Name: Yash Rajiv Walke **Roll no.: 15**

Class: MSc CS Part I **Subject: Bioinformatics**

Academic Year: 2021-2022

Index

Sr. No	Practical	Page No.	Signature
1.	Pairwise Alignment	2	
2.	Find Identity of Sequence	5	
3.	Similarity of Sequence	8	
4.	Percentage matching of sequence	12	
5.	Global Alignment (Scoring matrix)	18	
6.	Multiple Sequence Alignment	24	
7.	Find Regular Expression of given sequences.	29	
8.	Enter six sequence of different organism and write a program to find a fingerprint of sequence.	33	
9.	Motif Finding	38	
10.	Perform BLAST search and Find the no of repetition of each nucleotide in the sequence.	41	

Practical 1

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

Code:

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


def get_sequences() → tuple[list[str]]:
    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 : 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 pairwise_alignment(sequence_1 : list[str],
sequence_2 : list[str]) → list[str]:
    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:\Temp\Bioinformatics>python pairwise_alignment.py
Sequence 1 is>
 ['C', 'C', 'T', 'G', 'G', 'G', 'A', 'C', 'C', 'C', 'G', 'G', 'C', 'T', 'T', 'C', 'A', 'G', 'T', 'A', '
T', 'C', 'A', 'G', 'C', 'T', 'G', 'T', 'T', 'T', 'C', 'C', 'G', 'C', 'G', 'A', 'C', 'T', 'A', 'T', 'T',
 'C', 'G', 'C', 'T', 'T', 'A', 'A', 'C']
Sequence 2 is>
 ['A', 'T', 'T', 'A', 'T', 'G', 'G', 'A', 'T', 'C', 'C', 'T', 'G', 'G', 'G', 'T', 'A', 'A', 'A', 'G', '
C', 'T', 'T', 'C', 'C', 'G', 'G', 'T', 'A', 'G', 'T', 'G', 'A', 'A', 'C', 'A', 'A']


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


Score list is>
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
Score is 13
```

Practical 2

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

Code:

```
from random import choice, randint


def get_sequences() → tuple[list[str]]:
    char_sequence = 'ACTG'
    sequence_length = randint(10, 20)
    sequence_1 = [choice(char_sequence) for i in
range(sequence_length)]
    sequence_2 = [choice(char_sequence) for i in
range(sequence_length)]


    return sequence_1, sequence_2


def identity(sequence_1 : list[str], sequence_2 :
list[str]) → tuple[int, list[list[int]]]:


    result_matrix = [[1 if i == j else 0 for j in
sequence_1] for i in sequence_2]
```

```
    result = sum([sum(i) for i in result_matrix])

    return result, result_matrix

def print_matrix(matrix : list[list[int]]):
    for i in matrix:
        print(i)
    print()



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")

    result, result_matrix = identity(sequence_1,
sequence_2)
    print("Result matrix is>\n")
    print_matrix(result_matrix)
    print(f"Identity is {round((result /
(len(sequence_1) * len(sequence_2))) * 100, 2)}")
```

Output:

```
C:\Temp\Bioinformatics>python identity.py
Sequence 1 is>
 ['A', 'A', 'C', 'A', 'C', 'A', 'C', 'C', 'C', 'A', 'T', 'A']
Sequence 2 is>
 ['T', 'A', 'A', 'C', 'T', 'A', 'G', 'C', 'G', 'A', 'A', 'C']


Result matrix is>

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0]

Identity is 32.64
```

Practical 3

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

Code:

```
from random import choice, randint

from string import ascii_uppercase


sequence_list = []


def get_sequences() → tuple[list[str]]:

    sequence_length = randint(8, 50)

    sequence_1 = [choice(ascii_uppercase) for i in
range(sequence_length)]

    sequence_2 = [choice(ascii_uppercase) for i in
range(sequence_length)]


    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]) → tuple[int, list[str]]:
```

```
    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, similarity_list



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")

