Gavin Koma
Dr. Justin Shi
Scripting for Sciences and Business
Wednesday, June 2022

Lab 4: Python for Bioinformatics

Rubric: Questions and Point Values	
[1]	(1 pt) Compose a Python program to calculate the total number of DNA bases, and totals for ‘A’, ‘C’, ‘G’, ‘T’ bases individually for file “dna1.dat”. Save your output to “dna1-baseCount.txt”.
[2]	(2 pts) Identifying the occurrences of a fixed pattern in DNA sequence can help to build the physical gene mapping. Compose a Python program to calculate the occurrences of ‘ATGTTG’ in “dna1.dat”. Save your output to “dna1-patternCount.txt”.
[3]	(2 pts) The double counter-rotating helix structure of DNA allows sequencing a single strand and derive the reverse complement for the pairing strand. Given DNA ‘GTCCGTCCGAGGGAAATTGCGCATTCTGG’, its reverse complement is rev(‘CAGGCAGGCTCCCTTTAACGCGTAAGACC’). The complement rules are {‘A’:’T’, ‘C’:’G’, ‘G’:’C’, ‘T’:’A’}. Compose a Python program to compute the reverse complement of file “dna1.dat”. Save your output to “dna1-revComplement.txt”. (Hint: use extended slicing L[::-1] for reverse the list of L)
[4]	(1 pt) Compose a Python program to generate a random DNA sequence with 10000 bases. Save your output to “dnaOut.txt”. (Hint: Use “random.choice” and “range” functions)
[5]	1.(2 pts) DNA with low GC-content is less stable than DNA with high GC-content. It is generally believed that GC content plays a necessary role in adaptation temperatures. Compose a Python program to determine the percentage of “GC” contents (G+C/(A+T+G+C)) in “dna1.dat”. Save your output to “dna1-gcCount.txt”. (Hint: Build a gc-string then compare, or count g + count c then compare)
[6]	1.(1 pts) Given two DNA sequences of equal length, the hamming distance is the number of mismatching characters. For example, the hamming distance between ‘TCCGA’ and ‘CTGGA’ is 3. The Hamming distance between two DNA sequences plays a role in DNA molecular recognition. Compose a Python program to calculate the hamming distance between “dna1.dat” and “dna2.dat”. Save your output to “hammingOut.txt”. (Hint: Consider use “set.difference” to shorten your code)
[7]	1.(1 pt) In genetics, a sequence motif is a nucleotide or amino-acid sequence pattern that is wide-spread and has or is conjectured to have a biological significance. Compose a Python program to find the top 5 most frequent k=5 motifs in dna1.dat. Order your output and save it to “Top5Motifs.txt”.

Question 1:
Code:

```python
#%% question1

dnaIn = open("dna1.dat")
contents = dnaIn.read()
dnaIn.close()

#pattern search:
    #find number of occurences of pattern in contents

#count total number of DNA bases and totals for ACGT
countacgt = contents.count("ACGT")
counta = contents.count("A")
countc = contents.count("C")
countg = contents.count("G")
countt = contents.count("T")

with open("dna1-baseCount.txt",'w') as f:
    f.write('Base Pair count: ' + str(countacgt) + '\n')
    f.write('Total Adenine Count (A): ' + str(counta) + '\n')
    f.write('Total Cytosine Count (C): ' + str(countc) + '\n')
    f.write('Total Guanine Count (G): ' + str(countg) + '\n')
    f.write('Total Thymine Count (T) ' + str(countt) + '\n')


#%%question2
```

Output File:

dna1-baseCount.txt

```
Base Pair count: 39
Total Adenine Count (A): 2476
Total Cytosine Count (C): 2550
Total Guanine Count (G): 2486
Total Thymine Count (T) 2488
```

Question 2:
Code:

```
#%%question2
#calculate occurrences of ATGTTG in dn1.dat
countatgttg = contents.count("ATGTTG")
with open("dna1-patternCount.txt",'w') as f:
    f.write('\n\n' + 'Occurrences of "ATGTTG": ' + str(countatgttg))


#%%question3
```

Output File:

dna1-patternCount.txt

```
Occurrences of "ATGTTG": 2
```

Question 3:

