PART 1: FREQUENCY ANALYSIS

I started by counting the letters in the cipher text, then naively replacing them according to the letter frequency found in the link provided in the lab handout.

The functions I made can be seen below.

home > jaron > Downloads > SUTD > FCS > lab2 > ex1.py

```python
def freqAnalysis(text):
    count = {}
    ref = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    for i in range(len(ref)):
        count[ref[i]] = 0

    for i in range(len(text)):
        if text[i] in ref:
            count[text[i]] += 1

    print(count)

    sortedValues = sorted(count.values())
    freq = ""
    for i in sortedValues:
        for k in count.keys():
            if count[k] == i:
                freq += k
                break

    print(freq)
    return freq

def replaceV1(text, freq):
    standard = "qjzxvkwyfbghmpduclsntoirae" #"eariotnslcudpmhgbfywkvxzjq"
    for i in range(26):
        text = text.replace(freq[i], standard[i])
    return text
```

```python
def doStuff(filein, fileout):
    newText = ""

    with open(filein, mode="r", encoding="utf-8", newline="\n") as fin:
        text = fin.read()
        sortedCount = freqAnalysis(text)
        newText = replaceV1(text, sortedCount)

    with open(fileout, mode="w", encoding="utf-8", newline="\n") as fout:
        fout.write(newText)
```

```python
if __name__ == "__main__":
    # set up the argument parser
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", dest="filein", help="input file", required=True)
    parser.add_argument("-o", dest="fileout", help="output file", required=True)

    # parse our arguments
    args = parser.parse_args()

    filein = args.filein
    try:
        if not os.path.exists(filein):
            raise argparse.ArgumentTypeError()
    except(argparse.ArgumentTypeError):
        print(f"Error: {filein} does not exist")
        sys.exit()

    fileout = args.fileout

    doStuff(filein, fileout)
```

This results in the following “decrypted” text:

```
[jaron@Arch lab2]$ cat out.txt
fsia rn nhpysodeil. Vol i coooootd, coootd arpe r sike tekel woaseleu etdidrtd phnec
V rt asrn Vlitmsrne. r uru toa gtuelnaitu fsia ra rn. tof asia ase nsof rn sikrtd ra
n cina neinot, r uemrueu ao Vrticch drke rt, drke nhpysodeil i alh Vlop ase kelh nai
la. r fotueleu sof sike r prnneu oga ot ase itrpe oV ase uemiue icc asene heiln. r s
otenach uru toa vtof fsia ao ezyema fiamsrtd ase kelh Vrlna eyrnoue vtofrtd iwnocgae
ch toasrtd iwoga ase Vlitmsrne. ase nsof aoheu fras ph epoarotn no pgms rt asia oyet
rtd neayreme. ra etueu gy wertd ote oV ase pona crVe iVVrlprtd nsofn oga asele. i na
gttrtd urnycih oV ruromh itu imarot asia rn woas msilprtd itu miyarkiartd. ra rn mot
Vrueta rt ran naletdasn itu yiliuen ran feivtennen yloguch, i nsof asia rn woas icc
nahce itu icc ngwnaitme. os wga pona oV icc, ra rn i alge loccel moinael oV epoarotn
, itu r uo toa gne asia aelp crdsach. r cigdseu, r mlreu, r doa Vlgnaliaeu ia ase rt
eyarague itu nagyrurah oV woas ase msilimaeln itu ase mleiaoln, wga pona oV icc, r c
okeu. fset ase mglne oV wicic Vecc rt ycime, notdn narcc pitideu ao wlrude asia diy
itu mottema gn icc aodeasel. ase ote epoarot ase nsof tekel Vircn ao uecrkel rn shye
. ase nglde oV iuleticrte itu etuolysrtn itu icc ase mseprmicn rt hogl wlirt fseteke
l nopeasrtd ifenope rn siyyetrtd otnmleet rn i lile aleia rt pona oasel itrpe, wga i
 motnaita ommglletme rt asrn nsof. ase shye uoen omminroticch Virc ao uecrkel, ase s
rdsel ezyemaiarotn mit nopearpen we os aoo pgms, wga narcc, fset hog seil srwrvr nml
eip, hog vtof nse peitn wgnrtenn itu Zort sel rt notd. itu asone notdn ile ylemrnech
 fsia veeyn nhpysodeil rt ase prtun oV pith icc asrn fsrce. r ip toa it ruoc itrpe y
elnot, r uenyrne pona ruoc itrpe itu hea. itu hea. ase popeta r seilu ase notdn Vol
```

I did the substitution with lowercase letters to ensure that each character is replaced at most once. This also allowed me to check if all letters are replaced, and as seen in the output above, there are still some uppercase letters.