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

    similarity_value, similarity_list =
similarity(sequence_1, sequence_2)
    print("Similarity list is>\n", similarity_list)
    print(f"Similarity is {round((similarity_value /
len(sequence_1)) * 100, 2)}%")
```

Output:

```
C:\Temp\Bioinformatics>python similarity.py
Sequence 1 is>
 ['B', 'T', 'P', 'P', 'H', 'G', 'H', 'N', 'T', 'Z', 'I', 'J', 'G', 'E', 'K', 'Z', 'S', 'T', 'X', 'M', '
C', 'S', 'I', 'R', 'E', 'Y', 'Z', 'J', 'O', 'J', 'E', 'R', 'L']
Sequence 2 is>
 ['C', 'Y', 'D', 'O', 'X', 'T', 'O', 'G', 'E', 'P', 'W', 'O', 'T', 'C', 'C', 'Q', 'X', 'C', 'Q', 'Z', '
F', 'K', 'D', 'J', 'L', 'B', 'H', 'G', 'R', 'D', 'B', 'P', 'N']


Enter the number of similar protein sets>       5
Enter similar protein set 1>    QPBRY
Enter similar protein set 2>    WYT
Enter similar protein set 3>    PX
Enter similar protein set 4>    SBEXC
Enter similar protein set 5>    GA
Similar protein sets are>
 [['Q', 'P', 'B', 'R', 'Y'], ['W', 'Y', 'T'], ['P', 'X'], ['S', 'B', 'E', 'X', 'C'], ['G', 'A']]


Similarity list is>
 [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0]
Similarity is 21.21%
```

Practical 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:

```
C:\Temp\Bioinformatics>python percent_matching.py
Sequence 1 is>
 ['H', 'B', 'Z', 'E', 'N', 'M', 'E', 'O', 'N', 'R', 'E', 'J', 'F', 'C', 'N', 'R', 'F', 'A']
Sequence 2 is>
 ['B', 'B', 'Q', 'F', 'G', 'X', 'J', 'X', 'S', 'P', 'B', 'Z', 'D', 'W', 'N', 'X', 'Z', 'E', 'T', 'Y', '
C', 'U']


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


Enter the number of similar protein sets>       5
Enter similar protein set 1>    PX
Enter similar protein set 2>    GA
Enter similar protein set 3>    WYT
Enter similar protein set 4>    SBEXC
Enter similar protein set 5>    QPBRY

Similar protein sets are>
 [['P', 'X'], ['G', 'A'], ['W', 'Y', 'T'], ['S', 'B', 'E', 'X', 'C'], ['Q', 'P', 'B', 'R', 'Y']]
```

```
Similarity value is>     3
Identity value is>       1
Gap count is>    4


Percentage of matching is 22.22%
```

Practical 5

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

Code:

```
from random import randint, choice

from math import inf


def generate_sequences() → tuple[list[str]]:

    sequence_1 = [choice(('A', 'C', 'T', 'G')) for i in
range(randint(6, 7))]

    sequence_2 = [choice(('A', 'C', 'T', 'G')) for i in
range(randint(6, 7))]


    return sequence_1, sequence_2



class GlobalAlignment:


    @staticmethod
```

```
    def generate_scoring_matrix(sequence_1:list[str],
sequence_2:list[str]) → list[list[list[int,
list[str]]]]:
        sequence_1.insert(0, '-')
        sequence_2.insert(0, '-')
        result_matrix = [[[0] for i in
range(len(sequence_2))] for j in
range(len(sequence_1))]


        for i in range(1, len(sequence_1)):
            result_matrix[i][0][0] = result_matrix[i -
1][0][0] - 2
            result_matrix[i][0].append(["up"])


        for j in range(1, len(sequence_2)):
            result_matrix[0][j][0] = result_matrix[0]
[j - 1][0] - 2
            result_matrix[0][j].append(["besides"])


        for i in range(1, len(sequence_1)):
            for j in range(1, len(sequence_2)):
                besides_value = result_matrix[i][j -
1][0] - 2
```

