```
 1 USARKOVBDJPEFLNMWZYIHTCQGX
 2 DYAFSRUCEMNVIXOJKQLTZPHBWG
 3 DEURGVYFBAKHWCLJPTXZSNIOMQ
 4 XGANEKSCPBFRUYJHOILWMVQDZT
 5 CNKMIXHBJVRZOTAYDWLFEPSQUG
 6 IXMBGDYKATQJWCHZOUEPLNVFRS
 7 SQNYHETBIRKOLGDPMWUAVFCXJZ
 8 WPSYEHBRQXNKDFIMVACLJZUOTG
 9 YBKJRFHWCEXTSZILGDPOQAMUNV
10 TNJGFVEIRZLCWBDSUYOKPHQXAM
11 PFSWUTEGBAHDQZNRKVMOLCIXJY
12 PUFIBJAYGSZHMRVLNDWKQETXCO
13 KOWDJUBNVIGATQMPHCSLFREZYX
14 MKDCXTYJOIBNUQPVWHGFRZASEL
15 AMCRJETNDKIYUPSVXWZBOHGQFL
16 IJVFLTOQUBHYWNZMKGXPAECDRS
17 GPUAWMBSDHJLEQROTVYXZCKFNI
18 UNOCVBHEXPIAZWJYGTMFQSDKRL
19 GIJXNBWAZYRUVSOCQKELTDPFMH
20 DWQOYXUNJAIZRHLEGCKTBFSPMV
21 KCIYJRNBSWOAEQUTXVDFZGPMHL
22 HPFBVITYQGXDRMENCSLJKWOZUA
23 XMWVPYJINOSDRAZUKQBGCLFETH
24 AHUCWELZOVGFJXMSIPKTNBRDYQ
25 OXKZLRIQFPCNUSHGYMWBTDVJEA
26 DTZXANOPCUFKMVEJRQISGLBYWH
27 VORYZJCQWMKEGHIPUBDAXLTNFS
28 JYSGERWVZPHOQFKXBNAUMTLCID
29 CTFJLYIXPMWVNSDGBUHQKOAERZ
30 OMDXEQALZVPYWUJSCRIFHBGTKN
31 JCGFMRUNPLVAQOYZBIKXSHTDEW
32 JSUCRIDKQLEHXFVNBOGAYZMWTP
33 TVKDPELHJNZCMWSIFUGQROXABY
34 JVYMEISNHZRFTXLAWOQGBCUKDP
35 FIJBYXLNQHGZWTEUPKSDOMAVCR
36 JDABIQYPXRLGTUVMSFCNHWEOZK
37 JWBKLIVOTQMXZASYRHDUPGEFCN
38 DKEZGYSURVIBAQMTXJCHPNWFOL
39 NRDLFEOBPSTKZUVHJQAWYCIGMX
40 UHVGZECDRXTLNQWOBSMYJPAFKI
```

```
41 HMNUYIEOGJBSACZLPRXTKDFQWV
42 QWAPNHEDYXIOMRJKLVBUTSFGCZ
43 QDFTLNPKCHSYUGAVIBXWJRMEZO
44 AVNORSLQKPTFXYCGJZUHIBDWEM
45 WLPFNUQYBTICMDOVZEXSRAGJKH
46 GBAOLUSERKCDYVWFTMQJPZIHNX
47 RGDNCZVTIJKQBMHAELSWUXPFYO
48 JIFWGPARQNLSTUZDKHMEXBCVYO
49 ULINABRXWDFZTEYHVSCMJQKOPG
50 WYQEKHVSDPLNTOFCRJIUMGZXBA
51 OVACYIHRNJSUEKMLBXQPFWTZDG
52 YXCPGMUWLENOSZVDFJBHRTAKIQ
53 YDJUSVXNIEBCOGMRQKLTPAWZFH
54 GAPOLISMZQCHNRJWDXEUTVFKBY
55 BAQTXSWNPEZHMDLIYGVUKCJOFR
56 MCHXVTNWBISULROJAZYDGPKQEF
57 VMQOSLCXRFYKZJDBAUWIGTENPH
58 NQTCEDJHFZOSPUBAWXRIYGMLVK
59 HEBNTXOKQCVJLWMIZSUDAGYRPF
60 BJXNCHDULEVYWSFAOPZIRGTMKQ
61 MDCTYPREOQVKGSHJFUWILNXZBA
62 CJZKEGOTRPUSAFMQBLDIHVXNYW
63 LEFXPHKRJNWCUIYAQZMOGBVDTS
64 YURBCLNZHXATQWKDEIOGVSFPMJ
65 PAOBYXMLJCTHWFSEVQUGZNDKRI
66 AXTYLVCEDZRGIHMJKNQWOFSPBU
67 LEQKNDSZGWMOHXCIBJPRYFAUTV
68 YDCJSHPVRIBWNUTMXQGLKAZEOF
69 GKFSNHPZATMUBVYDWQLJOCXEIR
70 RKVJWEIAODHPBQUNZXYLGFCMST
71 NVBXPJRDQMWIYACGLFUSZEHTOK
72 HRVLAOEFKYWTIXCMUJGPDSZBQN
73 OVNCYLHUMEQFIRBGDSXWJTAZPK
74 KGBXYULMEPTSZNJOHQFICRVADW
75 JQADWBSYCOLURTFNVZHMEKXIGP
76 ZASEGXTIVDLPMOQYRCKHFUJBNW
77 TJSAGKVFBEOZCMRLIDYWPUHNXQ
78 HGTOBCMFYLXUSWAIQZRDNKPEJV
79 WGPCJKEZMTSYFDLBUHAIRXNQOV
80 UIGOANHCJMVTFBQSELWKDXPRYZ
```

```
 81 WPQCBNYEGDKXIMVHJASRZFOUTL
 82 IUCXKWQYDGFBTANSJREHPOMZLV
 83 WVNZRJKFXMBYIAUDEHTOGSPLCQ
 84 LARKWDBXYNQPJMCEOFZSUHGVIT
 85 XHRKLFJZOEIVCPBQUMYTANWGDS
 86 DGZUONMBIKVYESCWXRLFTHAJQP
 87 GNALHJTFDUPIWXBVRMYEKZOSCQ
 88 TCRDSZUNABMQKIVEPLYJHFOGXW
 89 PBLXDRNYOIZGSTJHKQAFEVUWCM
 90 OBSUWPIAYGMHNJTLRVKCQFZEXD
 91 CSDHBTOLZANIMYEVKXPQGJUFRW
 92 COWEAJVKHXUYFRPSZMLGBNQTID
 93 JEBUWYZVMFPSCKGNTHQLIROXAD
 94 XNYPVBJTUHCDOALIGQFWZKERMS
 95 CXLYHWTNPVSKGZQFJERIMBUOAD
 96 CEVATNKFYMJLBGIZQPOWRHUSXD
 97 CBMKIAPVDOGWJUEHTZYNSQXFLR
 98 OSRWMXFGIVCUDYEHAZKNLQJPTB
 99 KFCTBXESAOGNRZPHWYQUIJMLDV
100 FGOHCWZSTMADNKPBUYXLIRJEVQ
101
```