Code:

```python
#%%question3
#(2 pts) The double counter-rotating helix structure of DNA allows
#sequencing a single strand and derive the reverse complement for the
#pairing strand. Given DNA 'GTCCGTCCGAGGGAAATTGCGCATTCTGG', its reverse
#complement is rev('CAGGCAGGCTCCCTTTAACGCGTAAGACC'). The complement
#rules are {'A':'T', 'C':'G', 'G':'C', 'T':'A'}. Compose a Python program
#to compute the reverse complement of file "dna1.dat". Save your output to
#"dna1-revComplement.txt". (Hint: use extended slicing L[::-1] for reverse
#the list of L)

#smalldna = "GTCCGTCCGAGGGAAATTGCGCATTCTGG"
#print(smalldna)

contents = contents.replace('\n',"")
contents = contents.replace('\t',"")

smalldna = str(contents)

def reverseddna(smalldna):
    complement = {'A':'T','C':'G','G':'C','T':'A'}
    rc = ''.join([complement[base] for base in smalldna[::-1]])
   # print(rc)
    return rc

#print("reversed complement: " + str(reverseddna(smalldna)))

rc = reverseddna(smalldna)

with open("dna1-revComplement.txt","w") as f:
    f.write("The reversed complement of the provided DNA is: " + '\n'
            + str(rc))
```

Output File:

dna1-revComplement.txt

```
The reversed complement of the provided DNA is:
GGTGACGCCACTAGCAAGCATAGGCCCTCACACGGACCGGCCGCTTGAATGGCAAAACGCACGATAGGAAGCTCAGCTACCGATGAGGGA
GCGCGGACAGGTGTCTTCCTCGCATGGGCACAGGTCAGCAGGGGCGTATGGACATGGTTGGGCCGTATTGGCCATAATGTTCTGTCCTCT
ACGTCCGGCGTCATTTTTTGGGTAGACAAGGATCGAAGAACTAGGTGTATGCCATAACGTGGCAATTTTGGTCTCTCTCCTTTGGGGAAG
GACCATCTGCAATTGTAGGCAGAAGATATACTTGCCCGGGGACATCCAGGTCATATACGTGGAAAGGTAGCCAAGCATTGAGAAGTCACT
AATCTTTAACAACTCACTTACTTGGGTGTCTACGGAAGCCCCTTTGTATAAACATAGGTAAATGTAGTGCCAGATGCCTAATTTCTATAT
AGGGTATTCCATATTGTCACTACGCCTGCGTCATTCAGATCGCCGTGCGGTGCTATCTAGTTTCAGGTCTAGCTACTTCCCAATGAAGTC
ATCATCGACCTGGCCCTTGGATTTTATGACAAAGCTAATCCTATTATACAACCACGGAACTCAGGAGTATCGCCCAGTTCCTACAGATAT
ATGTCGTCGTAGCGTGGGATTATGTTGGTTGCGTTTCATACTGAGATACCGCGGGCGCCATCACTAAATGCCTCATCTACATAGAGAGTT
CCATGTGATACACAGCTTTCTATCCAGGGCTTAAATGTAGCCGCCGCTAGATTGTACTACGCGGTAAGATTATGTCACCACCGTACGTCG
GGTGCCTCCCCTAGAGCTCAAGAGTTAGCACTCGTGCATAAACAGGCAATACGGAGCTATTCTGATGAAATTACCCGCCGAGTGTATATT
CGGGCTATGTTTTTTCAGCGTGAGACGTGTCTGCATCCTGGCCGAGATCGCATGCCCTTGCCTAGACTCCGGCGCAAGATGTGCAGTGTA
GTAGCGGCTTTACGCGATCGCGAGCGAGTCTAAAGCTCCCCCTATACTGCGCAAAATATGCGCTGGCAGAGCTTATACGGCTGACATAAG
ACCGCTAACTTTAACGTGAAACTCCGCAGTACTATCCCCCCTCTCTGAAAAAACCACGGCAAGGGGTTCATGCCGCCACAGTTGACTGGC
AACCAATGCCGTTGAATTTATCTGCTATTCAAGTTATCTCGGGAGTTGCCCTATGGTTAACCGTCGGATCCGTACAATATTCCGGATGGA
CCGCAACAATATTTCCAATCAGGTATCAATGCAAAGTGCAAGTATTGCAAGTTGTGTTGTGGCATATAGCAAGTTGTGCAAACCCTGTGA
TACCTTCCCTGCCCACACTTGTGAAAAGATGGTTGCCACCGGGACGTTTAATCGCACGGAACTATGCCATATCAACCTGATACTCACTCA
AGACGTAAGACGTACTGCACTTGGCCAGGCGCGATGAGTCCGCGTATACGGTTTAGACTCCACCGTAGGAGAGGAGGTCCTTATGTAACT
GTTATTCCCTGACTTGCGGGTGCGTGCAATTTGCTGAGATACTAGTCCGGCGAGCTCGCTGCGACGTTCTCTACGGCTGACGTGTGACAG
GCTCTGCTAGTCTGAGCACAATTACAATTCGCAATTCAGAATTGCTCTTCCATTAACATAGTGCACAGTTACTATGCCTGGTAGACGAGC
GTAGGTCCGCAGAGAGCGGGGAACCGATGCTCCTGAAGGGGACGGACCGGTCGGGTCTTGGTGCCGGCGAAAAGACATTAGCTAGACATC
GCATCCCAATGAACTTTACTCGGTTGCTTGGCAGATGCTTTGCTTGTGGACGGGTAGGACGCACTTGCAAGCCAGCCCAGCTGTGGTCTA
GCACTGACACCGGACTATTGTCATTGTCAGCAAACCTGGCTTTTCTATAACTTCAGCGGCACTACAGCGCAGTGCCAAATACGCACGCAA
AGGCTAGTAGCCTTGTTGTGCACCTTTAGACTTGCCAAAGAAGTTTGCGCAAGGTCACAGTGCATCCCACTACGCCGTAGTGCGACCTTA
GTCACTCTAACTTCCCCGACTCTGGACACGAATCCTCTGGTCGAGTGATCCCCAGGCCTAGGTTACGAGCGGCCGCATCTGAGAAAGCGG
CAATTCGCCTATAACTAACTAGCCAAGTAGTAAAAGGACAGCTTCAGAGTGATCGGTGTTCGCCAGCATACTTAAATTACGCCAAGCTGT
AGACGACAATCTAAGCAGTTCTCCCATCGCAGAACTTCTGGAGGCGGTAGCCGCACGGGGTGAGTTGGCCCAGTACTGCGTTTTCGGGTC
CTCTCTCGATGTGGATAAATATGTCTATACCCCTGAGGGAATGATGATCAGTAATTTCCCGCCCCCACGGGCCTGCGGAGTTATCCGATG
ATACAACCGGTAACTCGACTTGCACACGGCCTGCATCATAGACGGTGGAATTCTTGCGTTAGCACCCATCGAACACAGTGCATTGATACA
TTAATTCTGCTTTCGGGCCTGTAGCAACCTAGATTGTGATTCGATGCTAAATGGGGTGGGGGTACCTTTCACCATATGGTTTCTGGTATG
```

Question 4:
Code:

```python
#%% question 4
#compose a program to generate a random DNA sequence with 10,000 bases
#save your output to "dnaOut.txt" ((use random.choice and range functions))

#example: generate 10000 random DNA bases #question #4?

choice = ''.join([random.choice('ACGT') for dna in range(10000)])

with open("dnaOut.txt",'w') as f:
    f.write("A randomly generated DNA sequence of 10000 bases:\n" + str(choice))



#%% question 5
```

