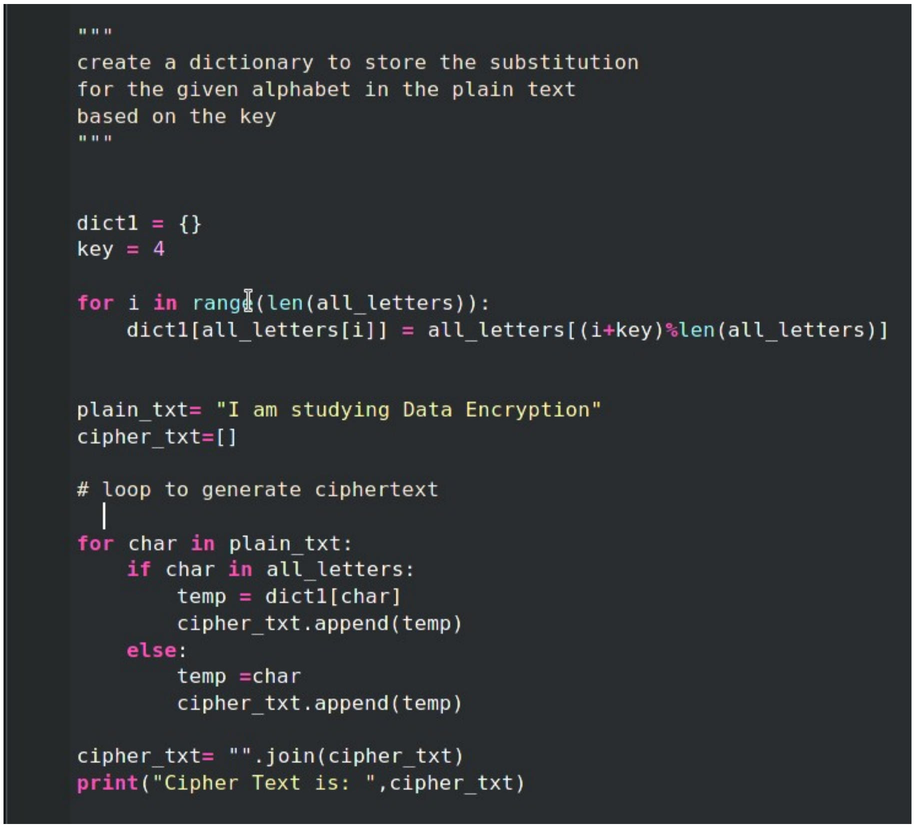
CYSE600 – Cybersecurity Principles HW#2 – Substitution Cipher

Dr. Hongyi Wu 07Feb2022

The ciphertext provided for analysis:

```
LWNSOZBNWVWBAYBNVBSQWVUOHWDIZWRBBNPBPOOUWRPAWXAWPBWZWMYPOBNPBBN
WJPAWWRZSLWZQJBNVIAXAWPBSALIBNXWABPIRYRPOIWRPQOWAIENBVBNPBPUSREBNWVW
PAWOIHWOIQWABJPRZBNWFYAVYIBSHNPFFIRWVVBNPBBSVWXYAWBNWVWAIENBVESDWAR
UWRBVPAWIRVBIBYBWZPUSREUWRZWAIDIREBHWIATYVBFSLWAVHASUBNWXSRVWRBSHBOT
ESDWARWZBNPBLNWWDWAPRJHSAUSHESDWARUWRBQWXSUWVZWVBAYXBIDWSHBNWVW
WRZVIBIVBNVAIENBSHBNWFWSFOWBSPOBWASABSPQSOIVNIBPRZBSIRVBIBYBWRWLESDWA
RUWRBOPJIREIBVHSYRZPBISRSRVYXNFAIRXIFOOTPRZSAEPRIKIREIBVFSLWAVIRVYXNHSAUPV
BSVWMJSVBOICWOJBSWHHWXBBNWIAVPHWBJPRZNPFFIRWW
```

My first step of analysis for this ciphertext was to research and learn more about substitution ciphers in general. This website [1] provided an overview and a scripting framework for me to work with.