```
                up_value = result_matrix[i - 1][j][0]
- 2
                diagonal_value = result_matrix[i - 1]
[j - 1][0]

                additive_value = None
                direction = []

                if(sequence_1[i] == '-') or
(sequence_2[j] == '-'):
                    additive_value = - 2
                elif (sequence_1[i] != sequence_2[j]):
                    additive_value = - 1
                elif (sequence_1[i] == sequence_2[j]):
                    additive_value = 1

                diagonal_value += additive_value

                largest_value = besides_value
                direction.append("besides")

                if (largest_value < up_value):
```

```
                    largest_value = up_value
                    direction[0] = "up"

                elif (largest_value == up_value):
                    direction.append("up")

                if (largest_value < diagonal_value):
                    largest_value = diagonal_value
                    direction[0] = "diagonal"
                    if len(direction) > 1:
                        direction.pop()

                elif (largest_value ==
diagonal_value):
                    largest_value = diagonal_value
                    direction.append("diagonal")



                result_matrix[i][j][0] = largest_value
                result_matrix[i][j].append(direction)

        return result_matrix
```

```
    @staticmethod
    def
print_scoring_matrix(result_matrix:list[list[list[int,
list[str]]]]):
        for i in range(len(result_matrix)):
            print(result_matrix[i])
            print("\n")



if __name__ == "__main__":
    sequence_1, sequence_2 = generate_sequences()

    print("Sequence 1 is:\t", sequence_1)
    print("Sequence 2 is:\t", sequence_2)

    result_matrix =
GlobalAlignment.generate_scoring_matrix(sequence_1,
sequence_2)
    print("Scoring matrix is:")
    GlobalAlignment.print_scoring_matrix(result_matrix)
```

Output:

```
C:\Temp\Bioinformatics>python global_alignment.py
Sequence 1 is:   ['A', 'C', 'C', 'C', 'T', 'T']
Sequence 2 is:   ['C', 'A', 'T', 'G', 'G', 'C']
Scoring matrix is:
[[0], [-2, ['besides']], [-4, ['besides']], [-6, ['besides']], [-8, ['besides']], [-10, ['besides']], [
-12, ['besides']]]


[[-2, ['up']], [-1, ['diagonal']], [-1, ['diagonal']], [-3, ['besides']], [-5, ['besides']], [-7, ['bes
ides']], [-9, ['besides']]]


[[-4, ['up']], [-1, ['diagonal']], [-2, ['diagonal']], [-2, ['diagonal']], [-4, ['besides', 'diagonal']
], [-6, ['besides', 'diagonal']], [-6, ['diagonal']]]


[[-6, ['up']], [-3, ['up', 'diagonal']], [-2, ['diagonal']], [-3, ['diagonal']], [-3, ['diagonal']], [-
5, ['besides', 'diagonal']], [-5, ['diagonal']]]


[[-8, ['up']], [-5, ['up', 'diagonal']], [-4, ['up', 'diagonal']], [-3, ['diagonal']], [-4, ['diagonal'
]], [-4, ['diagonal']], [-4, ['diagonal']]]


[[-10, ['up']], [-7, ['up']], [-6, ['up', 'diagonal']], [-3, ['diagonal']], [-4, ['diagonal']], [-5, ['
diagonal']], [-5, ['diagonal']]]
```

```
[[-12, ['up']], [-9, ['up']], [-8, ['up', 'diagonal']], [-5, ['up', 'diagonal']], [-4, ['diagonal']], [
-5, ['diagonal']], [-6, ['diagonal']]]
```

Practical 6

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

Code:

```
from random import choice, randint


def get_sequences(no_of_sequences : int) →
list[list[str]]:
    sequence_list = []
    print("Enter the sequences (All sequences should
have equal length)>\t")


    for i in range(no_of_sequences):
        sequence_list.append(list(input(f"Enter
sequence {i + 1}>\t")))


    return sequence_list



def get_random_sequences(no_of_sequences : int) →
list[list[str]]:
```

```
    sequence_length = randint(8, 20)
    sequence_list = [[choice('ABCDE') for j in
range(sequence_length)] for i in
range(no_of_sequences)]
    return sequence_list



