In [12]:

```
Alphabet = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','

Unscaled = [8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2.0, 6.1, 7.0, 0.15, 0.77, 4.0

EnglishFreqs = [i/100 for i in Unscaled]

def CleanText(Text):   #Removes all characters that are not in the 26 let
    CleanText = ""
    for Char in Text:
        if Char.upper() in Alphabet or Char == " ":
            CleanText = CleanText + Char.upper()
    return(CleanText)

def RemoveSpaces(Text):   #Removes all spaces
    NoSpaceText = ""
    for Char in Text:
        if Char.upper() in Alphabet:
            NoSpaceText = NoSpaceText + Char.upper()
    return(NoSpaceText)

def CharacterCount(Text):   #Counts the number of times each letter appea
    CharacterCount = [0] * 26
    for Char in Text:
        if Char.upper() in Alphabet:
            k = Alphabet.index(Char.upper())
            CharacterCount[k] = CharacterCount[k] + 1
    return(CharacterCount)

def Frequencies(Text):   #Gives the frequency of each letter in Text
    LenNoSpaces = len(RemoveSpaces(CleanText(Text)))
    Frequencies = [i/LenNoSpaces for i in CharacterCount(Text)]
    return(Frequencies)

def CompareFreqs(Freqs1, Freqs2):   #Produces a bar chart comparing two f
    Freqs1Spaced = []
    Freqs2Spaced = []
    for i in range(26):
        Freqs1Spaced.append(Freqs1[i])
        Freqs1Spaced.append(0)
        Freqs1Spaced.append(0)
        Freqs2Spaced.append(0)
        Freqs2Spaced.append(Freqs2[i])
        Freqs2Spaced.append(0)
    G=Graphics()
    G += bar_chart(Freqs1Spaced, axes = False, width = 0.8)
    G += bar_chart(Freqs2Spaced, rgbcolor=(1,0,0), axes = False, width =
    return(G)

def Grouping(Text, Group):   #Arranges the input text into groups of size
    Count = 0
    GroupedText = ""
    for Char in Text:
        if Char.upper() in Alphabet:
            if Count > 0 and Count % Group == 0:
                GroupedText = GroupedText + " " + Char.upper()
            else:
                GroupedText = GroupedText + Char.upper()
            Count = Count + 1
    return(GroupedText)

def ShiftEncryption(Text, Shift):   #Encrypts Text by shifting each lette
    CryptText = ""
    for Char in Text:
        if Char == " ":
            CryptText = CryptText + " "
        elif Char.upper() in Alphabet:
            CryptText = CryptText + Alphabet[(Alphabet.index(Char.upper()
    return(CryptText)

def ShiftDecryption(Text, Shift):   #Reverses ShiftEncryption
    DecryptText = ShiftEncryption(Text, -Shift)
    return(DecryptText)

def IndexOfCoincidence(Text):
    CharacterCount = [0] * 26   #Initialise a list of 26 copies of 0.
    for Char in Text:
        if Char.upper() in Alphabet:   #Loop over every character in the
            i = Alphabet.index(Char.upper())   #Find the position of the
            CharacterCount[i] = CharacterCount[i] + 1   #Add 1 to the pos
    Sum = 0
    N = len(RemoveSpaces(CleanText(Text)))   #N is the total number of le
    for i in CharacterCount:
        if i > 1:
            Sum = Sum + ( i * (i-1) ) / ( N * (N-1) )
    return(Sum.numerical_approx(digits = 4))

def RandomText(Length):   #Generates a text with Length characters, chose
    Text = ""
    Count = 0
    while Count < Length:
        Text = Text + Alphabet[floor(random()*26)]
        Count = Count + 1
    return(Text)

def VigenereEncryption(Text, Key):   #Encrypts Text using the Vigenere ci
    NumKey = []
    for Char in Key:
        NumKey.append(Alphabet.index(Char.upper()))
    LenKey = len(Key)
    CryptText = ""
    Count = 0
    for i in range(len(Text)):
        Char = Text[i].upper()
        if Char.upper() in Alphabet:
            CryptText = CryptText + Alphabet[(Alphabet.index(Char) + NumK
            Count = Count + 1
        else:
            CryptText = CryptText + Char
    return(CryptText)

def VigenereDecryption(Text, Key):   #Decrypts Text using the Vigenere ci
    NumKey = []
    for Char in Key:
        NumKey.append(Alphabet.index(Char.upper()))
    LenKey = len(Key)
    Decrypt = ""
    Count = 0
    for i in range(len(Text)):
        Char = Text[i].upper()
        if Char.upper() in Alphabet:
            Decrypt = Decrypt + Alphabet[(Alphabet.index(Char) - NumKey[C
            Count = Count + 1
        else:
            Decrypt = Decrypt + Char
    return(Decrypt)

def PeriodicTexts(Text, Period):   #Extracts every (Period)th letter from
    PeriodicTexts = [""] * Period
    NewText = RemoveSpaces(CleanText(Text))
    for i in range(len(NewText)):
        PeriodicTexts[i % Period] = PeriodicTexts[i % Period] + NewText[i
    return(PeriodicTexts)

def ChiSquared(Text, Period, StartingPosition):
    ExtractedText = PeriodicTexts(Text, Period)[StartingPosition]   #Extr
    N = len(ExtractedText)
    Counts = CharacterCount(ExtractedText)   #This counts the number of t
    ChiValues = []
    for Shift in range(26):
        Chi = 0
        for i in range(26):
            ShiftedFreq = EnglishFreqs[(i - Shift) % 26]   #This is the f
            Chi = Chi + (Counts[i] - ShiftedFreq * N)^2 / (ShiftedFreq *
        ChiValues.append(Chi)
    return(bar_chart(ChiValues))
```