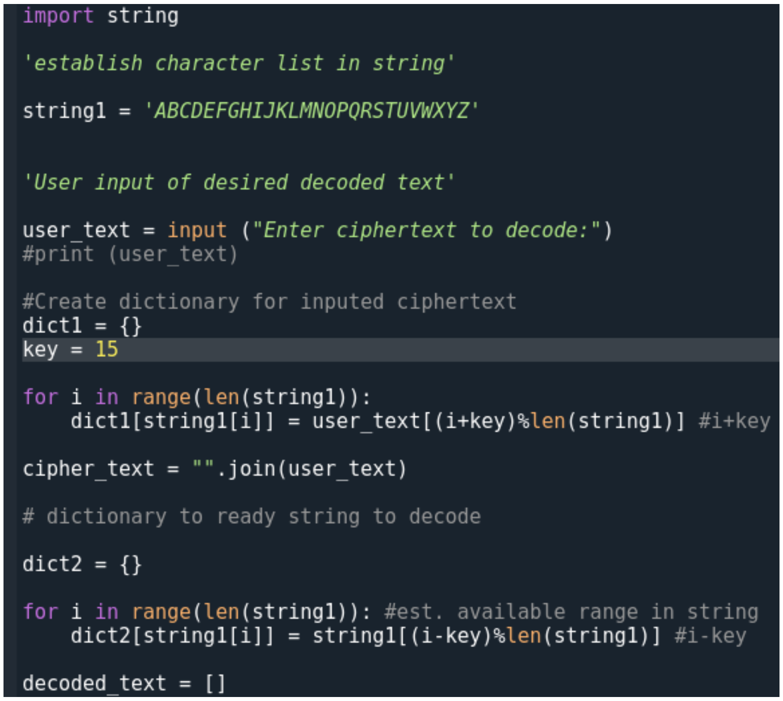


```python
"""
create a dictionary to store the substitution
for the given alphabet in the plain text
based on the key
"""


dict1 = {}
key = 4

for i in range(len(all_letters)):
    dict1[all_letters[i]] = all_letters[(i+key)%len(all_letters)]


plain_txt= "I am studying Data Encryption"
cipher_txt=[]

# loop to generate ciphertext

for char in plain_txt:
    if char in all_letters:
        temp = dict1[char]
        cipher_txt.append(temp)
    else:
        temp =char
        cipher_txt.append(temp)

cipher_txt= "".join(cipher_txt)
print("Cipher Text is: ",cipher_txt)
```

Figure 1: Excerpt of geeksforgeeks.org webpage authors code. This is an example script which takes plaintext and encodes to ciphertext then decodes back to plaintext. Unhelpful for my task.

This script example did allow me to expose myself to formatting and syntax of python. Further analysis revealed this is more inline to a shift cipher shifting characters according to the chosen key = 4.

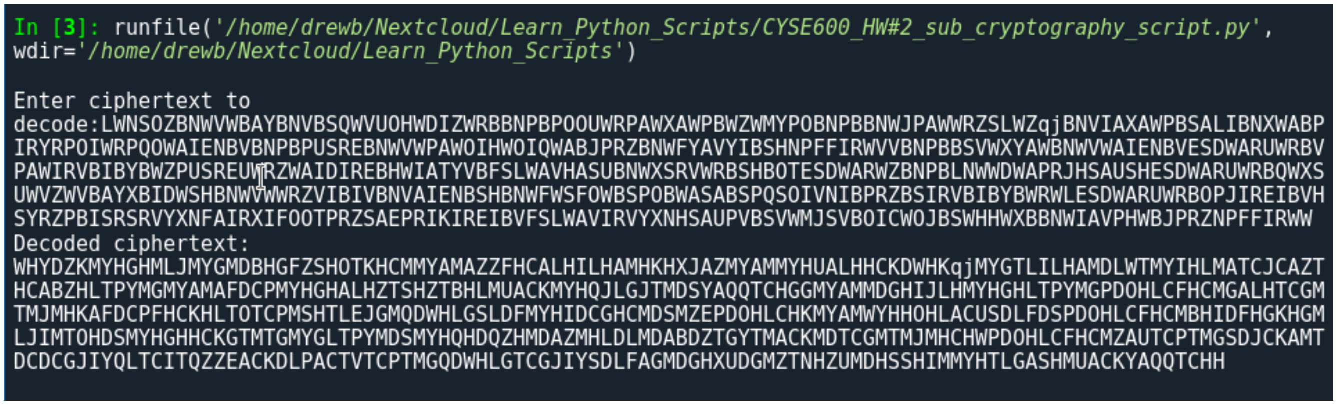
I attempted to adapt the above code for my use case to allow for user input of cipher text and then manipulate ciphertext according to a plaintext English character dictionary.



```
import string

'establish character list in string'

string1 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'


'User input of desired decoded text'

user_text = input ("Enter ciphertext to decode:")
#print (user_text)

#Create dictionary for inputed ciphertext
dict1 = {}
key = 15

for i in range(len(string1)):
    dict1[string1[i]] = user_text[(i+key)%len(string1)] #i+key

cipher_text = "".join(user_text)

# dictionary to ready string to decode

dict2 = {}

for i in range(len(string1)): #est. available range in string
    dict2[string1[i]] = string1[(i-key)%len(string1)] #i-key

decoded_text = []
```

Figure 2: Python script adaptation to allow user input of ciphertext then manipulate inputs according to chosen key = 15.