Then, I printed the frequency table (**sortedCount** in the code) to see what went wrong.

```
[jaron@Arch lab2]$ python3 ex1.py -i story_cipher.txt -o out.txt
{'A': 20, 'B': 102, 'C': 70, 'D': 205, 'E': 206, 'F': 43, 'G': 0, 'H': 131, 'I': 198
, 'J': 263, 'K': 51, 'L': 29, 'M': 50, 'N': 3, 'O': 58, 'P': 1, 'Q': 213, 'R': 35, '
S': 61, 'T': 98, 'U': 305, 'V': 50, 'W': 71, 'X': 160, 'Y': 229, 'Z': 3}
GPNNALRFMMKOSCWTBHXIDEQYJU
[jaron@Arch lab2]$
```

It turns out that N and Z have 3 counts each, while M and V have 50 counts each. As such, after sorting the keys in the frequency table, I have some entries that are duplicated while some entries are omitted.

I added some code to ensure there is a 1-to-1 mapping. I also took the opportunity to split my **freqAnalysis** function into two separate functions.

```
19  def freqAnalysis(count):
20      sortedValues = sorted(count.values())
21      freq = ""
22      for i in sortedValues:
23          if i == 3:
24              if "N" not in freq:
25                  freq += "N"
26              else:
27                  freq += "Z"
28          elif i == 50:
29              if "V" not in freq:
30                  freq += "V"
31              else:
32                  freq += "M"
33          else:
34              for k in count.keys():
35                  if count[k] == i:
36                      freq += k
37                      break
38
39      print(freq)
40      return freq
```

```
7   def countLetters(text):
8       count = {}
9       ref = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
10      for i in range(len(ref)):
11          count[ref[i]] = 0
12
13      for i in range(len(text)):
14          if text[i] in ref:
15              count[text[i]] += 1
16
17      print(count)
18      return count
```

