Name: Yash Kesharwani **Roll No: 3**

Class: MSC CS Part 1 **Subject: Bioinformatics**

Academic Year: 2021-2022

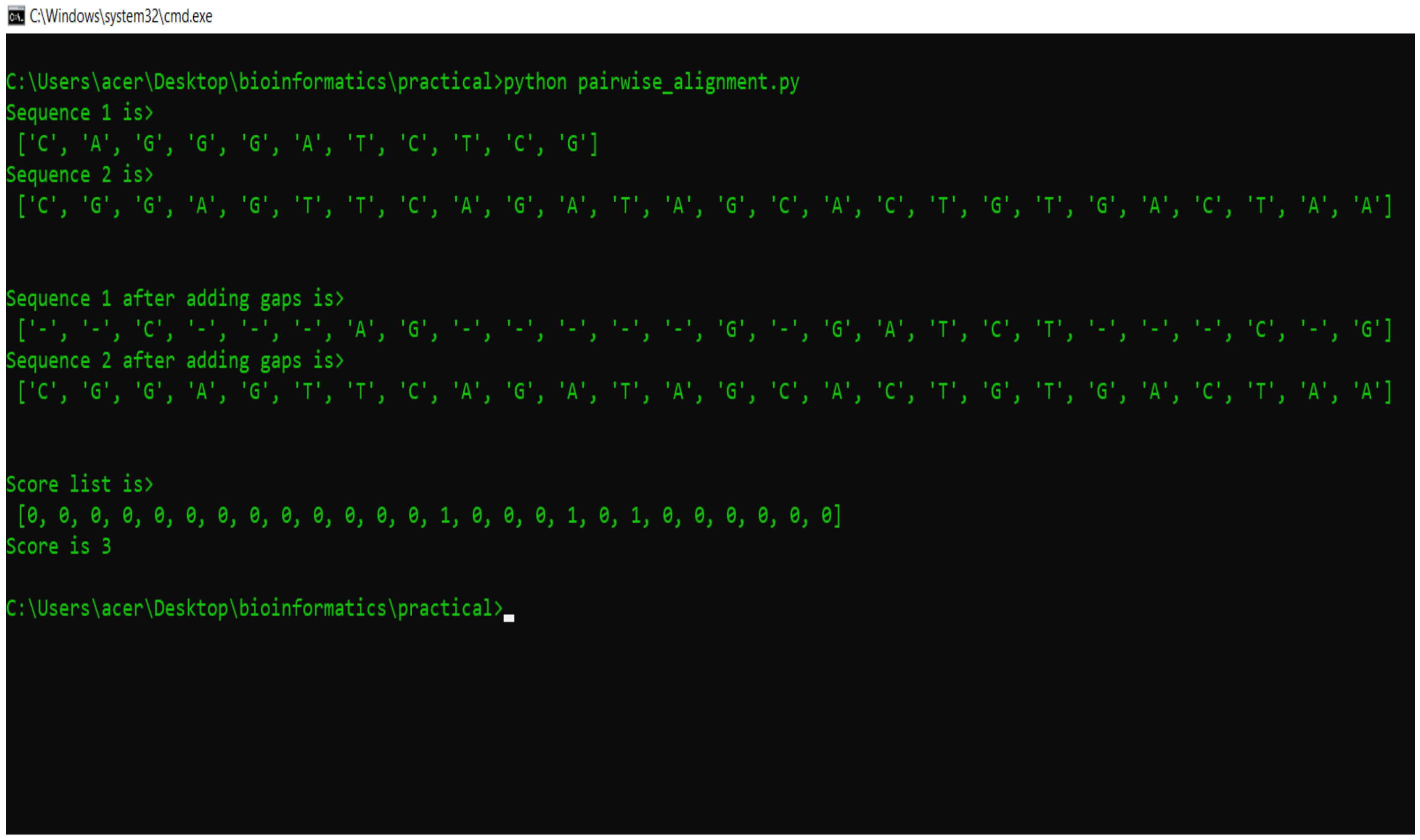
Index

Practical No: 1

Aim: Write a Python/Java code to perform pairwise alignment. Take 2 sequences from user and calculate the score.

Code:
```
from random import choice, randint
from operator import eq

def get_sequences():
    char_sequence = 'ACTG'
    sequence_1 = [choice(char_sequence) for i in range(randint(10, 50))]
    sequence_2 = [choice(char_sequence) for i in range(randint(10, 50))]

    return sequence_1, sequence_2



def insert_gap(sequence):
    sequence.insert(randint(0, len(sequence) - 1), '-')

    return sequence


def insert_gaps(sequence_1 , sequence_2 ):
    while len(sequence_1) != len(sequence_2):
        if len(sequence_1) < len(sequence_2):
            sequence_1 = insert_gap(sequence_1)
        else:
            sequence_2 = insert_gap(sequence_2)

    return sequence_1, sequence_2

def pairwise_alignment(sequence_1, sequence_2):
    return list(map(eq, sequence_1, sequence_2))

if __name__ == "__main__":
    sequence_1, sequence_2 = get_sequences()
    print("Sequence 1 is>\n", sequence_1)
    print("Sequence 2 is>\n", sequence_2)
    print("\n")

    sequence_1, sequence_2 = insert_gaps(sequence_1, sequence_2)
    print("Sequence 1 after adding gaps is>\n", sequence_1)
    print("Sequence 2 after adding gaps is>\n", sequence_2)
    print("\n")

    score_list = pairwise_alignment(sequence_1, sequence_2)
    print("Score list is>\n", [1 if i else 0 for i in score_list])
    print(f"Score is {sum(score_list)}")
```

Output:



```
C:\Windows\system32\cmd.exe

C:\Users\acer\Desktop\bioinformatics\practical>python pairwise_alignment.py
Sequence 1 is>
 ['C', 'A', 'G', 'G', 'G', 'A', 'T', 'C', 'T', 'C', 'G']
Sequence 2 is>
 ['C', 'G', 'G', 'A', 'G', 'T', 'T', 'C', 'A', 'G', 'A', 'T', 'A', 'G', 'C', 'A', 'C', 'T', 'G', 'T', 'G', 'A', 'C', 'T', 'A', 'A']


Sequence 1 after adding gaps is>
 ['-', '-', 'C', '-', '-', '-', 'A', 'G', '-', '-', '-', '-', '-', 'G', '-', 'G', 'A', 'T', 'C', 'T', '-', '-', '-', 'C', '-', 'G']
Sequence 2 after adding gaps is>
 ['C', 'G', 'G', 'A', 'G', 'T', 'T', 'C', 'A', 'G', 'A', 'T', 'A', 'G', 'C', 'A', 'C', 'T', 'G', 'T', 'G', 'A', 'C', 'T', 'A', 'A']


Score list is>
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]
Score is 3

C:\Users\acer\Desktop\bioinformatics\practical>
```