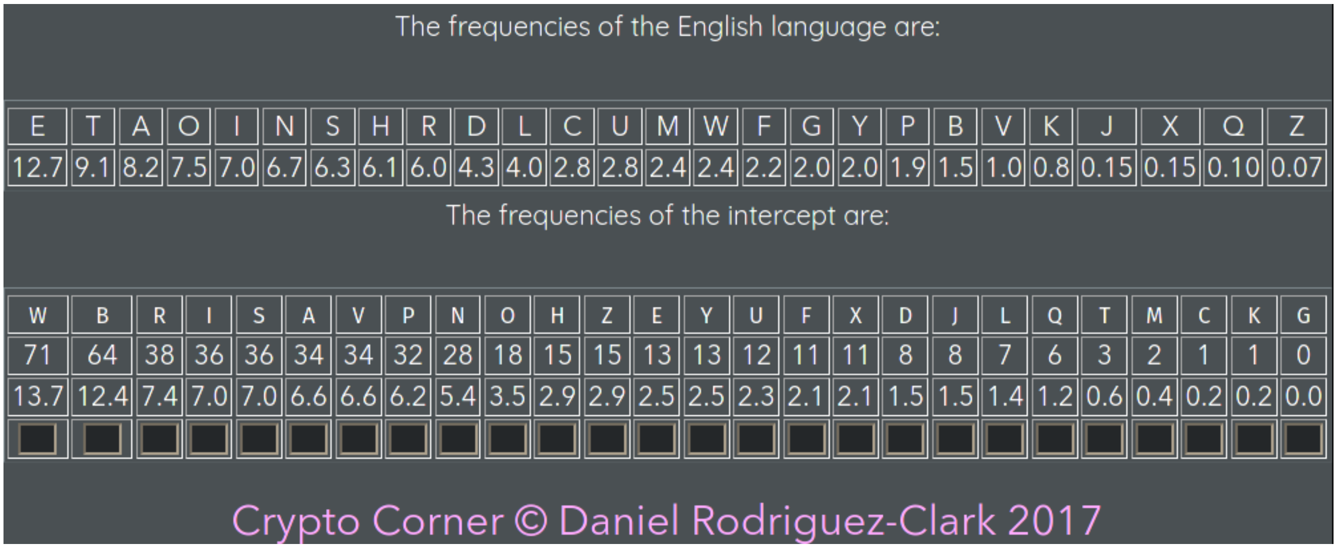
Recalling the dictionary length of 26 characters, I continued to input the ciphertext (Fig. 3) to watch for a human readable decoded plaintext.



```
In [3]: runfile('/home/drewb/Nextcloud/Learn_Python_Scripts/CYSE600_HW#2_sub_cryptography_script.py',
wdir='/home/drewb/Nextcloud/Learn_Python_Scripts')

Enter ciphertext to
decode:LWNSOZBNWVWBAYBNVBSQWVUOHWDIZWRBBNPBPOOUWRPAWXAWPBWZWMYPOBNPBBNWJPAWWRZSLWZqjBNVIAXAWPBSALIBNXWABP
IRYRPOIWRPQOWAIENBVBNPBPUSREBNWVWPAWOIHWOIQWABJPRZBNWFYAVYIBSHNPFFIRWVVBNPBBSVWXYAWBNWVWAIENBVESDWARUWRBV
PAWIRVBIBYBWZPUSREUWRZWAIDIREBHWIATYVBFSLWAVHASUBNWXSRVWRBSHBOTESDWARWZBNPBLNWWDWAPRJHSAUSHESDWARUWRBQWXS
UWVZWVBAYXBIDWSHBNWVWWRZVIBIVBNVAIENBSHBNWFWSFOWBSPOBWASABSPQSOIVNIBPRZBSIRVBIBYBWRWLESDWARUWRBOPJIREIBVH
SYRZPBISRSRVYXNFAIRXIFOOTPRZSAEPRIKIREIBVFSLWAVIRVYXNHSAUPVBSVWMJSVBOICWOJBSWHHWXBBNWIAVPHWBJPRZNPFFIRWW
Decoded ciphertext:
WHYDZKMYHGHMLJMYGMDBHGFZSHOTKHCMMYAMAZZFHCALHILHAMHKHXJAZMYAMMYHUALHHCKDWHKqjMYGTLILHAMDLWTMYIHLMATCJCAZT
HCABZHLTPYMGMYAMAFDCPMYHGHALHZTSHZTBHLMUACKMYHQJLGJTMDSYAQQTCHGGMYAMMDGHIJLHMYHGHLTPYMGPDOHLCFHCMGALHTCGM
TMJMHKAFDCPFHCKHLTOTCPMSHTLEJGMQDWHLGSLDFMYHIDCGHCMDSMZEPDOHLCHKMYAMWYHHOHLACUSDLFDSPDOHLCFHCMBHIDFHGKHGM
LJIMTOHDSMYHGHHCKGTMTGMYGLTPYMDSMYHQHDQZHMDAZMHLDLMDABDZTGYTMACKMDTCGMTMJMHCHWPDOHLCFHCMZAUTCPTMGSDJCKAMT
DCDCGJIYQLTCITQZZEACKDLPACTVTCPTMGQDWHLGTCGJIYSDLFAGMDGHXUDGMZTNHZUMDHSSHIMMYHTLGASHMUACKYAQQTCHH
```