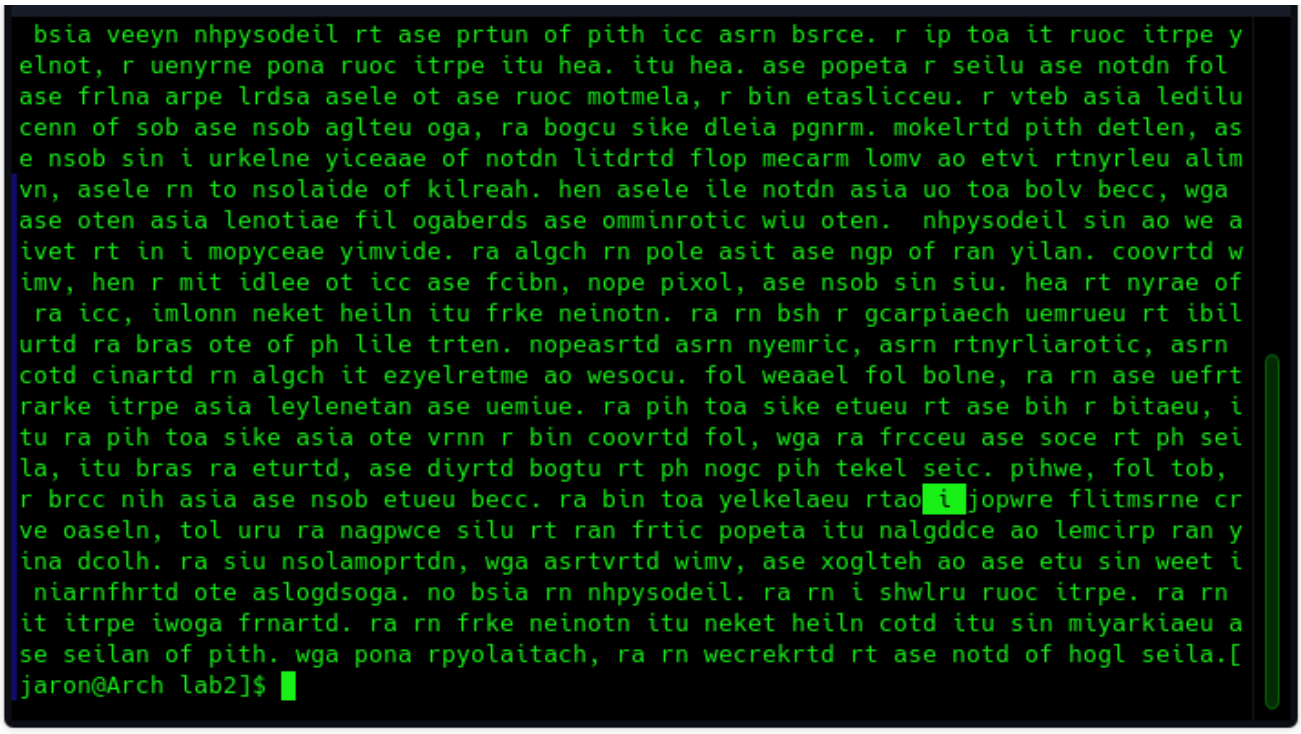
```
def doStuff(filein, fileout):
    newText = ""

    with open(filein, mode="r", encoding="utf-8", newline="\n") as fin:
        text = fin.read()
        count = countLetters(text)
        sortedCount = freqAnalysis(count)
        newText = replaceV1(text, sortedCount)

    with open(fileout, mode="w", encoding="utf-8", newline="\n") as fout:
        fout.write(newText)
```

```
[jaron@Arch lab2]$ python3 ex1.py -i story_cipher.txt -o out.txts
{'A': 20, 'B': 102, 'C': 70, 'D': 205, 'E': 206, 'F': 43, 'G': 0, 'H': 131, 'I': 198
, 'J': 263, 'K': 51, 'L': 29, 'M': 50, 'N': 3, 'O': 58, 'P': 1, 'Q': 213, 'R': 35, '
S': 61, 'T': 98, 'U': 305, 'V': 50, 'W': 71, 'X': 160, 'Y': 229, 'Z': 3}
GPZNALRFVMKOSCWTBHXIDEQYJU
[jaron@Arch lab2]$
```