def multiple_sequence_alignment(sequence_list :
list[list[str]]) → list[str]:
    output_sequence = []

    for i in range(len(sequence_list[0])):
        char_list = list()

        for j in range(len(sequence_list)):
            char_list.append(sequence_list[j][i])

        char_set = set(char_list)
        char_at_pos = ""

        if len(char_set) == 1:
            char_at_pos = char_set[0]
```

```
        elif len(sequence_list) % len(char_set) == 0:
            for i in char_set:
                char_at_pos += f"{i}/"
            char_at_pos = char_at_pos[: -1]

        else:
            largest_count = 0
            largest_char = None

            for i in char_set:
                if char_list.count(i) ⩾
largest_count:
                    largest_char = i
                    largest_count = char_list.count(i)

            char_at_pos = largest_char.lower()

        output_sequence.append(char_at_pos)

    return output_sequence
```

```
if __name__ == "__main__":
    print("Multiple sequence alignment in Python 3.6+")
    no_of_sequences = int(input("Enter the number of
input sequences>\t"))


    random_flag = False if input("Do you want the
sequences to be randomly generated? [Yes]/No>\
t").lower() == "no" else True
    sequence_list =
get_random_sequences(no_of_sequences) if random_flag
else get_sequences(no_of_sequences)


    print("Sequences are as follows:")
    for i in range(len(sequence_list)):
        print(f"Sequence {i + 1}>\t", sequence_list[i])


    print("Multiple sequence alignment for given
sequences is:\t",
multiple_sequence_alignment(sequence_list))
```

Output:

```
C:\Temp\Bioinformatics>python multiple_sequence_alignment.py
Multiple sequence alignment in Python 3.6+
Enter the number of input sequences>    6
Do you want the sequences to be randomly generated? [Yes]/No>
Sequences are as follows:
Sequence 1>     ['D', 'B', 'D', 'C', 'E', 'E', 'D', 'B', 'E', 'A', 'C']
Sequence 2>     ['C', 'D', 'B', 'A', 'D', 'A', 'D', 'B', 'E', 'B', 'E']
Sequence 3>     ['E', 'A', 'D', 'D', 'D', 'E', 'C', 'C', 'D', 'D', 'E']
Sequence 4>     ['D', 'D', 'C', 'D', 'C', 'D', 'D', 'B', 'C', 'B', 'B']
Sequence 5>     ['B', 'D', 'C', 'C', 'B', 'D', 'C', 'C', 'A', 'D', 'A']
Sequence 6>     ['A', 'A', 'E', 'B', 'B', 'B', 'D', 'E', 'B', 'C', 'B']
Multiple sequence alignment for given sequences is:     ['d', 'B/A/D', 'd', 'd', 'd', 'd', 'C/D', 'B/C
/E', 'e', 'd', 'e']
```

Practical 7

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

Code:

```
from random import choice, randint

from string import ascii_uppercase


def get_sequences(no_of_sequences : int) →
list[list[str]]:

    sequence_list = []

    print("Enter the sequences (All sequences should
have equal length)>\t")


    for i in range(no_of_sequences):

        sequence_list.append(list(input(f"Enter
sequence {i + 1}>\t")))


    return sequence_list
```

```
def get_random_sequences(no_of_sequences : int) →
list[list[str]]:
    sequence_length = randint(8, 20)
    sequence_list = [[choice(ascii_uppercase[: 6]) for
j in range(sequence_length)] for i in
range(no_of_sequences)]
    return sequence_list



def get_regular_expression(sequence_list :
list[list[str]]) → list[str]:
    output_sequence = []

    for i in range(len(sequence_list[0])):
        char_column = set()

        for j in range(len(sequence_list)):
            if sequence_list[j][i] ≠ '-':
                char_column.add(sequence_list[j][i])

        char_at_i = ""

        if len(char_column) == 1:
```

```
            char_at_i = char_column[0]


        else:
            if len(char_column) == len(sequence_list):
                char_at_i = 'X'
            else:
                char_at_i += "["
                for i in char_column:
                    char_at_i += i
                char_at_i += "]"


        output_sequence.append(char_at_i)


    return output_sequence


if __name__ == "__main__":
    print("Regular Expression in Python 3.6+")
    no_of_sequences = int(input("Enter the number of
input sequences>\t"))
```