Figure 3: Output of 'decoded' text of key = 15. Keys of 1-26 proved this is not a simple shift cipher and requires knowledge of frequency analysis.

Knowing that I now need to identify the frequency of each character to develop a dictionary key other than ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’, I attempted to code a frequency analysis script but it was a bit out of my reach for the time so I resorted to this webpage [2] to report the frequency count.



The frequencies of the English language are:

E	T	A	O	I	N	S	H	R	D	L	C	U	M	W	F	G	Y	P	B	V	K	J	X	Q	Z
12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8	2.4	2.4	2.2	2.0	2.0	1.9	1.5	1.0	0.8	0.15	0.15	0.10	0.07

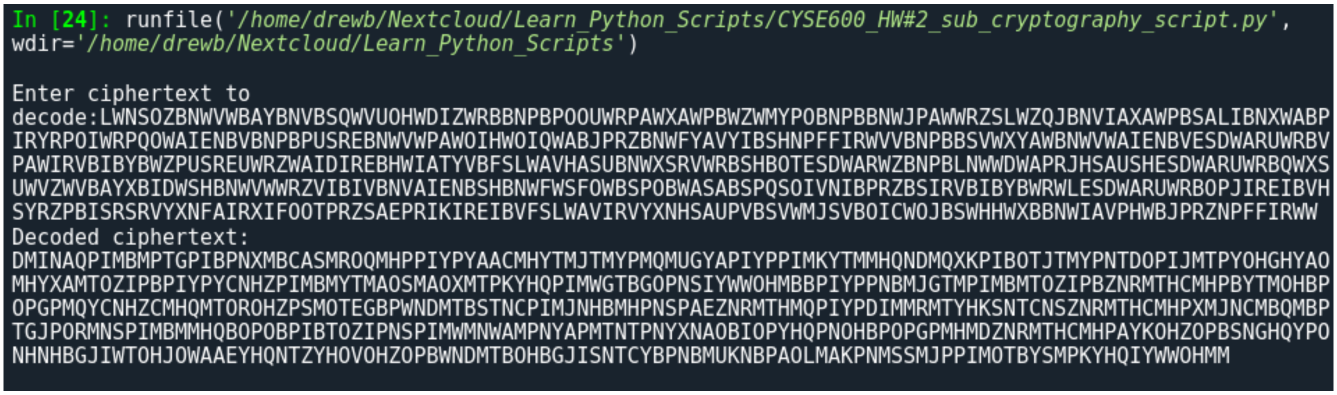
The frequencies of the intercept are:

W	B	R	I	S	A	V	P	N	O	H	Z	E	Y	U	F	X	D	J	L	Q	T	M	C	K	G
71	64	38	36	36	34	34	32	28	18	15	15	13	13	12	11	11	8	8	7	6	3	2	1	1	0
13.7	12.4	7.4	7.0	7.0	6.6	6.6	6.2	5.4	3.5	2.9	2.9	2.5	2.5	2.3	2.1	2.1	1.5	1.5	1.4	1.2	0.6	0.4	0.2	0.2	0.0

Crypto Corner © Daniel Rodriguez-Clark 2017

Figure 4: Showing the ciphertext character frequency and the English language frequency.