In [13]:

```
WikiText = CleanText("The various patterns of activity are thought to be
WikiText
```

Out[13]:

```
'THE VARIOUS PATTERNS OF ACTIVITY ARE THOUGHT TO BE MAINLY ANTIPREDATOR
ADAPTATIONS THOUGH SOME COULD EQUALLY WELL BE PREDATORY ADAPTATIONS MANY
PREDATORS FORAGE MOST INTENSIVELY AT NIGHT WHEREAS OTHERS ARE ACTIVE AT
MIDDAY AND SEE BEST IN FULL SUN THE CREPUSCULAR HABIT MAY BOTH REDUCE PR
EDATION PRESSURE INCREASING THE CREPUSCULAR POPULATIONS AND OFFER BETTER
FORAGING OPPORTUNITIES TO PREDATORS THAT INCREASINGLY FOCUS THEIR ATTENT
ION ON CREPUSCULAR PREY UNTIL A NEW BALANCE IS STRUCK SUCH SHIFTING STAT
ES OF BALANCE ARE OFTEN FOUND IN ECOLOGY SOME PREDATORY SPECIES ADJUST T
HEIR HABITS IN RESPONSE TO COMPETITION FROM OTHER PREDATORS FOR EXAMPLE
THE SUBSPECIES OF SHORTEARED OWL THAT LIVES ON THE GALPAGOS ISLANDS IS N
ORMALLY ACTIVE DURING THE DAY BUT ON ISLANDS LIKE SANTA CRUZ THAT ARE HO
ME TO THE GALAPAGOS HAWK THE OWL IS CREPUSCULAR APART FROM THE RELEVANCE
TO PREDATION CREPUSCULAR ACTIVITY IN HOT REGIONS ALSO MAY BE THE MOST EF
FECTIVE WAY OF AVOIDING HEAT STRESS WHILE CAPITALIZING ON AVAILABLE LIGH
T'
```

In [ ]:

```
bar_chart(Frequencies(WikiText))
```

In [ ]:

```
CompareFreqs(EnglishFreqs, Frequencies(WikiText))
```

In [ ]:

```
bar_chart(Frequencies(RandomText(1000)))
```

In [ ]:

```
IndexOfCoincidence(RandomText(1000))
```

In [ ]:

```
IndexOfCoincidence(WikiText)
```

In [ ]:

```
CryptText = "UIGIP FOFVI VVDLO SHPSR NSTBN EWIRH YNOWO SKLKN SCPTR EDFNE
CryptText
```

In [ ]:

```
PeriodicTexts(CryptText, 2)
```

In [ ]:

```
IndexOfCoincidence(PeriodicTexts(CryptText, 2)[0])
```

In [ ]:

```
MeanIndices = [0]
for Period in range(1,12):
    PTexts = PeriodicTexts(CryptText, Period)
    Indices = []
    for i in range(Period):
        Indices.append(IndexOfCoincidence(PTexts[i]))
    MeanIndices.append(mean(Indices))
bar_chart(MeanIndices)
```

In [141…

```
PTexts = PeriodicTexts(CryptText, 2)
```

In [ ]:

```
bar_chart(Frequencies(PTexts[0]))
```

In [ ]:

```
CompareFreqs(EnglishFreqs, Frequencies(ShiftEncryption(PTexts[0],-1)))
```

In [ ]:

```
RemoveSpaces(VigenereDecryption(CryptText,"AA"))
```

In [ ]:

```
ChiSquared(CryptText, 2, 0)
```

In [11]:

```
# Can you decode the following crypttext?

CryptText = "PBLPK GMZAE OBEEG RIGCZ TITBA NLSRW WJFWJ HWGWV ITURL LCMQF
CryptText
```

Out[11]:

```
'PBLPK GMZAE OBEEG RIGCZ TITBA NLSRW WJFWJ HWGWV ITURL LCMQF VMXBL RVMPM
WFWZG KIYUL CIBRJ WKGMN ULEAI CLITM PRHPC RMEAW VTAXW HYBBU VIHXI GARIX
DRJCG GMZQA KMPZW PGXIA VVCGO RVEVM IPCWC WLVLM QGIYE IVAWQ KEXTQ YSFNX
BBLLG ITNQI TBVPD YFXSA GGMBV TGYVX VREMG LEVLL CWWBJ EPWNV FMUAQ AYQQO
MFWEE AAGSK GAIFS ZCKQR VHGLQ TFGQF XBKIF HNYSV IXWCW RTHWZ KAKMP ZSRAP
ILKJQ KMAWQ KXAGG ERIZB SGJMP RHPCR MEESU MKUST VXZFZ EXXBU WTNTG RJXCD
MPGRV KWYGJ CWQSX ITXVG ULCKI PLITX IPZGJ TZNUX GKJRZ EXXAQ AJHXZ RFXNR
XBKWG LAVFK WGQDM ICUQY AXKXA YAOGW WQYMP ZWEVI HXIGA RIHXC GRGGB FFSPE
MGZEN EGFGQ GVPNJ EEMME KGCGK UGSUX NEGQC LMYWG VBWAG JYXIC GRUHZ ZSWML
BUWPC MBRJS HPPVU LOHLV XMGLB UWMTT JVDMV BMFVI RXVQA RIHVG ZIOTA XULQL
MA'
```