Output File:

dnaOut.txt

```
A randomly generated DNA sequence of 10000 bases:
GTAATGCACGCCACACGGCAGCCTATGCATGGAGGTTACTTGCATTCTTGCGTGCCAGGCTGAAGGGTAACATTAACCTTGC
CAGGCGATATTGCCCAAACGTGCTCGTCTTATTGTGAATGAGTTATCGCCCTCGGCCAGAAATGATCCGAGCAGAGGCTACG
TTATGACTTCGACATCCTCGGACGACCATGGATCTGGTCTACACACCCAACGGCCCAGCAAAAGCTGTGTAGAATGTGATAC
AGTCCGTTAAGTATCGCTATGCTAGGTTGGCGGCAATCACGATGTCTCTCCTGCGGCACCACCAGTATTCCGGTGAGTTTGG
GCACTAGTGGGTCAGCACGTTTATTCGTTCCATATATTTATGGAAACTCAGCGTAGGGGGGCACGAACTGAGGTGATGGATG
GGACTCAACCCGTCACGGTTCGTGGGCGGAGGCCCGACAGGAGTGCTTTACGAGTGCCGACAGCTTTCAAGTCAAACCAAAC
TAATAATCGTTACTATCTGACGCTTAACTTCTGAGCCGCCATGCAGAAGCGGTAACAGGGTATGTGTCCTTCCGAACACATC
ATCATTGCCCCAGTAATGCCGCGGGGGAACCGGCAGAATGAACTCTGTTAGCATCGCACCAACTACAGACCTCTATGTTTCT
CGGTACGGCGCGTTTTACTCAACAGATGCGGCCGCGTACTCCGTCTGCAACCCCTCCATTGTGCCTACCACTCCACTTGGTA
ATGTATCCCTGCACTCCTACTTTGGGCGTTCGTAGGGCATACGCTCTTAGCGTTCACAGCTTTTCGGCCGGGAATACTATAA
GGTTCAGTCAACGCGCCCGGCGGGATTTCAGGGAGCCAGCGCCTTCAGCTTGAACAGGACAGTTCAGCTCAGCCGTATTGCC
CTATACCGATCGGGGGAACGGGACGTACGGGCCCACACACCAGGCAGCACAATTCATTACGGCTCGGAGAATTCCGAAGGCT
ATGCGGAACTTCACCAGTCGCGAGGTCGAAATCTCACCCGGATAGAAAGGCCGCGCACGTGAACGGAACACACAGGAGCTTC
TTGACTGACCGATGTTAGTTGTCAGCCACGGCATCCCGAGTAATCTCAACTACGAAAGCTCGAGAGGTCAATCGTGATAGTT
CCCACGTGCATGCTTGACGCGTTCTTGTATCTCATAAGCTAGGCTCATAACGATTTGTGATCTTAACATTGTCGCCTTTCTT
TACCCTAATAGACCTGGGGGTTTGAACTTAGGAGTTAGTGTTGTACAACGTCGATGCGCCTCTCACTAGCAGGTCTCTCTAA
TGGTTTAAGGGTGCAAGCCACAAGTATGACGCCCTTTCGGAGAACCCTTGTCCTTAAAAAAGAAGGAGGTCGACACTTGGGC
ATAGGGTACTGAAGTGCGTTGTCGGAGTCTTCACAGGCCTGAGCAACGGCAAAAAGAGAACGGCATGGGGCGACGTTCGTGC
GGACGCAGAGAATGTGTTAGTACTAATCTAGGATATCGCACATAATATCAATGGTTTAATCGTGCTCACCGCTGCGATAAAC
GTCTCCTGCCACGTACGCCTGGTCTGCAATGGGAGGAAACTAACCCTGTTTTATTACAGATGCTATGTAAGATGAGGCGGGC
TAGGCTTACAAATGCATGGTGTTCTATGATTCTACGATTGCAACCAGAGTTTTCCACGCTCAAGAACTATATCTACAGAAAG
GGATCTAGGGTGGGCACATGATTGCTCAACGTCCACAAACAAGTCGGTTCTCCCTTTCGTGTCACGTTGGTATCCTATTGTT
CTCTTCACACGTGGTGCCCGTTCCAAGAGTCACGGACTATTGGGAGCCCCTCTGATCAGTCATAGCTGGTTAGTTTAAGCCG
```