The above frequency report shows the frequency of the English letters and the frequency of letters in the ciphertext. At this point, I should be able to input a new key ‘ETAOINSHRDLCUMWFGYPBVKJXQZ’ and see what is returned from my script.



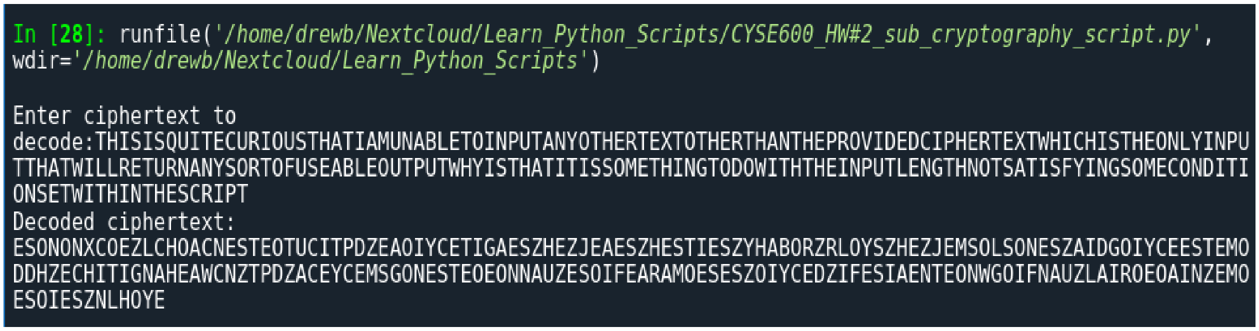
```
In [24]: runfile('/home/drewb/Nextcloud/Learn_Python_Scripts/CYSE600_HW#2_sub_cryptography_script.py',
wdir='/home/drewb/Nextcloud/Learn_Python_Scripts')

Enter ciphertext to
decode:LWNSOZBNWVWBAYBNVBSQWVUOHWDIZWRBBNPBPOOUWRPAWXAWPBWZWMYPOBNPBBNWJPAWWRZSLWZQJBNVIAXAWPBSALIBNXWABP
IRYRPOIWRPQOWAIENBVBNPBPUSREBNWVWPAWOIHWOIQWABJPRZBNWFYAVYIBSHNPFFIRWVVBNPBBSVWXYAWBNWVWAIENBVESDWARUWRBV
PAWIRVBIBYBWZPUSREUWRZWAIDIREBHWIATYVBFSLWAVHASUBNWXSRVWRBSHBOTESDWARWZBNPBLNWWDWAPRJHSAUSHESDWARUWRBQWXS
UWVZWVBAYXBIDWSHBNWVWWRZVIBIVBNVAIENBSHBNWFWSFOWBSPOBWASABSPQSOIVNIBPRZBSIRVBIBYBWRLESDWARUWRBOPJIREIBVH
SYRZPBISRSRVYXNFAIRXIFOOTPRZSAEPRIKIREIBVFSLWAVIRVYXNHSAUPVBSVWMJSVBOICWOJBSWHHWXBBNWIAVPHWBJPRZNPFFIRWW
Decoded ciphertext:
DMINAQPIMBMPTGPIBPNXMBCASMROQMHPPIYPYAACMHYTMJTMYPMQMUGYAPIYPPIMKYTMMHQNDMQXKPIBOTJTMYPNTDOPIJMTPYOHGHYAO
MHYXAMTOZIPBPIYPYCNHZPIMBMYTMAOSMAOXMTPKYHQPIMWGTBGOPNSIYWWOHMBBPIYPPNBMJGTMPIMBMTOZIPBZNRMTHCMHPBYTMOHBP
OPGPMQYCNHZCMHQMTOROHZPSMOTEGBPWNDMTBSTNCPIMJNHBMHPNSPAEZNRMTHMQPIYPDIMMRMTYHKSNTCNSZNRMTHCMHPXMJNCMBQMBP
TGJPORMNSPIMBMMHQBOPOBPIBTOZIPNSPIMWMNWAMPNYAPMTNTPNYXNAOBIOPYHQPNOHBPOPGPMHMDZNRMTHCMHPAYKOHZOPBSNGHQYPO
NHNHBGJIWTOHJOWAAEYHQNTZYHOVOHZOPBWNDMTBOHBGJISNTCYBPNBMUKNBPAOLMAKPNMSSMJPPIMOTBYSMPKYHQIYWWOHMM
```

Figure 5: Output results of string1 alterations from Key = 1 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' to 'ETAOINSHRDLCUMWFGYPBVKJXQZ'.