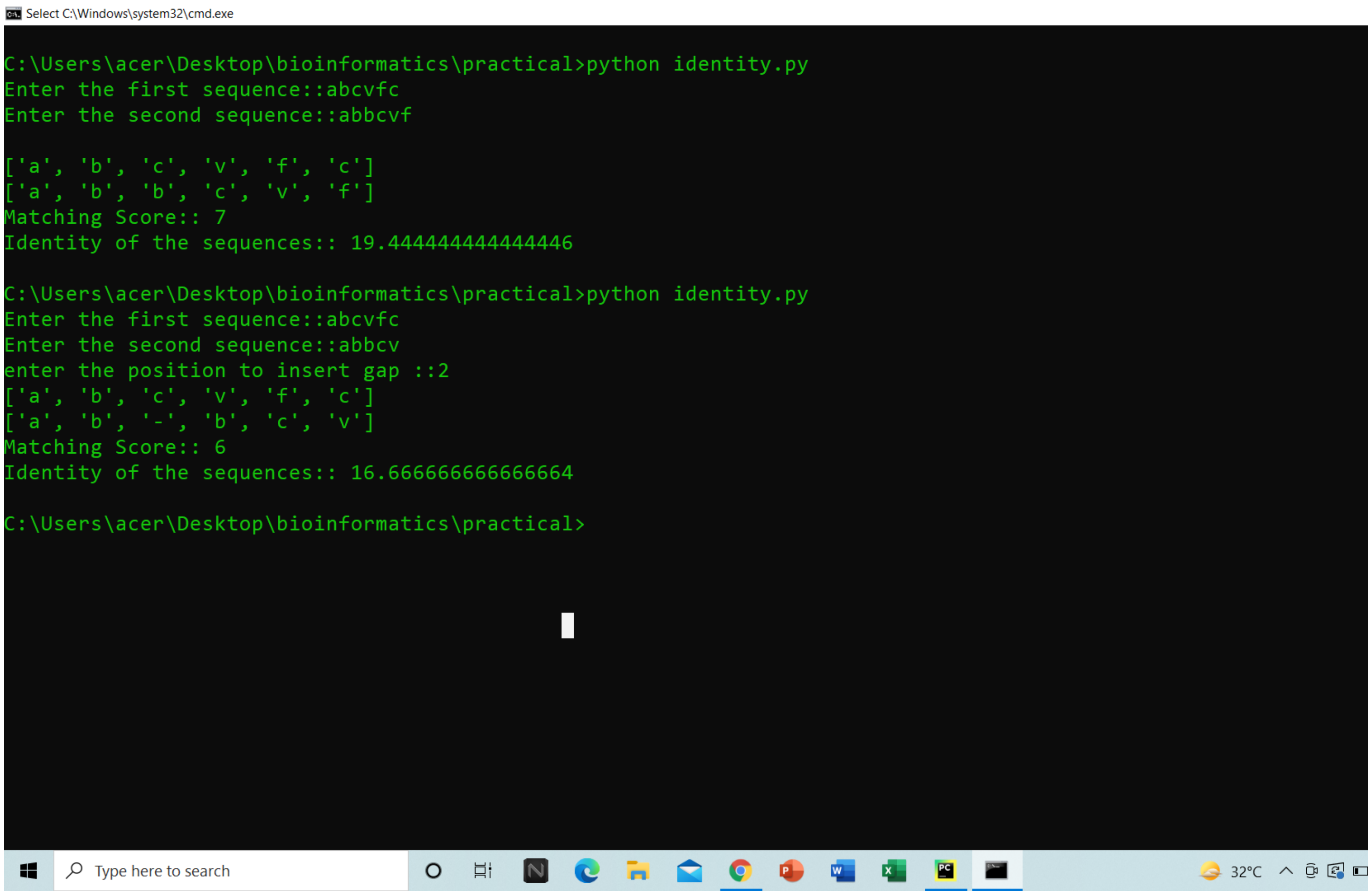
Practical No: 2

Aim: Write a Python/Java code to find the identity value of a given sequences. Take the sequence from user.

Code:

```
se1=input("Enter the first sequence::")
se2=input("Enter the second sequence::")
seq1=list(se1)
seq2=list(se2)
def find_identity(a,b):
    gap(a,b)
    print(a)
    print(b)
    score=0
    length=len(a)
    total_elements=len(a)*len(b)
    for i in range(0,length):
        for j in range(0,length):
            if(a[i]==b[j]):
                score=score+1
    identity=(score/total_elements)*100
    print("Matching Score::",score)
    print("Identity of the sequences::",identity)

def gap(a,b):
    if(len(a)==len(b)):
        print()
    else:
        k=int(input("enter the position to insert gap ::"))
        if (len(a)<len(b)):
            a.insert(k,'-')
        else:
            b.insert(k,'-')
    return(a,b)


if __name__ == "__main__":
    find_identity(seq1, seq2)
```

Output:



Select C:\Windows\system32\cmd.exe

```
C:\Users\acer\Desktop\bioinformatics\practical>python identity.py
Enter the first sequence::abcvfc
Enter the second sequence::abbcvf

['a', 'b', 'c', 'v', 'f', 'c']
['a', 'b', 'b', 'c', 'v', 'f']
Matching Score:: 7
Identity of the sequences:: 19.444444444444446

C:\Users\acer\Desktop\bioinformatics\practical>python identity.py
Enter the first sequence::abcvfc
Enter the second sequence::abbcv
enter the position to insert gap ::2
['a', 'b', 'c', 'v', 'f', 'c']
['a', 'b', '-', 'b', 'c', 'v']
Matching Score:: 6
Identity of the sequences:: 16.666666666666664

C:\Users\acer\Desktop\bioinformatics\practical>
```