```
    random_flag = False if input("Do you want the
sequences to be randomly generated? [Yes]/No>\
t").lower() == "no" else True

    sequence_list =
get_random_sequences(no_of_sequences) if random_flag
else get_sequences(no_of_sequences)


    print("Sequences are as follows:")

    for i in range(len(sequence_list)):

        print(f"Sequence {i + 1}>\t", sequence_list[i])


    print("Regular expression for given sequences is:\
t", get_regular_expression(sequence_list))
```

Output:

```
C:\Temp\Bioinformatics>python regular_expression.py
Regular Expression in Python 3.6+
Enter the number of input sequences>    6
Do you want the sequences to be randomly generated? [Yes]/No>
Sequences are as follows:
Sequence 1>     ['C', 'A', 'C', 'D', 'D', 'F', 'A', 'B', 'B', 'B', 'C', 'B', 'A', 'E', 'B', 'F', 'A',
'F', 'C']
Sequence 2>     ['F', 'B', 'F', 'B', 'A', 'B', 'E', 'B', 'F', 'B', 'A', 'F', 'F', 'D', 'F', 'C', 'C',
'D', 'A']
Sequence 3>     ['F', 'A', 'F', 'E', 'F', 'D', 'B', 'B', 'B', 'B', 'D', 'B', 'A', 'E', 'D', 'B', 'F',
'E', 'D']
Sequence 4>     ['A', 'E', 'D', 'A', 'B', 'B', 'F', 'E', 'F', 'F', 'A', 'F', 'C', 'B', 'D', 'E', 'B',
'C', 'E']
Sequence 5>     ['A', 'A', 'E', 'D', 'D', 'F', 'C', 'D', 'D', 'C', 'F', 'C', 'E', 'A', 'F', 'E', 'C',
'A', 'D']
Sequence 6>     ['B', 'B', 'E', 'F', 'E', 'E', 'C', 'A', 'C', 'D', 'C', 'E', 'A', 'D', 'F', 'A', 'E',
'B', 'B']
Regular expression for given sequences is:      ['[BCAF]', '[BEA]', '[DECF]', '[FDEAB]', '[DAEFB]', '[
DBEF]', '[AECFB]', '[ABED]', '[DBCF]', '[DBCF]', '[DFCA]', '[BCEF]', '[EFCA]', '[BEAD]', '[DBF]', '[FEC
AB]', '[AECFB]', 'X', '[DECAB]']
```

Practical 8

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

Code:

```
from random import choice, randint

import sys


def generate_sequences() → list[list[str]]:

    sequence_length = randint(8, 20)

    sequence_count = randint(8, 20)


    return [[choice("ACTG") for i in
range(sequence_length)] for j in range(sequence_count)]




def
calculate_fingerprint(sequence_list:list[list[str]]) →
list[dict[str, int]]:

    fingerprint_list = []
```

```
    for j in range(len(sequence_list[0])):
        finger_print_dict = dict()

        for i in range(len(sequence_list)):
            finger_print_dict[sequence_list[i][j]] =
finger_print_dict.get(sequence_list[i][j], 0) + 1

        for i in 'ACTG':
            if i not in finger_print_dict:
                finger_print_dict[i] = 0

        fingerprint_list.append(finger_print_dict)

    return fingerprint_list



def print_result(fingerprint_list:list[dict[str,
int]]):

    print("+-------+-------+-------+-------+-------+")
    print("|Col\t|A\t|C\t|G\t|T\t|")
    print("+-------+-------+-------+-------+-------+")
```