No human readable output is returned with a different dictionary definition. Essentially, the order of dictionary characters called does not matter so long as all characters of the decoded ciphertext are present in order to properly display the message. It is the shifting and proper character attribution of the decode key that is important for decrypting this ciphertext message. My python script is not functioning in the manner necessary for plaintext recovery. Frequency of character appearance is not being accounted for in the script and character attributions are not being maintained; therefore, simple character shifts are the continuing and sole result of my decoding outputs. In essence, I am more deeply encoding the ciphertext, rather than decoding towards a plaintext message.

As a means to provide me with new ideas for decoding the assigned ciphertext, I started inputting plaintext into the python script where it prompts for user input.

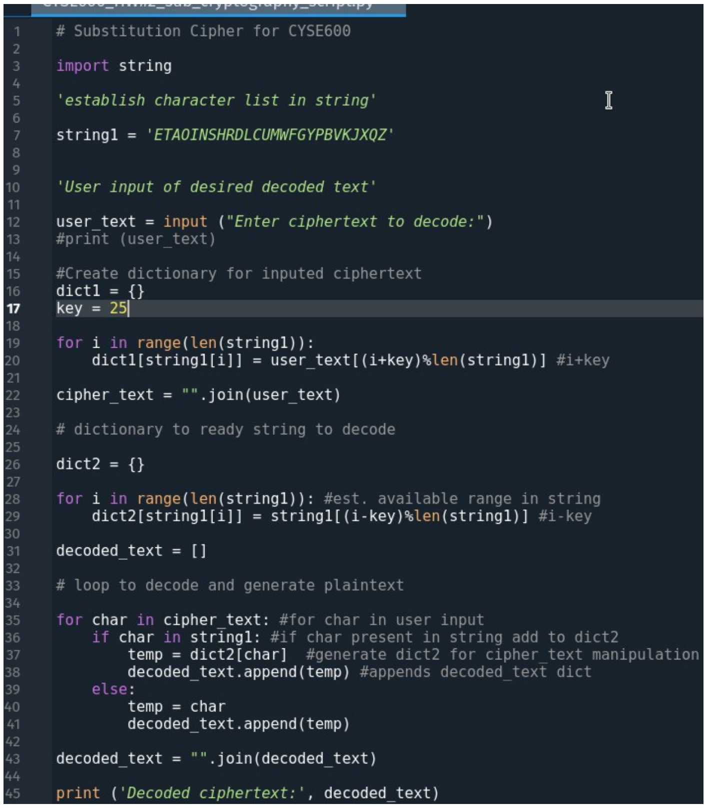
I inputted:



```
In [28]: runfile('/home/drewb/Nextcloud/Learn_Python_Scripts/CYSE600_HW#2_sub_cryptography_script.py',
wdir='/home/drewb/Nextcloud/Learn_Python_Scripts')

Enter ciphertext to
decode:THISISQUITECURIOUSTHATIAMUNABLETOINPUTANYOTHERTEXTOTHERTHANTHEPROVIDEDCIPHERTEXTWHICHISTHEONLYINPU
TTHATWILLRETURNANYSORTOFUSEABLEOUTPUTWHYISTHATITISSOMETHINGTODOWITHTHEINPUTLENGTHNOTSATISFYINGSOMECONDITI
ONSETWITHINTHESCRIPT
Decoded ciphertext:
ESONONXCOEZLCHOACNESTEOTUCITPDZEAOIYCETIGAESZHEZJEAESZHESTIESZYHABORZRLOYSZHEZJEMSOLSONESZAIDGOIYCEESTEMO
DDHZECHITIGNAHEAWCNZTPDZACEYCEMSGONESTEOEONNAUZESOIFEARAMOESESZOIYCEDZIFESIAENTEONWGOIFNAUZLAIROEOAINZEMO
ESOIESZNLHOYE
```

Figure 6: Plaintext inputted into script to observe script behavior and analyze its output in a web-based calculator to see if results mirror anything I have inputted.

The script ran in Fig. 6, I have the string1 character list defined as: ETAOINSHRDLCUMWFGYPBVKJXQZ with a key = 1.