Now that I have a 1-to-1 mapping, I looked for common short words to better identify certain characters.

```
 bsia veeyn nhpysodeil rt ase prtun of pith icc asrn bsrce. r ip toa it ruoc itrpe y
elnot, r uenyrne pona ruoc itrpe itu hea. itu hea. ase popeta r seilu ase notdn fol
ase frlna arpe lrdsa asele ot ase ruoc motmela, r bin etaslicceu. r vteb asia ledilu
cenn of sob ase nsob aglteu oga, ra bogcu sike dleia pgnrm. mokelrtd pith detlen, as
e nsob sin i urkelne yiceaae of notdn litdrtd flop mecarm lomv ao etvi rtnyrleu alim
vn, asele rn to nsolaide of kilreah. hen asele ile notdn asia uo toa bolv becc, wga
ase oten asia lenotiae fil ogaberds ase omminrotic wiu oten.  nhpysodeil sin ao we a
ivet rt in i mopyceae yimvide. ra algch rn pole asit ase ngp of ran yilan. coovrtd w
imv, hen r mit idlee ot icc ase fcibn, nope pixol, ase nsob sin siu. hea rt nyrae of
 ra icc, imlonn neket heiln itu frke neinotn. ra rn bsh r gcarpiaech uemrueu rt ibil
urtd ra bras ote of ph lile trten. nopeasrtd asrn nyemric, asrn rtnyrliarotic, asrn
cotd cinartd rn algch it ezyelretme ao wesocu. fol weaael fol bolne, ra rn ase uefrt
rarke itrpe asia leylenetan ase uemiue. ra pih toa sike etueu rt ase bih r bitaeu, i
tu ra pih toa sike asia ote vrnn r bin coovrtd fol, wga ra frcceu ase soce rt ph sei
la, itu bras ra eturtd, ase diyrtd bogtu rt ph nogc pih tekel seic. pihwe, fol tob,
r brcc nih asia ase nsob etueu becc. ra bin toa yelkelaeu rtao i jopwre flitmsrne cr
ve oaseln, tol uru ra nagpwce silu rt ran frtic popeta itu nalgddce ao lemcirp ran y
ina dcolh. ra siu nsolamoprtdn, wga asrtvrtd wimv, ase xoglteh ao ase etu sin weet i
 niarnfhrtd ote aslogdsoga. no bsia rn nhpysodeil. ra rn i shwlru ruoc itrpe. ra rn
it itrpe iwoga frnartd. ra rn frke neinotn itu neket heiln cotd itu sin miyarkiaeu a
se seilan of pith. wga pona rpyolaitach, ra rn wecrekrtd rt ase notd of hogl seila.[
jaron@Arch lab2]$
```

```
 bsia veeyn nhpysodeil rt ase prtun of pith icc asrn bsrce. r ip toa it ruoc itrpe y
elnot, r uenyrne pona ruoc itrpe itu hea. itu hea. ase popeta r seilu ase notdn fol
ase frlna arpe lrdsa asele ot ase ruoc motmela, r bin etaslicceu. r vteb asia ledilu
cenn of sob ase nsob aglteu oga, ra bogcu sike dleia pgnrm. mokelrtd pith detlen, as
e nsob sin i urkelne yiceaae of notdn litdrtd flop mecarm lomv ao etvi rtnyrleu alim
vn, asele rn to nsolaide of kilreah. hen asele ile notdn asia uo toa bolv becc, wga
ase oten asia lenotiae fil ogaberds ase omminrotic wiu oten.  nhpysodeil sin ao we a
ivet rt in i mopyceae yimvide. ra algch rn pole asit ase ngp of ran yilan. coovrtd w
imv, hen r mit idlee ot icc ase fcibn, nope pixol, ase nsob sin siu. hea rt nyrae of
 ra icc, imlonn neket heiln itu frke neinotn. ra rn bsh r gcarpiaech uemrueu rt ibil
urtd ra bras ote of ph lile trten. nopeasrtd asrn nyemric, asrn rtnyrliarotic, asrn
cotd cinartd rn algch it ezyelretme ao wesocu. fol weaael fol bolne, ra rn ase uefrt
rarke itrpe asia leylenetan ase uemiue. ra pih toa sike etueu rt ase bih r bitaeu, i
tu ra pih toa sike asia ote vrnn r bin coovrtd fol, wga ra frcceu ase soce rt ph sei
la, itu bras ra eturtd, ase diyrtd bogtu rt ph nogc pih tekel seic. pihwe, fol tob,
r brcc nih asia ase nsob etueu becc. ra bin toa yelkelaeu rtao i jopwre flitmsrne cr
ve oaseln, tol uru ra nagpwce silu rt ran frtic popeta itu nalgddce ao lemcirp ran y
ina dcolh. ra siu nsolamoprtdn, wga asrtvrtd wimv, ase xoglteh ao ase etu sin weet i
 niarnfhrtd ote aslogdsoga. no bsia rn nhpysodeil. ra rn i shwlru ruoc itrpe. ra rn
it itrpe iwoga frnartd. ra rn frke neinotn itu neket heiln cotd itu sin miyarkiaeu a
se seilan of pith. wga pona rpyolaitach, ra rn wecrekrtd rt ase notd of hogl seila.[
jaron@Arch lab2]$
```



I first noticed the letters r and i form words on their own, suggesting they are the letters a and i. I saw other words that start with the letter r, which makes it seem as though r corresponds to i. This also means that i in the “decrypted” text corresponds to a.