```
 1 # Name: Vivian Feng and Shriya Muthukumar
 2 # Date: 3/11/2020
 3
 4 import random
 5 import string
 6
 7 # create code pad with certain number of lines and
   write to file
 8 numLines = 100
 9 alphabetList = list(string.ascii_uppercase)
10 # each line will contain 26 letters the message is
   encrypted by going to the line corresponding to the
   position of
11 # the letter in the message and the position of letter
    in the line is the same as the position in the
   message, cycling every 26 lines
12 oneTimePad = open('pad.txt', 'w')
13 for line in range(numLines):
14     newLine = "".join(random.sample(alphabetList, len(
   alphabetList)))
15     oneTimePad.write(newLine + "\n")
16     print newLine
17 oneTimePad.close()
18
19
20 # encryption
21 def encrypt(plaintext, OTPName):
22     global alphabetList
23     # read pad as list
24     OTP = open(OTPName, 'r').read().split("\n")
25     cyphertext = ""
26     line = 0
27     letterPos = 0
28     # for each letter in processed message:
29     for letter in plaintext:
30         #   - go to corresponding line number and
   letter number
31         padLetter = OTP[line][letterPos]
32         # get index of letter on pad
33         padIndex = alphabetList.index(padLetter)
```

```
34          # get plaintext index
35          plainIndex = alphabetList.index(letter)
36          # calculate index of cipher letter
37          cipherIndex = (plainIndex + padIndex) % 26
38          #   - encode by adding index of letter
39          cyphertext += alphabetList[cipherIndex]
40
41          # calculate next lines index
42          line += 1
43          letterPos = (letterPos + 1) % 26
44
45      return cyphertext
46
47
48  # decryption:
49  def decrypt(ciphertext, OTPName):
50      global alphabetList
51      # read pad as list
52      OTP = open(OTPName, 'r').read().split("\n")
53      plaintext = ""
54      line = 0
55      letterPos = 0
56      # for each letter in processed message:
57      for letter in ciphertext:
58          #   - go to corresponding line number
59          #   - go to corresponding letter in line
60          padLetter = OTP[line][letterPos]
61          padIndex = alphabetList.index(padLetter)
62
63          cipherIndex = alphabetList.index(letter)
64
65          #   - decode by substracting index of letter
66          plainIndex = (cipherIndex - padIndex + 26) %
    26  # to ensure index is positive
67          plaintext += alphabetList[plainIndex]
68          line += 1
69          letterPos = (letterPos+1)%26
70
71      return plaintext
72
```

```
73
74 # strip message of spaces and punctuation, make
   message uppercase
75 message = raw_input("Enter your message: ")
76 for char in string.punctuation:
77     while char in message:
78         message = message.replace(char, "")
79 message = message.replace(" ", "")
80 message = message.upper()
81
82 encryptedMessage = encrypt(message,'pad.txt')
83 print "Your encrypted message", encryptedMessage
84 print "Your decrypted message", decrypt(
   encryptedMessage, 'pad.txt')
```