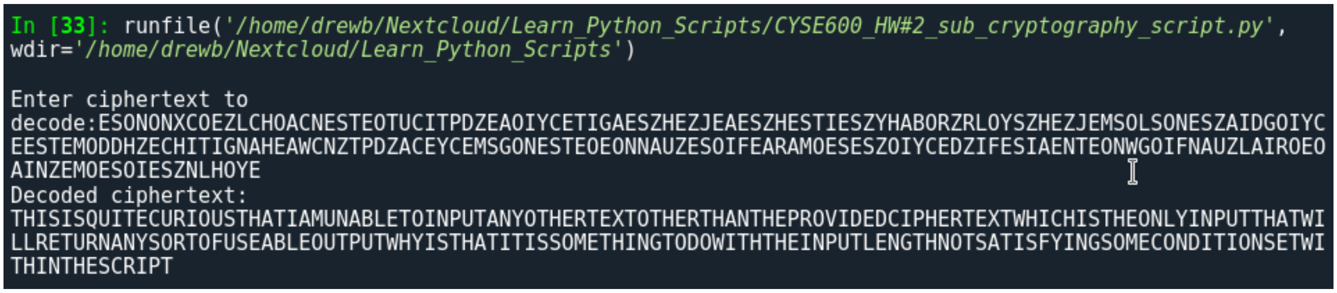


```python
# Substitution Cipher for CYSE600

import string

'establish character list in string'

string1 = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'


'User input of desired decoded text'

user_text = input ("Enter ciphertext to decode:")
#print (user_text)

#Create dictionary for inputed ciphertext
dict1 = {}
key = 25

for i in range(len(string1)):
    dict1[string1[i]] = user_text[(i+key)%len(string1)] #i+key

cipher_text = "".join(user_text)

# dictionary to ready string to decode

dict2 = {}

for i in range(len(string1)): #est. available range in string
    dict2[string1[i]] = string1[(i-key)%len(string1)] #i-key

decoded_text = []

# loop to decode and generate plaintext

for char in cipher_text: #for char in user input
    if char in string1: #if char present in string add to dict2
        temp = dict2[char]  #generate dict2 for cipher_text manipulation
        decoded_text.append(temp) #appends decoded_text dict
    else:
        temp = char
        decoded_text.append(temp)

decoded_text = "".join(decoded_text)

print ('Decoded ciphertext:', decoded_text)
```

Figure 7: Python script example showing key change from key = 1 to key = 25 which successfully decoded my sample input.

At this substitution cipher [3] web page, multiple decryption calculation attempts were necessary to return the proper submitted plaintext of my script input, however, a different string was reported as having encrypted the message: TPLRZWFSOVKDUIAYXHNECBMJGQ.

There is syntax and function operations knowledge that I am missing.



```
In [33]: runfile('/home/drewb/Nextcloud/Learn_Python_Scripts/CYSE600_HW#2_sub_cryptography_script.py',
wdir='/home/drewb/Nextcloud/Learn_Python_Scripts')

Enter ciphertext to
decode:ESONONXCOEZLCHOACNESTEOTUCITPDZEAOIYCETIGAESZHEZJEAESZHESTIESZYHABORZRLOYSZHEZJEMSOLSONESZAIDGOIYC
EESTEMODDHZECHITIGNAHEAWCNZTPDZACEYCEMSGONESTEOEONNAUZESOIFEARAMOESESZOIYCEDZIFESIAENTEONWGOIFNAUZLAIROEO
AINZEMOESOIESZNLHOYE
Decoded ciphertext:
THISISQUITECURIOUSTHATIAMUNABLETOINPUTANYOTHERTEXTOTHERTHANTHEPROVIDEDCIPHERTEXTWHICHISTHEONLYINPUTTHATWI
LLRETURNANYSORTOFUSEABLEOUTPUTWHYISTHATITISSOMETHINGTODOWITHTHEINPUTLENGTHNOTSATISFYINGSOMECONDITIONSETWI
THINTHESCRIPT
```

Figure 8: Changing only key = 1 to key = 25, the encrypted message was properly displayed in plaintext. Identifying the key string used, I'm not sure.

The previous script exercise steps were not to derail, but to showcase my decoding attempts of the assigned ciphertext using rudimentary python scripting skills.

To complete the assignment, I resorted to handwriting the entire ciphertext message and then transcribing plaintext characters above the ciphertext as the character relations became known. About 15 of the 26 characters in the ciphertext were posted within the C2-Cryptography lecture notes. That provided enough decoding to be able to fill in decryption gaps. I attempted to find patterns in the letter spacing of the CT to PT characters moving both forward in the alphabet and backwards in the alphabet. Neither of which produced any usable results.

