Challenge1 - Shannon Entropy

In information theory, the entropy of a random variable is the average level of uncertainty inherent to the variable's possible outcomes. The ASCII characters are numbered from 0 to 255, hence the maximum entropy of an ASCII string is 8 bits ($28 = 256$) . We have the following ASCII string:

```
#The ASCII string
stringASCII = """6386419283728937821372837289378283178327382737372737612281
95185760759869989057645530122647115911285708569659594024
44531655927273797239218141824248086816028499381698198865
9
39062956454949910471787520911687720233006622996348095941
7
59680645554974661972791876464441493215195714488687853734
6
70177241338593795678290614012152015234783897889911336795
9
368427877691197821889887704400368429082820042418140108973"""

#string wich contains only the numbers from the previous ASCII string
part1 = "638641928372893782137283728937828317832738273737273761228"
part2 = "195185760759869989057645530122647115911285708569659594024"
part3 = "445316559272737972392181418242480868160284993816981988659"
part4 = "390629564549499104717875209116877202330066229963480959417"
part5 = "596806455549746619727918764644414932151957144886878537346"
part6 = "701772413385937956782906140121520152347838978899113367959"
part7 = "368427877691197821889887704400368429082820042418140108973"
stringOfNums = part1 + part2 + part3 + part4 + part5 + part6 + part7
```

Implement a Shannon entropy function!

```
import numpy as np
import string

#Shannon Entropy Function
def shannonEntropy(text):
    """Computes the input text's Shannon Entropy"""

    characterProbabilityMap = {}
    textLen = len(text)

    for char in text:
        if char in characterProbabilityMap:
            characterProbabilityMap[char] += 1 / textLen
        else:
            characterProbabilityMap[char] = 1 / textLen #create the keys from text

    return sum([-P * np.log2(P) for P in characterProbabilityMap.values() if P>0])
```

```python
print(f"The Shannon entropy of the string is {shannonEntropy(stringASCII)}.")
print("The Shannon entropy of the numbers in the string "
      f"(without the \\n symbols) is {shannonEntropy(stringOfNums)}.")
```

```python
#Checked
```

```
The Shannon entropy of the string is 3.3539980796058515.
The Shannon entropy of the numbers in the string (without the \n symbols) is 3.2915202937337
```

Challenge2 - Something wrong

The output of a random number generator is the following:

```python
output = '3897456613370665187213127141567448077196076232687'
```

Find the flaw in this generator!

I made a NIST test on the numbers of the challenge, the test output is the following:

```
Type of Test                                        P-Value                   Conclusion
01. Frequency Test (Monobit)                        0.07186063822585162       Random
02. Frequency Test within a Block                   0.34159973041842334       Random
03. Run Test                                        0.5875475347144012        Random
04. Longest Run of Ones in a Block                  3.9515319071901934e-07    Non-Random
05. Binary Matrix Rank Test                         -1.0                      Non-Random
06. Discrete Fourier Transform (Spectral) Test      0.005905391532593261      Non-Random
07. Non-Overlapping Template Matching Test          0.9997806438208291        Random
08. Overlapping Template Matching Test              nan                       Non-Random
09. Maurer's Universal Statistical test             -1.0                      Non-Random
10. Linear Complexity Test                          -1.0                      Non-Random
11. Serial test:
                                                    6.193980513333234e-39     Non-Random
                                                    2.8726610726784846e-10    Non-Random
12. Approximate Entropy Test                        1.0                       Random
13. Cummulative Sums (Forward) Test                 0.14372114316990908       Random
14. Cummulative Sums (Reverse) Test                 0.1286270420485355        Random
15. Random Excursions Test:
                State       Chi Squared             P-Value                   Conclusion
                -4          7.991670137442733       0.15669530053672578       Random
                -3          3.6207999999999996      0.6051929954436529        Random
                -2          30.987654320987655      9.419970854648779e-06     Non-Random
                -1          16.75                   0.004999164131192691      Non-Random
                +1          4.75                    0.4471459016395012        Random
                +2          2.6666666666666665      0.7512117103661213        Random
                +3          1.6                     0.9012493445012736        Random
                +4          1.1428571428571428      0.95024055775506894       Random
16. Random Excursions Variant Test:
                State       COUNTS                  P-Value                   Conclusion
                -9.0        9                       0.9516507699020016        Random
                -8.0        10                      0.897278961260083         Random
                -7.0        14                      0.6773916019262776        Random
                -6.0        16                      0.5464935954065822        Random
                -5.0        15                      0.5596689271994115        Random
                -4.0        16                      0.4496917979688909        Random
                -3.0        23                      0.09353251268909311       Random
                -2.0        23                      0.03038282197657749       Random
                -1.0        15                      0.08011831372763417       Random
                +1.0        1                       0.08011831372763417       Random
```

We can see that the numbers failed multiple tests, so we can be sure that the random number generator is not good enough.