```
    for i in range(len(fingerprint_list)):
        print(f"|{i + 1}\t|{fingerprint_list[i]['A']}\
t|{fingerprint_list[i]['C']}\t|{fingerprint_list[i]
['G']}\t|{fingerprint_list[i]['T']}\t|")

    print("+-------+-------+-------+-------+-------+")


if __name__ == "__main__":
    sequence_list = generate_sequences()

    for i in range(len(sequence_list)):
        print(f"Sequence {i + 1} is>\n",
sequence_list[i], "\n")

    fingerprint_list =
calculate_fingerprint(sequence_list)
    print_result(fingerprint_list)
```

Output:

```
C:\Temp\Bioinformatics>python fingerprint.py
Sequence 1 is>
 ['T', 'T', 'A', 'G', 'A', 'G', 'A', 'C', 'G', 'A', 'C', 'C', 'T', 'A']

Sequence 2 is>
 ['C', 'C', 'C', 'A', 'C', 'G', 'T', 'C', 'A', 'T', 'T', 'A', 'C', 'G']

Sequence 3 is>
 ['G', 'C', 'G', 'A', 'C', 'C', 'A', 'T', 'A', 'T', 'G', 'A', 'T', 'T']

Sequence 4 is>
 ['T', 'G', 'G', 'C', 'G', 'C', 'C', 'C', 'G', 'G', 'G', 'T', 'C', 'T']

Sequence 5 is>
 ['A', 'A', 'T', 'C', 'C', 'C', 'T', 'G', 'A', 'T', 'C', 'T', 'G', 'A']

Sequence 6 is>
 ['G', 'T', 'C', 'G', 'G', 'G', 'C', 'T', 'C', 'T', 'A', 'T', 'C', 'T']

Sequence 7 is>
 ['C', 'A', 'A', 'A', 'G', 'A', 'T', 'C', 'A', 'T', 'T', 'G', 'T', 'C']

Sequence 8 is>
 ['G', 'G', 'G', 'T', 'G', 'T', 'C', 'T', 'T', 'G', 'A', 'A', 'C', 'T']

Sequence 9 is>
 ['T', 'G', 'C', 'T', 'C', 'G', 'A', 'A', 'A', 'G', 'G', 'A', 'G', 'A']
```

```
Sequence 10 is>
 ['A', 'A', 'T', 'C', 'A', 'C', 'C', 'A', 'A', 'G', 'A', 'C', 'A', 'T']

Sequence 11 is>
 ['G', 'A', 'T', 'G', 'C', 'C', 'G', 'C', 'C', 'T', 'C', 'G', 'G', 'T']

Sequence 12 is>
 ['A', 'T', 'G', 'G', 'C', 'C', 'A', 'A', 'C', 'G', 'G', 'C', 'T', 'G']

Sequence 13 is>
 ['T', 'G', 'G', 'A', 'C', 'T', 'T', 'A', 'T', 'G', 'C', 'C', 'C', 'G']

Sequence 14 is>
 ['T', 'A', 'G', 'A', 'G', 'C', 'A', 'G', 'T', 'C', 'A', 'A', 'G', 'G']

Sequence 15 is>
 ['A', 'G', 'T', 'A', 'C', 'T', 'G', 'C', 'C', 'A', 'G', 'G', 'A', 'G']

Sequence 16 is>
 ['G', 'C', 'A', 'C', 'A', 'G', 'C', 'C', 'C', 'A', 'T', 'T', 'G', 'G']

Sequence 17 is>
 ['G', 'G', 'T', 'C', 'T', 'A', 'A', 'A', 'T', 'G', 'C', 'G', 'G', 'A']