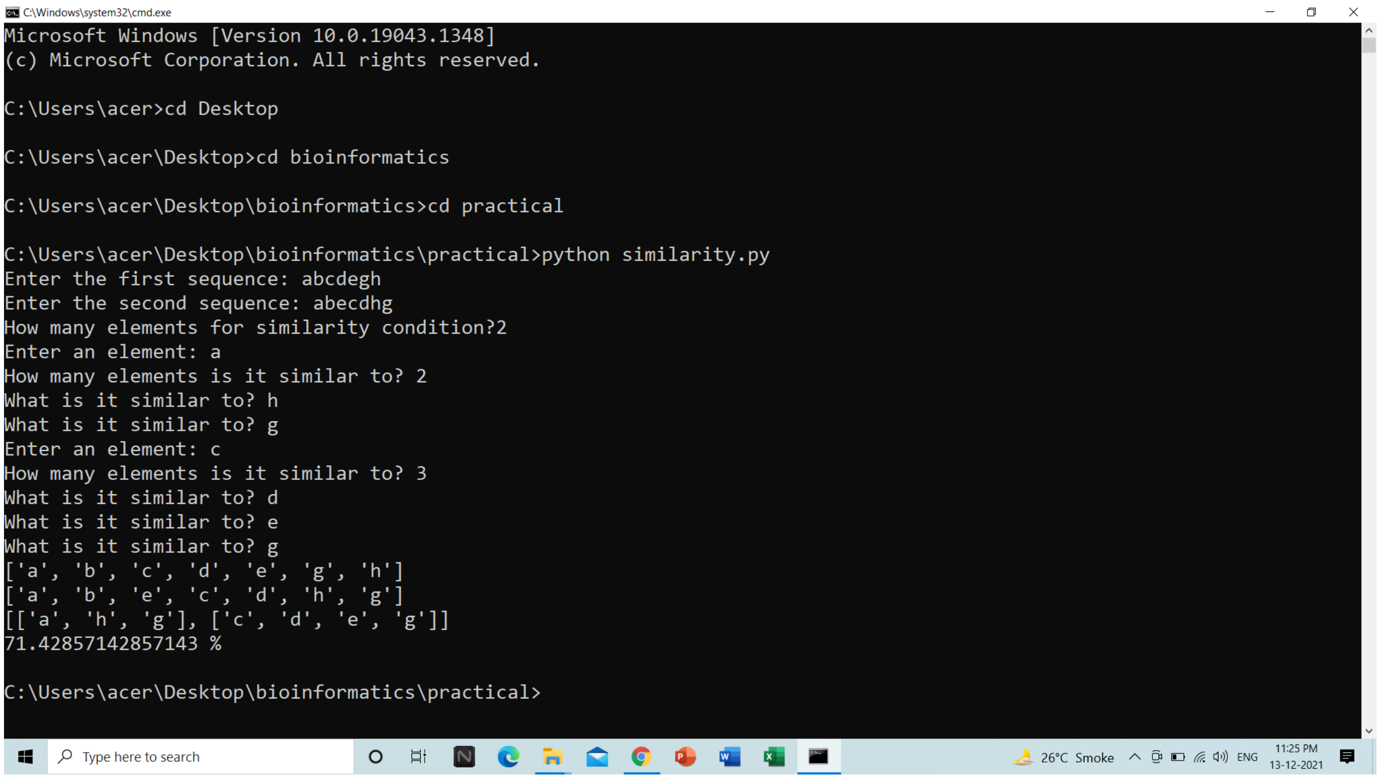
Practical No: 3

Aim: Write a Python/Java code to find the Similarity value of a given sequences. Take the sequence from user.

Code:

```
sequence_one=input("Enter the first sequence: ")
sequence_two=input("Enter the second sequence: ")
how_many=int(input("How many elements for similarity condition?"))
similarities=[]
for i in range(0,how_many):
    a=input("Enter an element: ")
    c=int(input("How many elements is it similar to? "))
    similarities.append([])
    similarities[i].append(a)

    for j in range(0,c):
        b=input("What is it similar to? ")
        similarities[i].append(b)
def compare(o,t,s):
  print(o)
  print(t)
  print(s)

  score=0
  for i in range(len(o)):
    for j in range(len(s)):
       if o[i] in s[j] and t[i] in s[j] and o[i] != t[i]:
           score+=1

  similarity= (score*100)/len(o)
  return similarity
print(compare(list(sequence_one),list(sequence_two),similarities),"%")
```

Output:


C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\acer>cd Desktop

C:\Users\acer\Desktop>cd bioinformatics

C:\Users\acer\Desktop\bioinformatics>cd practical

C:\Users\acer\Desktop\bioinformatics\practical>python similarity.py
Enter the first sequence: abcdegh
Enter the second sequence: abecdhg
How many elements for similarity condition?2
Enter an element: a
How many elements is it similar to? 2
What is it similar to? h
What is it similar to? g
Enter an element: c
How many elements is it similar to? 3
What is it similar to? d
What is it similar to? e
What is it similar to? g
['a', 'b', 'c', 'd', 'e', 'g', 'h']
['a', 'b', 'e', 'c', 'd', 'h', 'g']
[['a', 'h', 'g'], ['c', 'd', 'e', 'g']]
71.42857142857143 %