Question 5:
Code:

```python
#%% question 5
#DNA with low GC-content is less stable than DNA with high GC-content.
#It is generally believed that GC content plays a necessary role in
#adaptation temperatures. Compose a Python program to determine the
#percentage of “GC” contents (G+C/(A+T+G+C)) in “dna1.dat”. Save your output
#to “dna1-gcCount.txt”. (Hint: Build a gc-string then compare, or count g +
#count c then compare)

#values were just taken from earlier counts
gval = countg
cval = countc
tval = countt
aval = counta

gval,cval = int(countg),int(countc)
tval,aval = int(countt),int(counta)

gctot = gval+cval
acgttot = gval+cval+tval+aval
gv_percent = gctot/acgttot
print(gv_percent)

with open("dna1-gcCount.txt",'w') as f:
    f.write('Total Adenine Count (A): ' + str(counta) + '\n')
    f.write('Total Cytosine Count (C): ' + str(countc) + '\n')
    f.write('Total Guanine Count (G): ' + str(countg) + '\n')
    f.write('Total Thymine Count (T): ' + str(countt) + '\n\n')
    f.write('Total "GC" Count (GC): ' + str(gctot) + '\n')
    f.write('Total Base Count (ACGT): ' + str(acgttot) +'\n')
    f.write('Percent of "GC" content in DNA1: ' + str(gv_percent))
```

Output File:

dna1-gcCount.txt

```
Total Adenine Count (A): 2476
Total Cytosine Count (C): 2550
Total Guanine Count (G): 2486
Total Thymine Count (T): 2488

Total "GC" Count (GC): 5036
Total Base Count (ACGT): 10000
Percent of "GC" content in DNA1: 0.5036
```

Question 6:
Code:

```python
#%%question 6
#Given two DNA sequences of equal length, the hamming distance is the
#number of mismatching characters. For example, the hamming distance between
#'TCCGA' and 'CTGGA' is 3. The Hamming distance between two DNA sequences
#plays a role in DNA molecular recognition. Compose a Python program to
#calculate the hamming distance between "dna1.dat" and "dna2.dat". Save your
#output to "hammingOut.txt". (Hint: Consider use "set.difference" to shorten
#your code)

dna1_content = open("dna1.dat")
dna1 = dna1_content.read()
dna1_content.close()

dna2_content = open('dna2.dat')
dna2 = dna2_content.read()
dna2_content.close()

# dna1 = 'TCCGA'
# dna2 = 'CTGGA'

#we want to loop through this probably
#and up the count as we compare

countval = 0

if len(dna1) != len(dna2):
    print("DNA strands not the same length~")
else:
    for index,(i,j) in enumerate(zip(dna1,dna2)):
        if i!=j:
            countval+=1
    print(countval)

with open("hammingOut.txt",'w') as f:
    f.write('Total Hamming Distance: ' + str(countval) + '\n')
```

Output File:

hammingOut.txt

```
Total Hamming Distance: 7559
```

Question 7:
Code:

```python
#%%question 7
#In genetics, a sequence motif is a nucleotide or amino-acid sequence
#pattern that is wide-spread and has or is conjectured to have a biological
#significance. Compose a Python program to find the top 5 most frequent k=5
#motifs in dna1.dat. Order your output and save it to “Top5Motifs.txt”.

k = 5

dna = 'GTCCGTCCGAGGGAATTGCGATTCTGG'

# kmers = defaultdict(int)

# for i in range(len(dna) - k + 1):
#     motif = dna[i:i+k]
#     kmers[motif] += 1

# all = sorted(kmers.items(), key=operator.itemgetter(1))

# print (all[-5:]) #print the last 5


motifs = Counter([dna[i:i+k] for i in range(len(dna)-k+1)])
print(motifs.most_common(5))

with open("Top5Motifs.txt",'w') as f:
    f.write('Top 5 Most Frequent Motifs:\n')
    f.write(str(motifs.most_common(5)))

#actually really cool question because i didnt know this was a thing
```

Output File:

Top5Motifs.txt

```
Top 5 Most Frequent Motifs:
[('GTCCG', 2), ('TCCGT', 1), ('CCGTC', 1), ('CGTCC', 1), ('TCCGA', 1)]
```

Debugging Issues:

My primary issues with this lab were not with understanding code & the challenges they presented but with the actual background concepts in the lab. I had never heard of Hamming Distance and I had to read deeper into complementary DNA sequences in order to properly translate the given DNA strands.

Overall the lab was great with little to know debugging issues. I had to follow the powerpoint//lecture for question 7. I read through it and watched the video a couple times but settled on the most efficient method to find the motifs.