I have attached handwritten work to the end of the pdf.

This was an interesting assignment which allowed me to work with new scripting skills to try and call in a txt file to attempt frequency analysis of the characters contained within. These attempts were just outside my skill set. More practice.

References:

[1] https://www.geeksforgeeks.org/substitution-cipher/ - Python Script Framework

[2] https://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html – Frequency Analysis Report

[3] https://planetcalc.com/8047/ - substitution cipher calculator

CYSE 600 HW#2 SUBSTITUTION CIPHER DREW BROWN. ①

1	2	3	4	5	6	7			8			9	10		
L	W	N	S	O	Z	B	N	W	V	W	B	A	Y	B	N
W	E	H	O	L	D	T	H	E	S	E	T	R	U	T	H
			13					14		15	11			16	
V	B	S	Q	W	V	W	O	H	W	D	I	Z	W	R	B
S	T	O	B	E	S	E	L	F	E	V	I	D	E	N	T
B	N	P	B	P											
T	H	A	T	A											
		12													

												26														
CT	✓	✓		✓				✓	✓			✓		✓	✓	✓	✓	✓	✓			✓	✓		✓	✓
	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
PT	✓	✓		✓	✓	✓			✓			✓		✓	✓			✓		✓		✓	✓			

	CT	11	PT
1	L	=	W
2	W	8 =	E
3	N	20 =	H
4	S	22 =	O
5	O	=	L
6	Z	=	D
7	B	=	T
8	V	=	S
9	A	18 =	R
10	Y	=	U
11	Q	=	B
12	H	=	F
13	D	=	V
14	I	=	I
15	R	=	N
16	P	=	A
17	C	=	K
18	E	=	G
19	F	=	P
20	G	=	
21	J	=	Y
22	K	=	Z
23	M	=	
24	T	=	J
25	U	=	M
26	X	=	C

T MIGHT BE J

```
WEHOL DTHES ETRUT HSTOB ESELF EVIDE
LWNSO ZBNWV WBAYB NVBSQ WVWOH WDIZW
NTTHA TALLM ENARE CREAT EDEQU ALTHA
RBBNP BPOOU WRPAW XAWPB WZWMY POBNP
TTHEY AREEN DOWED BYTHS IRCRE ATORW
BBNWJ PAWWR ZSLWZ QJBNV IAXAW PBSAL
ITHCE RTAIN UNALI ENABL ERIGH TSTHA
IBNXW ABPIR YRPOI WRPQO WAIEN BVBNP
TAMON GTHES EAREL IFELI BERTY ANDTH
BPUSR EBNWV WPAWO IHWOI QWABJ PRZBN
EPURS UITOF HAPPI NESST HATTO SECUR
WFYAV YIBSH NPFFI RWVVB NPBBS VWXYA
ETHES ERIGH TSGOV ERNME NTSAR EINST
WBNWN WAIEN BVESD WARUW RBVPA WIRVB
ITUTE DAMON GMEND ERIVI NGTHE IRJUS
IBYBW ZPUSR EUWRZ WAIDI REBHW IATYV
TPOWE RSFRO MTHEC ONSEN TOFTH EGOVE
BFSLW AVHAS UBNWX SRVWR BSHBO TESDW
RNEDT HATWH ENEVER ANYFO RMOFG OVERN
ARWZB NPBLN WWDWA PRJHS AUSHE SDWAR
MENTB ECOME SDEST RUCTI VEOFT HESSE
UWRBQ WXSUW VZWVB AYXBI DWSHB NWVWW
NDSIT ISTHE RIGHT OFTHE PEOPL ETOAL
RZVIB IVBNW AIENB SHBNW FWSFO WBSPO
TEROR TOABO LISHI TANDT OINST ITUTE
BWASA BSPQS OIVNI BPRZB SIRVB IBYBW
NEWGO VERNM ENTLA YINGI TSFOU NDATI
RWLES DWARU WRBOP JIREI BVHSY RZPBI
ONONS UCHPR INCIP LESAN DORGA NIZIN
SRSRV YXNFA IRXIF OOTPR ZSAEP RIKIR
GITSP OWERS INSUC HFORM ASTOTH EM v MOS
EIBVF SLWAV IRVYX NHSAU PVBSV WMJSV
TLIKE LYTOE FFECT THEIR SAFET YANDH
BOICW OJBSW HHWXB BNWIA VPHWB JPRZN
APPIN ESS
PFFIR WW
```

SHALL SEEM
V