C:\Users\acer\Desktop\bioinformatics\practical>
```

Practical No: 4

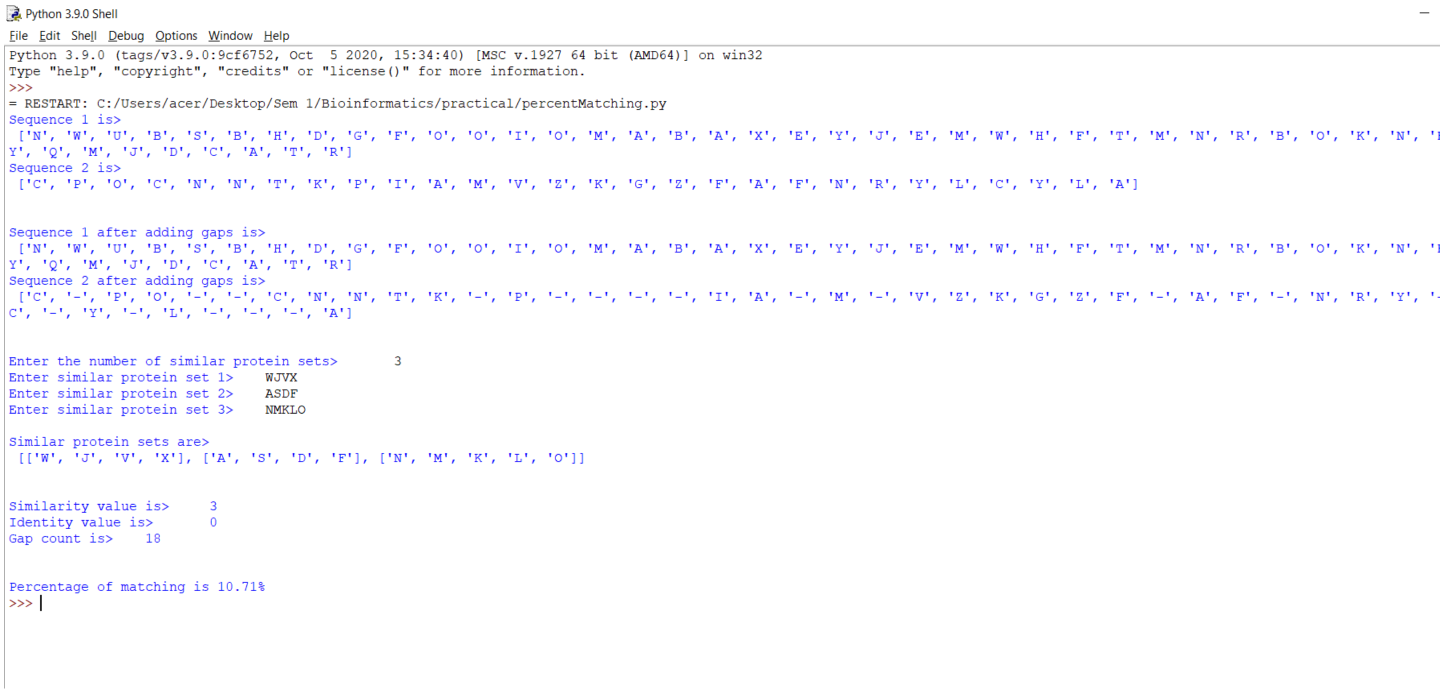
Aim: Write a Python3/Java program to calculate percentage of matching of two sequences..

Code:
```
from random import choice, randint
from string import ascii_uppercase
from operator import eq

sequence_list = []

def get_sequences() -> tuple[list[str]]:

    sequence_1 = [choice(ascii_uppercase) for i in range(randint(8, 50))]
    sequence_2 = [choice(ascii_uppercase) for i in range(randint(8, 50))]

    return sequence_1, sequence_2


def insert_gap(sequence : list[str]) -> list[str]:
    sequence.insert(randint(0, len(sequence) - 1), '-')

    return sequence


def insert_gaps(sequence_1 : list[str], sequence_2 : list[str]) -> tuple[list[str]]:
    while len(sequence_1) != len(sequence_2):
        if len(sequence_1) < len(sequence_2):
            sequence_1 = insert_gap(sequence_1)
        else:
            sequence_2 = insert_gap(sequence_2)

    return sequence_1, sequence_2


def get_similar_protein_set():
    sequence_count = int(input("Enter the number of similar protein sets>\t"))

    global sequence_list
    for i in range(sequence_count):
        sequence_list.append(list(input(f"Enter similar protein set {i + 1}>\t")))
```

```
def check_similarity(char_1 : str, char_2 : str) -> bool:
    global sequence_list
    for i in sequence_list:
        if (char_1 != char_2):
            if char_1 in i and char_2 in i:
                return True

    return False


def similarity(sequence_1 : list[str], sequence_2 : list[str]) -> int:
    similarity_list = [1 if i else 0 for i in list(map(check_similarity, sequence_1, sequence_2))]
    similarity_value = sum(similarity_list)

    return similarity_value


def identity(sequence_1 : list[str], sequence_2 : list[str]) -> int:

    return sum(map(eq, sequence_1, sequence_2))


def count_gaps(sequence_1 : list[str], sequence_2 : list[str]) -> int:
    return sequence_1.count("-") + sequence_2.count("-")


if __name__ == "__main__":
    sequence_1, sequence_2 = get_sequences()
    print("Sequence 1 is>\n", sequence_1)
    print("Sequence 2 is>\n", sequence_2)
    print("\n")

    sequence_1, sequence_2 = insert_gaps(sequence_1, sequence_2)
    print("Sequence 1 after adding gaps is>\n", sequence_1)
    print("Sequence 2 after adding gaps is>\n", sequence_2)
    print("\n")

    get_similar_protein_set()
    print("\nSimilar protein sets are>\n", sequence_list)
    print("\n")

    similarity_value = similarity(sequence_1, sequence_2)
    identity_value = identity(sequence_1, sequence_2)
    gap_count = count_gaps(sequence_1, sequence_2)
```

```
        print("Similarity value is>\t", similarity_value)
        print("Identity value is>\t", identity_value)
        print("Gap count is>\t", gap_count)
        print("\n")