Sequence 18 is>
 ['T', 'A', 'C', 'C', 'T', 'A', 'A', 'C', 'A', 'C', 'T', 'G', 'A', 'G']
```

Col	A	C	G	T
1	4	2	6	6
2	6	3	6	3
3	3	4	6	5
4	6	6	4	2
5	3	8	5	2
6	3	7	5	3
7	7	5	2	4
8	5	8	2	3
9	7	5	2	4
10	3	2	7	6
11	4	5	5	4
12	5	4	5	4
13	3	5	6	4
14	4	1	7	6

Practical 9

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

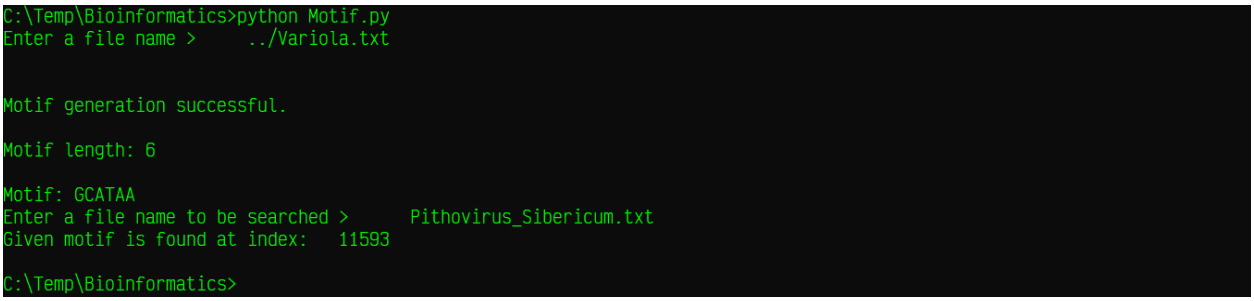
Code:

```
from random import randint


def motif(input_file_name : str) → str:
    with open(input_file_name) as input_file_handle:
        input_file_data = input_file_handle.read()
        input_file_handle.close()

    input_file_data = input_file_data.replace("\n", "")

    motif_length = randint(2, len(input_file_data) - 1)
    start_index = randint(0, len(input_file_data) -
motif_length)

    return input_file_data[start_index : start_index +
motif_length]
```

```
def search_for_motif(motif : str, search_file_name :
str) → int:
    with open(search_file_name) as search_file_handle:
        search_file_data = search_file_handle.read()
        search_file_handle.close()

    search_file_data = search_file_data.replace("\n",
"")

    index = search_file_data.find(motif)

    return index

if __name__ == "__main__":
    input_file_name = input("Enter a file name >\t")
    generated_motif = motif(input_file_name)
    print("\n\nMotif generation successful.")
    print(f"\nMotif length: {len(generated_motif)}")
    print(f"\nMotif: {generated_motif}")

    search_file_name = input("Enter a file name to be
searched >\t")
```

```
    index = search_for_motif(generated_motif,
search_file_name)


    if index > 0:
        print("Given motif is found at index:\t",
index)
    else:
        print("Motif not found in document")
```

Output:



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:

```
if __name__ == "__main__":
    file_handle = open(input("Enter a filename.>\t"))
    file_data = file_handle.read()
    file_handle.close()

    base_dict = {}

    for i in "ACGT":
        base_dict[i] = file_data.count(i)

    print("\nBLAST search successful.\nTest results:")
    print(f"File name:\t {file_handle.name}")
    print(f"Genome length:\t{sum(base_dict.values())}")
    print(f"Nucleotide count:")
```

```
    for i in "ACGT":
        print(f"\t{i} : {base_dict[i]}")
```

Output:

- Absolute filepath:

```
C:\Temp\Bioinformatics>python BLAST.py
Enter a filename.>      C:/Temp/Salmonella_Enterica.txt

BLAST search successful.
Test results:
File name:       C:/Temp/Salmonella_Enterica.txt
Genome length:  4791950
Nucleotide count:
        A : 1146820
        C : 1243619
        G : 1250672
        T : 1150839
```

- Relative filepath:

```
C:\Temp\Bioinformatics>python BLAST.py
Enter a filename.>      ../Variola.txt

BLAST search successful.
Test results:
File name:       ../Variola.txt
Genome length:  185578
Nucleotide count:
        A : 62782
        C : 30524
        G : 30223
        T : 62049
```

- Current working directory:

```
C:\Temp\Bioinformatics>python BLAST.py
Enter a filename.>      Pithovirus_Sibericum.txt

BLAST search successful.
Test results:
File name:       Pithovirus_Sibericum.txt
Genome length:  610033
Nucleotide count:
        A : 196380
        C : 109028
        G : 109354
        T : 195271
```