Before the NIST test I tried some easier methods like entropy calculation and compression with zip, but neither of them gave me bad results.

```
import secrets #to generate cryptograpycly safe random numbers
import os #for zip files and sizes

maximumEntropyForDecimals = shannonEntropy('0123456789')
outputEntropy = shannonEntropy(output)

#For numberMin I generated 10000 numbers with the same number of digits as the original outp
#then I chose the one which has minimum entropy over the 10000 numbers.
numberMin = ""
numberMinEntropy = float("inf")
for i in range(10000):
    num = ''.join(secrets.choice('0123456789') for i in range(len(output)))
    numEntropy = shannonEntropy(num)
    if numEntropy < numberMinEntropy:
        numberMin = num
        numberMinEntropy = numEntropy

print(f"Maximum entrophy for decimals is {maximumEntropyForDecimals}")
print(f"The output's entropy is {outputEntropy}")

print("\n10000 cryptographycly safe random numbers' minimum entropy with the generated numbe
print(f"number = {numberMin}, entropy = {numberMinEntropy}.")
```

```
Maximum entropy for decimals is 3.321928094887362
The output's entropy is 3.180011375826478

10000 cryptographycly safe random numbers' minimum entropy with the generated number:
number = 3721246486877823062260336877347638733671346877268, entropy = 2.8098284066955057.
```

The entropy of the output seems ok, I try to investigate compression. I created three .txt files and used zip on them.

```
f1 = open("numOut.txt", 'w')
f1.write(output)
f1.close()

f2 = open("numMin.txt", "w")
f2.write(numberMin)
f2.close()

f3 = open("numMax.txt", "w")
f3.write(''.join(secrets.choice('0123456789') for i in range(len(output))))
f3.close()
```

```python
os.system("zip numOut.zip numOut.txt")
os.system("zip numMin.zip numMin.txt")
os.system("zip numMax.zip numMax.txt")

#os.system("stat -c%s numMax.zip")

numMaxSize = os.popen("stat -c%s numMax.zip").read()[:-1]
numMinSize = os.popen("stat -c%s numMin.zip").read()[:-1]
numOutSize = os.popen("stat -c%s numOut.zip").read()[:-1]

print(f"Safe random number zipfile size: {numMaxSize} bytes.")
print(f"Minimum entropy safe random number zipfile size: {numMinSize} bytes.")
print(f"The output number zipfile size: {numOutSize} bytes.")
```

```
updating: numOut.txt (deflated 22%)
updating: numMin.txt (deflated 27%)
updating: numMax.txt (deflated 24%)
Safe random number zipfile size: 207 bytes.
Minimum entropy safe random number zipfile size: 206 bytes.
The output number zipfile size: 208 bytes.
```

No significant difference between compressions.

Challenge3 - Encrypted text

The encryption algorithm is unknown. Implement a solution which decipher the following text:

```python
text = """Vfkmfj Cfireu Lezmvijzkp zj r Ylexrizre glsczt ivjvrity
lezmvijzkp srjvu ze Slurgvjk. Wfleuvu ze 1635, VCKV zj fev
fw kyv crixvjk reu dfjk givjkzxzflj glsczt yzxyvi vultrkzfe
zejkzklkzfej ze Ylexrip. Kyv 28000 jkluvekj rk VCKV riv
fixrezqvu zekf ezev wrtlckzvj, reu zekf ivjvrity zejkzklkvj
cftrkvu kyiflxyflk Slurgvjk reu fe kyv jtvezt srebj fw kyv
Urelsv. VCKV zj rwwzczrkvu nzky 5 Efsvc crlivrkvj, rj nvcc
rj nzeevij fw kyv Nfcw Gizqv, Wlcbvijfe Gizqv reu Rsvc Gizqv,
kyv crkvjk fw nyzty nrj Rsvc Gizqv nzeevi Crjqcf Cfmrjq ze 2021."""
```

I tried the Caesar Cipher (Shift) decoder online and I think I got the solution:

Eotvos Lorand University is a Hungarian public research university based in Budapest. Founded in 1635, ELTE is one of the largest and most prestigious public higher education institutions in Hungary. The 28000 students at ELTE are organized into nine faculties, and into research institutes located throughout Budapest and on the scenic banks of the Danube. ELTE is affiliated with 5 Nobel laureates, as well as winners of the Wolf Prize, Fulkerson Prize and Abel

Prize, the latest of which was Abel Prize winner Laszlo Lovasz in 2021.