        print(f"Percentage of matching is {round(((similarity_value + identity_value) /
(len(sequence_1) - gap_count)) * 100, 2)}%")
```

Output:



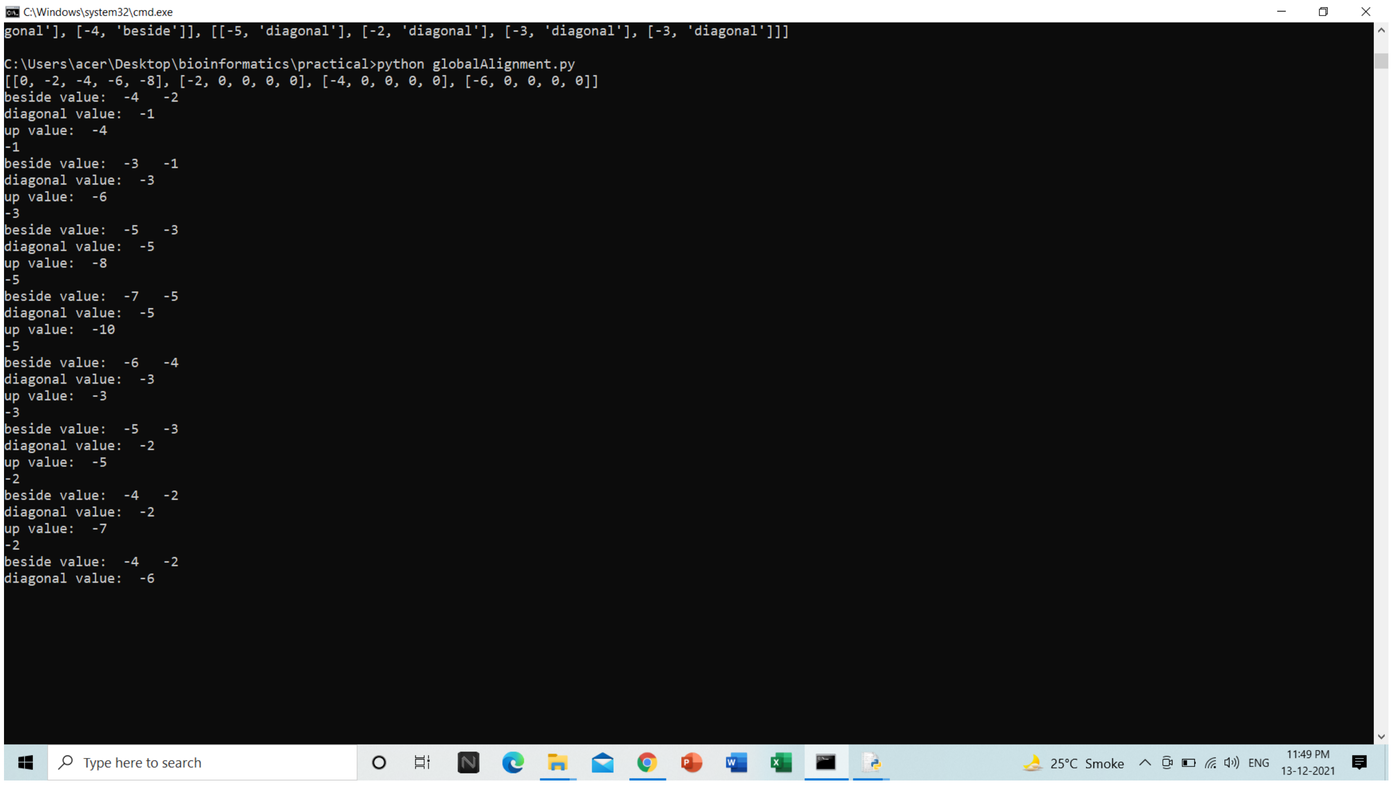
```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/acer/Desktop/Sem 1/Bioinformatics/practical/percentMatching.py
Sequence 1 is>
 ['N', 'W', 'U', 'B', 'S', 'B', 'H', 'D', 'G', 'F', 'O', 'O', 'I', 'O', 'M', 'A', 'B', 'A', 'X', 'E', 'Y', 'J', 'E', 'M', 'W', 'H', 'F', 'T', 'M', 'N', 'R', 'B', 'O', 'K', 'N', '
Y', 'Q', 'M', 'J', 'D', 'C', 'A', 'T', 'R']
Sequence 2 is>
 ['C', 'P', 'O', 'C', 'N', 'N', 'T', 'K', 'P', 'I', 'A', 'M', 'V', 'Z', 'K', 'G', 'Z', 'F', 'A', 'F', 'N', 'R', 'Y', 'L', 'C', 'Y', 'L', 'A']


Sequence 1 after adding gaps is>
 ['N', 'W', 'U', 'B', 'S', 'B', 'H', 'D', 'G', 'F', 'O', 'O', 'I', 'O', 'M', 'A', 'B', 'A', 'X', 'E', 'Y', 'J', 'E', 'M', 'W', 'H', 'F', 'T', 'M', 'N', 'R', 'B', 'O', 'K', 'N', '
Y', 'Q', 'M', 'J', 'D', 'C', 'A', 'T', 'R']
Sequence 2 after adding gaps is>
 ['C', '-', 'P', 'O', '-', '-', 'C', 'N', 'N', 'T', 'K', '-', 'P', '-', '-', '-', '-', 'I', 'A', '-', 'M', '-', 'V', 'Z', 'K', 'G', 'Z', 'F', '-', 'A', 'F', '-', 'N', 'R', 'Y', '
C', '-', 'Y', '-', 'L', '-', '-', '-', 'A']


Enter the number of similar protein sets>       3
Enter similar protein set 1>    WJVX
Enter similar protein set 2>    ASDF
Enter similar protein set 3>    NMKLO

Similar protein sets are>
 [['W', 'J', 'V', 'X'], ['A', 'S', 'D', 'F'], ['N', 'M', 'K', 'L', 'O']]


Similarity value is>     3
Identity value is>       0
Gap count is>    18


Percentage of matching is 10.71%
>>> 
```

Practical 5

Aim: Write a Python3/Java program to generate a scoring matrix for global alignment of a pair of sequences.

```
class GlobalAlignment:
    gap = -2
    mismatch = -1
    match = 1

    def __init__(self, seq1, seq2):
        self.s1 = seq1
        self.s2 = seq2

    def __repr__(self):
        return f"Sequence 1 : {self.s1}\nSequence 2: {self.s2}"

    def matrix(self):
        matrix = []
        start = 0
        anotherstart = -2
        for i in range(len(self.s2) + 1):
            submatrix = []
            for j in range(len(self.s1) + 1):
                if i == 0:
                    submatrix.append(start)
                    start -= 2
                elif j == 0 and i != 0:
                    submatrix.append(anotherstart)
                    anotherstart -= 2
                else:
                    submatrix.append(0)
            matrix.append(submatrix)
        return matrix

    @staticmethod
    def beside(value):
        return value + GlobalAlignment.gap

    @staticmethod
    def up(value):
        return value + GlobalAlignment.gap

    def align(self):
        alignMatrix = []
        matrix = self.matrix()
        print(matrix)
        for i in range(1, len(matrix)):
            submatrix = []
            for j in range(1, len(matrix[i])):
                besides = GlobalAlignment.beside(matrix[i][j - 1])  # Beside value
                print('beside value: ', besides, " ", matrix[i][j - 1])
                dg = matrix[i - 1][j - 1]
                if self.s1[j - 1] == self.s2[i - 1]:
                    diagonal = dg + GlobalAlignment.match  # Beside value on match
                else:
                    diagonal = dg + GlobalAlignment.mismatch  # Diagonal value o mismatch
                print('diagonal value: ', diagonal)

                up = GlobalAlignment.up(matrix[i - 1][j])  # Up value
                print('up value: ', up)
                check = [besides, diagonal, up]
                maximum = max(check)
                print(maximum)
                ind = check.index(maximum)
                if ind == 0:
                    path = 'beside'
                elif ind == 1:
                    path = 'diagonal'
                elif ind == 2:
                    path == 'up'
                submatrix.append([maximum, path])
                matrix[i][j] = maximum
            alignMatrix.append(submatrix)
        print(alignMatrix)
        print("Matrix is ", matrix)
        return [matrix, alignMatrix]

    def traceback(self):
        originalmatrix = self.align()[0]
        tracingmatrix = self.align()[1]
        print("\nInside traceback\n",originalmatrix,"\n\n",tracingmatrix)


gb = GlobalAlignment('ATCG', 'GCT')
gb.traceback()
```

Output:



```
gonal'], [-4, 'beside']], [[-5, 'diagonal'], [-2, 'diagonal'], [-3, 'diagonal'], [-3, 'diagonal']]]

C:\Users\acer\Desktop\bioinformatics\practical>python globalAlignment.py
[[0, -2, -4, -6, -8], [-2, 0, 0, 0, 0], [-4, 0, 0, 0, 0], [-6, 0, 0, 0, 0]]
beside value:  -4   -2
diagonal value:  -1
up value:  -4
-1
beside value:  -3   -1
diagonal value:  -3
up value:  -6
-3
beside value:  -5   -3
diagonal value:  -5
up value:  -8
-5
beside value:  -7   -5
diagonal value:  -5
up value:  -10
-5
beside value:  -6   -4
diagonal value:  -3
up value:  -3
-3
beside value:  -5   -3
diagonal value:  -2
up value:  -5
-2
beside value:  -4   -2
diagonal value:  -2
up value:  -7
-2
beside value:  -4   -2
diagonal value:  -6
```

Practical 6

Aim: Write a Python3/Java program to perform multiple sequence alignment.

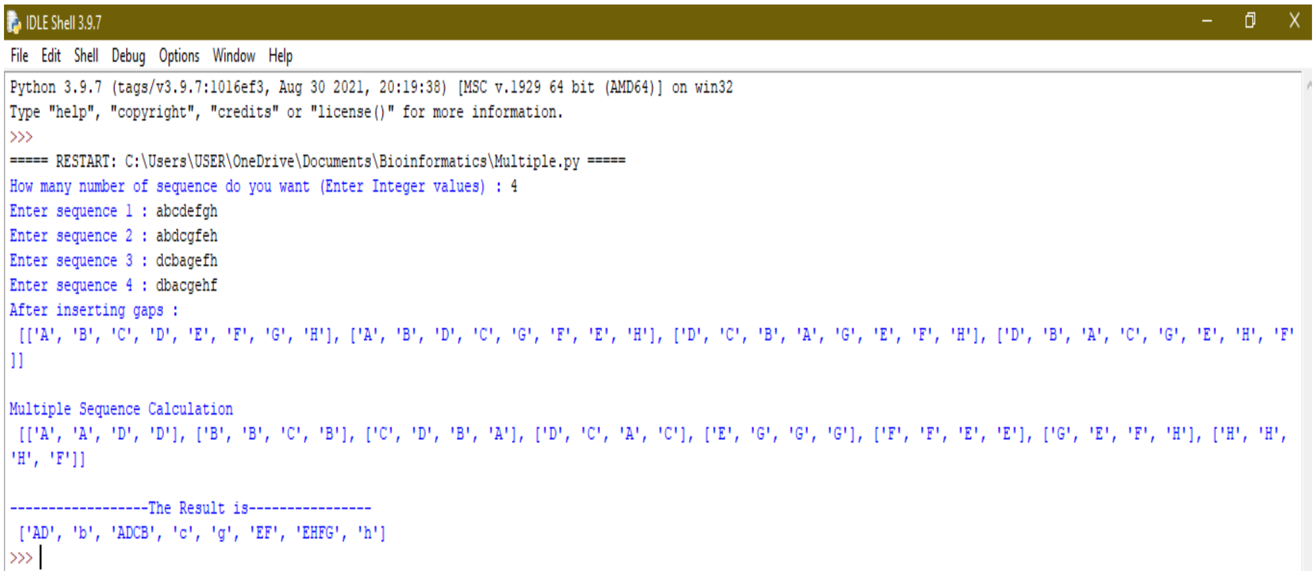
Code:

```
import random


class MultipleSeq:
    def __init__(self):
        try:
            howmany = int(input("How many number of sequence do you want (Enter Integer
values) : "))
        except Exception as e:
            print('\n\nError : ', e)
        self.sequenceList = []
        self.length = []
        for i in range(1, howmany + 1):
            seq = input(f'Enter sequence {i} : ').upper()  # Example : 'AOSLIE'
            self.length.append(len(seq))
            self.sequenceList.append([i for i in seq])

    def gap(self):  # Insert gap
        maxlength = max(self.length)
        aftergapseq = []
        for i in self.sequenceList:
            if len(i) < maxlength:
                difflength = maxlength - len(i)
                for gapper in range(difflength):
                    r = random.randrange(0, len(i), 1)
                    i.insert(r, '-')
                aftergapseq.append(i)
            else:
                aftergapseq.append(i)
        print("After inserting gaps : \n", aftergapseq)
        return aftergapseq

    def multipleSequenceAlignment(self):
        sequence = self.gap()
        seq = []
        for i in range(len(sequence[0])):
            innerseq = []
            for j in sequence:
```

```
        innerseq.append(j[i])
    seq.append(innerseq)
print("\nMultiple Sequence Calculation\n", seq)
resulter = []
maxer = []
for i in seq:
    innerresult = []
    innermax = []
    for j in i:
        if j != '-':
            counter = i.count(j)
            innermax.append(counter)
            tur = (j, counter)
            innerresult.append(tur)
    resulter.append(innerresult)
    maxer.append(innermax)
result = []

for i in range(len(maxer)):
    maximum = max(maxer[i])
    ir = set()
    for j in resulter[i]:
        if (maximum == j[1]):
            ir.add(j[0])
    result.append(ir)
maxerset = []
for i in maxer:
    setter = set()
    for j in i:
        setter.add(j)
    maxerset.append(setter)

lastlist = []
for i in range(len(maxerset)):
    stp = ''
    if (len(maxerset[i]) > 1):
        for k in result[i]:
            stp = stp + k.lower()
    else:
        for k in result[i]:
            stp = stp + k.upper()
    lastlist.append(stp)
print("\n------------------The Result is----------------\n", lastlist)
```