I went on to determine which letters in the original ciphertext correspond to r and i in the “decrypted” text, and hardcoded the substitutions for them.

```
50  def replaceV2(text, freq):
51      standard = "qjzxvkwyfbghmpduclsntore"
52      text = text.replace("Q", "a").replace("Y", "i")
53      freq = freq.replace("Q", "").replace("Y", "")
54      print(freq)
55      for i in range(24):
56          text = text.replace(freq[i], standard[i])
57          print(freq[i])
58          print(standard[i])
59      return text
```

```
 bsar veeyn nhpysodeal it rse pitun of path acc rsin bsice. i ap tor at iuoc atipe y
elnot, i uenyine ponr iuoc atipe atu her. atu her. rse popetr i sealu rse notdn fol
rse filnr ripe lidsr rsele ot rse iuoc motmelr, i ban etrslacceu. i vteb rsar ledalu
cenn of sob rse nsob rglteu ogr, ir bogcu sake dlear pgnim. mokelitd path detlen, rs
e nsob san a uikelne yacerre of notdn latditd flop mecrim lomv ro etva itnyileu rlam
vn, rsele in to nsolrade of kalierh. hen rsele ale notdn rsar uo tor bolv becc, wgr
rse oten rsar lenotare fal ogrbeids rse ommaniotac wau oten.  nhpysodeal san ro we r
avet it an a mopycere yamvade. ir rlgch in pole rsat rse ngp of irn yalrn. coovitd w
amv, hen i mat adlee ot acc rse fcabn, nope paxol, rse nsob san sau. her it nyire of
 ir acc, amlonn neket healn atu fike neanotn. ir in bsh i gcriparech uemiueu it abal
uitd ir birs ote of ph lale titen. nopersitd rsin nyemiac, rsin itnyilariotac, rsin
cotd canritd in rlgch at ezyelietme ro wesocu. fol werrel fol bolne, ir in rse uefit
irike atipe rsar leylenetrn rse uemaue. ir pah tor sake etueu it rse bah i batreu, a
tu ir pah tor sake rsar ote vinn i ban coovitd fol, wgr ir ficceu rse soce it ph sea
lr, atu birs ir etuitd, rse dayitd bogtu it ph nogc pah tekel seac. pahwe, fol tob,
i bicc nah rsar rse nsob etueu becc. ir ban tor yelkelreu itro a jopwie flatmsine ci
ve orseln, tol uiu ir nrgpwce salu it irn fitac popetr atu nrlgddce ro lemcaip irn y
anr dcolh. ir sau nsolrmopitdn, wgr rsitvitd wamv, rse xoglteh ro rse etu san weet a
 narinfhitd ote rslogdsogr. no bsar in nhpysodeal. ir in a shwliu iuoc atipe. ir in
at atipe awogr finritd. ir in fike neanotn atu neket healn cotd atu san mayrikareu r
se sealrn of path. wgr ponr ipyolratrch, ir in weciekitd it rse notd of hogl sealr.[
jaron@Arch lab2]$
```

I looked for other short words, such as “rse” which appears many times. This seems to correspond to the word “the”, so I replaced the letters accordingly in a similar fashion as before.

