Kenton Carrier
Homework 4

1. I knew from our notes that once you find the key length, there are formulas that can be used to determine the most likely key. I implemented this first in several functions; I made a function for separating the ciphertext into substrings based on the key length, a function implementing the theorem for estimating a character in the key, and the actual decryption algorithm. For finding the key length, I referred to the following link for guidance: https://inventwithpython.com/hacking/chapter21.html. Implementing these functions was more difficult since there was no set formula for frequency analysis or factorization. If I had more time I would make the program more flexible, but to save effort and time, I hardcoded the value for key length after finding the key length of 4.

   The plaintext has no E's, so the frequency analysis was thrown off since E is the most common letter in the English alphabet.

   Results of my functions are below:
   - Repeating Subsequences of length 3

     {'owa': 2, 'waf': 2, 'mft': 4, 'qbi': 2, 'rys': 2, 'aoq': 2, 'oqs': 2, 'qsa': 2, 'soa': 2, 'oao': 2, 'cjk': 2, 'aar': 2, 'arm': 2, 'ywx': 2, 'cyl': 2, 'afo': 2, 'rgg': 2, 'crg': 2, 'uhg': 2, 'rso': 2}

   - Distance between repeated subsequences

     {'owa': [152], 'waf': [152], 'mft': [284, 28], 'qbi': [20], 'rys': [83], 'aoq': [28], 'oqs': [28], 'qsa': [28], 'soa': [220], 'oao': [220], 'cjk': [152], 'aar': [200], 'arm': [200], 'ywx': [44], 'cyl': [48], 'afo': [20], 'rgg': [28], 'crg': [4], 'uhg': [33], 'rso': [4]}

   - Factors of distances between subsequences

     [3, 2, 4, 2, 4, 5, 2, 4, 2, 4, 2, 3, 4, 6, 2, 4, 5, 2, 4, 5, 2, 4, 2, 4]

   - Numbers of occurrences of each factor less than or equal to 6

     {2: 9, 3: 2, 4: 9, 5: 3, 6: 1}

   - Likely Key

     ['N', 'O', 'E', 'S']

   - Plaintext

UPONTHISBASISIAMGOINGTOSHOWYOUHOWABUNCHOFBRIGHTYOUNGF
OLKSDIDFINDACHAMPIONAMANWITHBOYSANDGIRLSOFHISOWNAMANOF
SODOMINATINGANDHAPPYINDIVIDUALITYTHATYOUTHISDRAWNTOHIMAS
ISAFLYTOASUGARBOWLITISASTORYABOUTASMALLTOWNITISNOTAGOSS
IPYYARNNORISITADRYMONOTONOUSACCOUNTFULLOFSUCHCUSTOMAR
YFILLINSASROMANTICMOONLIGHTCASTINGMURKYSHADOWSDOWNALO
NGWINDINGCOUNTRYROADITWASNTTHATHARDWASI

2. The implementation for this problem was not as difficult as the first one. I had never heard of Blum-Blum-Shub, so understanding that algorithm and how to implement it was the most challenging portion of the problem.

   I found the results interesting because at that large length of a sequence, the averages and frequencies were very even. For all the seeds I tested, the average number of zeros was always close to 500, half of the 1000 bit substring. All of the frequencies for the subsequences of length four were also very close across all sixteen options.