Next I will implement my own Caesar Cipher decoder.

```
#PREREQUISITES
##########################################################################################

#Necessary imports
import string
import numpy as np
from IPython.display import clear_output

# Characters to shift
lowerCaseCharacters = string.ascii_letters[:26]
upperCaseCharacters = string.ascii_letters[26:]

#relative letter frequency in the English language (source: Wikipedia)
abcLetterFrequencyDic = {"a" : 0.082, "b" : 0.015, "c" : 0.027, "d" : 0.047, "e" : 0.13, "f"
                         "h" : 0.062, "i" : 0.069, "j" : 0.0014, "k" : 0.0078, "l" : 0.041, "m"
                         "n" : 0.067, "o" : 0.078, "p" : 0.019, "q" : 0.0011, "r" : 0.059, "s"
                         "t" : 0.096, "u" : 0.027, "v" : 0.0097, "w" : 0.024, "x" : 0.0015, "y"

#The sum of the probabilities is greater than 1, so I needed to normalize the values (doubl
sumOfProbability = sum(abcLetterFrequencyDic.values())

for key in abcLetterFrequencyDic:
    abcLetterFrequencyDic[key] /= sumOfProbability

##########################################################################################

#Implement Caesar encoding and decoding function as caesarCipher
##########################################################################################

def charShift(char, shift, abc):
    """Shifts a character (char) with the shift parameter (shift) on the given characterlis

    charIndex = abc.index(char)
    newCharIndex = (charIndex + shift) % len(abc)

    return abc[newCharIndex]

def caesarCipher(text, key, abc, ABC=[], decode=False):
    """Encodes text when 'decode' == False and decodes when 'decode' == True using Caesar c
       given parameters."""
    if decode:
        key = -key
```

```
    shiftedText = []
    for char in text:
        if char in abc:
            shiftedText.append(charShift(char,key,abc))
        elif char in ABC:
            shiftedText.append(charShift(char,key,ABC))
        else:
            shiftedText.append(char)

    return ''.join(shiftedText)

############################################################################################

#Caesar cracker with bruteforce
def caesarCrackerBruteForce(text, abc, ABC=[]):

    for key in range(1,len(abc)):
        clear_output()
        print(caesarCipher(text, key, abc, ABC, decode=True))
        condition = input('\nIs this a meaningful text? (y : yes, other: no)')
        if condition == "y":
            return key

    print("Couldn't find key for Caesar code, the function returned key zero.")
    return 0

#Caesar cracker with letter frequency analysis
############################################################################################

def MSE(list1, list2):
    """Computes the mean squared error of two lists if this possible (the lists need to hav
       and should contain only numeric values like ints, floats or doubles)."""
    if len(list1) == len(list2):
        distance = 0
        for value1, value2 in zip(list1, list2):
            distance += (value1 - value2) ** 2

        return np.sqrt(distance) / len(list1)

    else:
        print("The two list don't have the same length.\nMSE returns 0.")
        return 0

def createCharProbabilityDictionary(text, abc, ABC=[]):
    """This function counts the abc characters occurences in a dictionary, normalize it and
       the dictionary."""
```

```python
    charCount = {}
    for char in abc:
        charCount[char] = 0

    for char in text:
        if char in abc:
            charCount[char] += 1
        elif char in ABC:
            charCount[abc[ABC.index(char)]] += 1

    #charCount become charProbability
    sumOfCharCount = sum(charCount.values())
    for key in charCount:
        charCount[key] /= sumOfCharCount

    return charCount

def caesarCrackerFrequency(text, abc, abcLetterFrequencyDic, ABC=[]):
    """Crack the Caesar cipher text with letter frequency analysis"""

    textLetterProbabilityDic = createCharProbabilityDictionary(text,abc,ABC)

    textLetterProbabilityList = [P for P in textLetterProbabilityDic.values()]
    abcLetterFrequencyList = [P for P in abcLetterFrequencyDic.values()] # also propability

    #Shift with key and compute distance (MSE) between the abc and the text's letter freque
    letterFrequencyDistances = []
    for key in range(len(abc)):
        letterFrequencyDistances.append(MSE(np.roll(textLetterProbabilityList,-key), abcLett

    #Key with the minimum distance value has the highest chance of success.
    return letterFrequencyDistances.index(min(letterFrequencyDistances))

##############################################################################################

#Bruteforce Caesar Cracker
print(f"The key value was {caesarCrackerBruteForce(text, lowerCaseCharacters, upperCaseChara
```

```
Eotvos Lorand University is a Hungarian public research
university based in Budapest. Founded in 1635, ELTE is one
of the largest and most prestigious public higher education
institutions in Hungary. The 28000 students at ELTE are
organized into nine faculties, and into research institutes
located throughout Budapest and on the scenic banks of the
Danube. ELTE is affiliated with 5 Nobel laureates, as well
as winners of the Wolf Prize, Fulkerson Prize and Abel Prize,
the latest of which was Abel Prize winner Laszlo Lovasz in 2021.
```

```
Is this a meaningful text? (y : yes, other: no)y
The key value was 17.
```

```
#Caesar decoding with caesarCipher function (key=17)
print(caesarCipher(text, 17, lowerCaseCharacters, upperCaseCharacters, decode=True))
```

```
Eotvos Lorand University is a Hungarian public research
university based in Budapest. Founded in 1635, ELTE is one
of the largest and most prestigious public higher education
institutions in Hungary. The 28000 students at ELTE are
organized into nine faculties, and into research institutes
located throughout Budapest and on the scenic banks of the
Danube. ELTE is affiliated with 5 Nobel laureates, as well
as winners of the Wolf Prize, Fulkerson Prize and Abel Prize,
the latest of which was Abel Prize winner Laszlo Lovasz in 2021.
```

```
#The output is a key value with the highest probability of success according to letter freq
caesarCrackerFrequency(text, lowerCaseCharacters, abcLetterFrequencyDic, ABC=upperCaseCharac
```

```
17
```