Since my original method would be quite tedious (manually adding **replace** for each of the 26 letters), I came up with an easier method. I used two strings **a** and **b**, both of which being the entire alphabet in uppercase. Then, for each letter than I wish to decrypt, I replace the uppercase letter in **b** with the corresponding lowercase letter. Afterwards, I do the original naive replacement with the remaining letters.

```python
def replaceV2(text, freq):
    a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    b = "ABCDEFGHItKLMNOPaRSTUVWhiZ"
    standard = "qjzxvkwyfbghmpduclsntoirae"

    for i in range(26):
        if a[i] != b[i]:
            text = text.replace(a[i], b[i])
            freq = freq.replace(a[i], "")
            standard = standard.replace(b[i], "")

    for i in range(len(freq)):
        text = text.replace(freq[i], standard[i])
        print(freq[i])
        print(standard[i])

    return text
```

```
f io thin fsaophine. i cic ort gocesntaoc bhat it in. orb that the nhrb in hakiou it
n lant neanro, i cepicec tr fioallm uike io, uike nmdyhrueas a tsm fsrd the kesm nta
st. i brocesec hrb hake i dinnec rgt ro the aoide rf the cepace all thene measn. i h
roentlm cic ort vorb bhat tr ezyept batphiou the kesm fisnt eyinrce vorbiou awnrlgte
lm orthiou awrgt the fsaophine. the nhrb trmec bith dm edrtiron nr dgph io that ryeo
iou netyiepe. it eocec gy weiou roe rf the drnt life affisdiou nhrbn rgt these. a nt
gooiou cinylam rf icirpm aoc aptiro that in wrth phasdiou aoc paytikatiou. it in pro
ficeot io itn ntseouthn aoc yasacen itn beavoennen ysrgclm, a nhrb that in wrth all
ntmle aoc all ngwntaope. rh wgt drnt rf all, it in a tsge srlles prantes rf edrtiron
, aoc i cr ort gne that tesd liuhtlm. i laguhec, i psiec, i urt fsgntsatec at the io
eytitgce aoc ntgyicitm rf wrth the phasaptesn aoc the pseatrsn, wgt drnt rf all, i l
rkec. bheo the pgsne rf walal fell io ylape, nroun ntill daoauec tr wsicue that uay
aoc prooept gn all truethes. the roe edrtiro the nhrb oekes failn tr celikes in hmye
. the ngsue rf acseoalioe aoc eocrsyhion aoc all the phedipaln io mrgs wsaio bheoeke
s nrdethiou abenrde in hayyeoiou ronpseeo in a sase tseat io drnt rthes aoide, wgt a
 prontaot rppgsseope io thin nhrb. the hmye cren rppaniroallm fail tr celikes, the h
iuhes ezyeptatiron pao nrdetiden we rh trr dgph, wgt ntill, bheo mrg heas hiwivi nps
ead, mrg vorb nhe deaon wgnioenn aoc xrio hes io nrou. aoc thrne nroun ase ysepinelm
 bhat veeyn nmdyhrueas io the diocn rf daom all thin bhile. i ad ort ao icrl aoide y
esnro, i cenyine drnt icrl aoide aoc met. aoc met. the drdeot i heasc the nroun frs
the fisnt tide siuht these ro the icrl propest, i ban eothsallec. i voeb that seuasc
lenn rf hrb the nhrb tgsoec rgt, it brglc hake useat dgnip. prkesiou daom ueosen, th
```

The above seems to be the phrase “I laughed, I cried”, so I continued replacing the letters, and I started to see words that are very close to being totally decrypted. From this point, it was a quick and easy iterative process to replace the remaining letters.

I also noticed the encryption used a shift cipher.

The below shows the final mapping. **a** corresponds to the original cipher text, while **b** is for the fully decrypted text.

```
50  def replaceV2(text, freq):
51      a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
52      b = "klmnopqrstuvwxyzabcdefghij"
53      standard = "qjzxvkwyfbghmpduclsntoirae"
54      for i in range(26):
55          if a[i] != b[i]:
56              text = text.replace(a[i], b[i])
57              freq = freq.replace(a[i], "")
58              standard = standard.replace(b[i], "")
59
60      for i in range(len(freq)):
61          text = text.replace(freq[i], standard[i])
62          print(freq[i])
63          print(standard[i])
64
65      return text
```

PART 2: TAMPERING AN OTP ENCRYPTED MESSAGE

I used the technique taught in class: create a mask with parts of the original text that I wish to change (**mask2**) and do an XOR operation with the edited text (**mask1**).

I then did an XOR operation of the result from above (**mod**) with the original cipher text to create the new cipher text.

```
38
39  def hax():
40      # TODO: manipulate ciphertext to decrypt to:
41      # "Student ID 100XXXX gets 4 points"
42      # Remember your goal is to modify the encrypted message
43      # therefore, you do NOT decrypt the message here
44      mask1 = b"00000000001005011000004000000\n"
45      mask2 = b"00000000001000000000000000000\n"
46      mod = XOR(mask1, mask2)
47      new_cipher = XOR(original_cipher, mod)
48      return new_cipher
49
```

```
[jaron@Arch lab2]$ python3 ex2.py
b'Student ID 1000000 gets 0 points\n'
b'Student ID 1005011 gets 4 points\n'
[jaron@Arch lab2]$
```