```
myseq = MultipleSeq()
myseq.multipleSequenceAlignment()
```

Output:



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\USER\OneDrive\Documents\Bioinformatics\Multiple.py =====
How many number of sequence do you want (Enter Integer values) : 4
Enter sequence 1 : abcdefgh
Enter sequence 2 : abdcgfeh
Enter sequence 3 : dcbagefh
Enter sequence 4 : dbacgehf
After inserting gaps :
 [['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'], ['A', 'B', 'D', 'C', 'G', 'F', 'E', 'H'], ['D', 'C', 'B', 'A', 'G', 'E', 'F', 'H'], ['D', 'B', 'A', 'C', 'G', 'E', 'H', 'F'
]]

Multiple Sequence Calculation
 [['A', 'A', 'D', 'D'], ['B', 'B', 'C', 'B'], ['C', 'D', 'B', 'A'], ['D', 'C', 'A', 'C'], ['E', 'G', 'G', 'G'], ['F', 'F', 'E', 'E'], ['G', 'E', 'F', 'H'], ['H', 'H',
'H', 'F']]

------------------The Result is----------------
 ['AD', 'b', 'ADCB', 'c', 'g', 'EF', 'EHFG', 'h']
>>> 
```

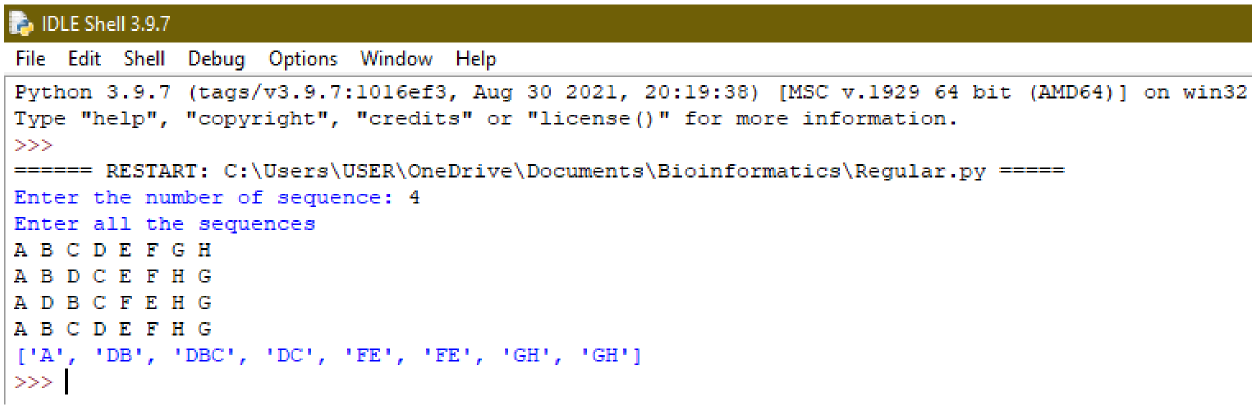
Practical 7

Aim: Write a Python3/Java program to find the regular expression from a set of sequences.

Code:

```
def gen_reg_exp(seq_list, no_of_col):
    final_list=[]
    for colnum in range(no_of_col):
        collist=[]
        for colseq in seq_list:
            collist.append(colseq[colnum])
        if len(set(collist))==len(collist):

            final_list.append('x')
        else:
            if len(set(collist))==1:
                final_list.append(collist[0])
            else:
                final_list.append(''.join(set(collist)))
    display_output(final_list)

def display_output(final_list):
    print(*final_list, sep='-')



if  __name__ == '__main__':

    no_of_seq = int(input("Enter the number of sequence: "))
    print("Enter all the sequences")
    seq_list = []
    for _ in range(no_of_seq):
        seq_list.append(list(map(str, input("").split())))
    gen_reg_exp(seq_list, len(seq_list[0]))
```

Output:



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\USER\OneDrive\Documents\Bioinformatics\Regular.py =====
Enter the number of sequence: 4
Enter all the sequences
A B C D E F G H
A B D C E F H G
A D B C F E H G
A B C D E F H G
['A', 'DB', 'DBC', 'DC', 'FE', 'FE', 'GH', 'GH']
>>>
```

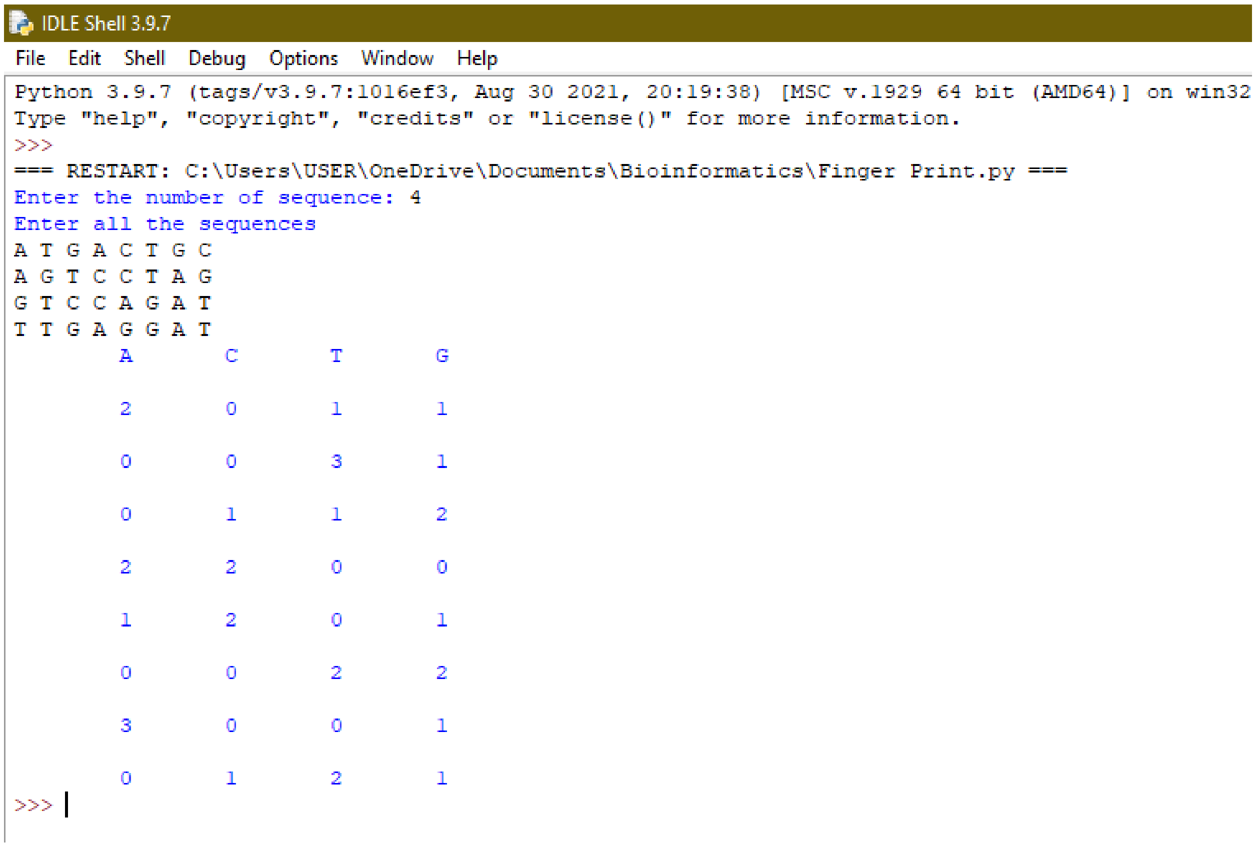
Practical 8

Aim: Write a Python3/Java program to find the fingerprint of the sequence.

Code:

```
def solve_fingerprint(seq_list, no_of_col):
  seq_dict=dict()
  for colnum in range(no_of_col):
    counta,countc,countt,countg=0,0,0,0
    for colseq in seq_list:
      if colseq[colnum]=='A':
        counta+=1
      elif colseq[colnum]=='T':
        countt+=1
      elif colseq[colnum]=='C':
        countc+=1
      elif colseq[colnum]=='G':
        countg+=1
    seq_dict[colnum]=[counta,countc,countt,countg]
  display_results(seq_dict)
def display_results(seq_dict):
  print("\tA \tC \tT \tG")
  for key in seq_dict:
    print("\n",*seq_dict[key],sep="\t")
no_of_seq=int(input("Enter the number of sequence: "))
print("Enter all the sequences")
seq_list=[]
for _ in range(no_of_seq):
  seq_list.append(list(map(str, input("").split())))
solve_fingerprint(seq_list,len(seq_list[0]))
```

Output:



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\USER\OneDrive\Documents\Bioinformatics\Finger Print.py ===
Enter the number of sequence: 4
Enter all the sequences
A T G A C T G C
A G T C C T A G
G T C C A G A T
T T G A G G A T
        A       C       T       G

        2       0       1       1

        0       0       3       1

        0       1       1       2

        2       2       0       0

        1       2       0       1

        0       0       2       2

        3       0       0       1

        0       1       2       1
>>> 
```

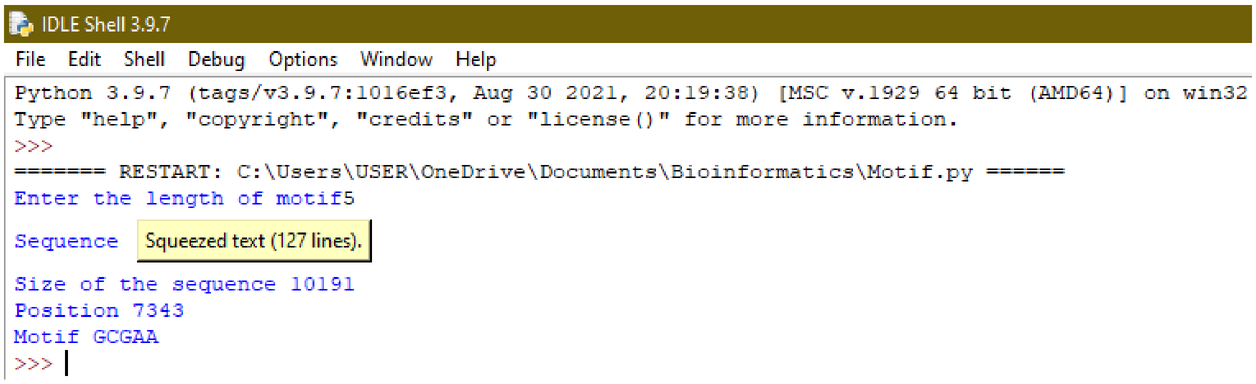
Practical 9

Aim: Write a Python3/Java program to find the motif of the sequence.

Code:

```
import random
l=int(input("Enter the length of motif"))
file=open("seqdump.txt","r")
r=file.read()
print("Sequence",r)
size=len(r)
print("Size of the sequence",size)
pos=random.randint(0,len(r)-5)

print("Position",pos)
motif=r[pos:pos+l]
print("Motif",motif)
i=pos+1
while(i<=size-1):
  if(motif==r[i:i+1]):
    str1=r[i:i+1]
    print("Match motif",str1)
    file1=open("motoutput.txt","a")
    file1.write(str1+" ")
  i+=1
```

Output:



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Users\USER\OneDrive\Documents\Bioinformatics\Motif.py ======
Enter the length of motif5

Sequence Squeezed text (127 lines).

Size of the sequence 10191
Position 7343
Motif GCGAA
>>>
```

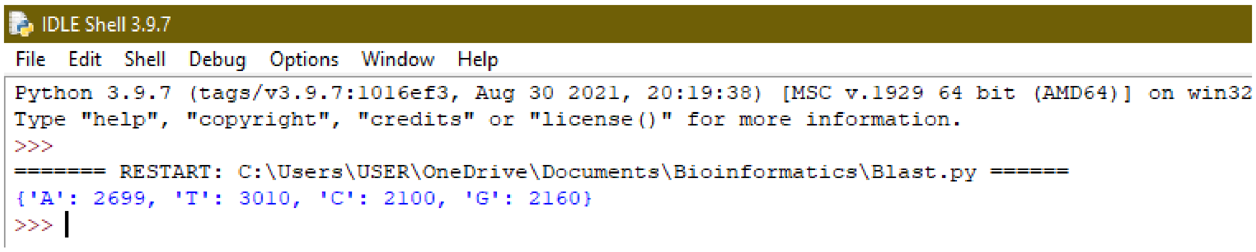
Practical 10

Aim: Write a Python3/Java program to perform BLAST search and find the no of repetition of each nucleotide in the sequence.

Code:

```
fasta=open('seqdump.txt','r')
seq=fasta.read()
data={'A':seq.count('A'),'T':seq.count('T'),'C':seq.count('C'),'G':seq.count('G')}
print(data)
```

Output:



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Users\USER\OneDrive\Documents\Bioinformatics\Blast.py ======
{'A': 2699, 'T': 3010, 'C': 2100, 'G': 2160}
>>>
```