Challenge4 - PRNG (X ,Y ,Z)

Random numbers are very important in many fields of computer science. Predictable random numbers can pose high security issues in modern applications. The following formula were used to generate random numbers:

$$A_{n+1} \equiv (XA_n + Y)(mod Z)$$

```
A_60 = 246416751162076914019450614023070953069
A_61 = 71744889648624900918616152933820948112
A_62 = 313795302357961401576505497564088201464
A_63 = 65184588491602661601360554078566915563
A_64 = 324784228708505567112999524522359547226
A_65 = 261576664269229262997120467444864381253
A_66 = 91964492393066574896153531497877807434
A_67 = 134532471980964472373259171662351157113


#Check results are the same:
#246416751162076914019450614023070953069
#71744889648624900918616152933820948112
#313795302357961401576505497564088201464
#65184588491602661601360554078566915563
#324784228708505567112999524522359547226
#261576664269229262997120467444864381253
```

```
#91964492393066574896153531497877807434
#134532471980964472373259171662351157113
```

Calculate the next number A_{68}!

```
#Solution from class

import functools,gmpy2,binascii

#Euler algorithm
def gcd(x, y):

    while(y):
        x, y = y, x % y

    return x


def _crack_unknown_increment(states, modulus, multiplier):
    increment = (states[1] - states[0]*multiplier) % modulus
    return modulus, multiplier, increment

def _crack_unknown_multiplier(states, modulus):
    inv = int(gmpy2.invert(states[1]-states[0] % modulus, modulus))
    multiplier = ((states[2] - states[1]) * inv) % modulus
    return _crack_unknown_increment(states, modulus, multiplier)

def _crack_unknown_modulus(states):
    diffs = [s1 - s0 for s0, s1 in zip(states, states[1:])]
    zeroes = [t2*t0 - t1*t1 for t0, t1, t2 in zip(diffs, diffs[1:],diffs[2:])]
    modulus = abs(functools.reduce(lambda x,y: gcd(x,y),zeroes))
    return _crack_unknown_multiplier(states, modulus)

def crack(seq):
    return _crack_unknown_modulus(seq)


states=[A_60, A_61, A_62, A_63, A_64, A_65, A_66, A_67]
output=crack([states[0],states[1],states[2],states[3],states[4],states[5],states[6],states[7

print("Recovered modulus     : ", output[0])
print("Recovered multiplier  : ", output[1])
print("Recovered incrementer : ", output[2])
```

```
Recovered modulus     :  33701941651768000179061142349242166739
Recovered multiplier  :  336410002395367246902039687686647440517
```

```
Recovered incrementer :  29166375975227388766220139676326874847З
```

```python
# Check the results

#LCG algorithm
def LCG(An, X, Y, Z):
    return (X * An + Y) % Z

An = 24641675116207691401945061402307095306‌9
print(An)
for i in range(7):
    An = LCG(An,output[1],output[2],output[0])
    print(An)

print("\n\n")
print(f"The next number, A_68 = {LCG(An,output[1],output[2],output[0])}")
```

```
246416751162076914019450614023070953069
71744889648624900918616152933820948112
313795302357961401576505497564088201464
65184588491602661601360554078566915563
324784228708505567112999524522359547226
261576664269229262997120467444864381253
91964492393066574896153531497877807434
134532471980964472373259171662351157113



The next number, A_68 = 226704693891942394351159226135530835265
```