⇒ Sample Python Programs :-

① Program to left rotate and right rotate a numpy array by 'n'.

```
arr = input ("Enter the array elements : ")
arr = list (map (int, arr.split ()))
arr = np.array (arr)
n = int (input ("Enter n : "))
print ("Original Array : ", arr)
print (f"Left Rotated Array by {n} : ", np.roll (arr, -n))
print (f"Right Rotated Array by {n} : ", np.roll (arr, n))
```

Output :-

```
Enter the array elements : 1 2 3 4 5
Enter n : 1
Original Array : [1 2 3 4 5]
Left Rotated array by 1 : [2 3 4 5 1]
Right Rotated array by 1 : [5 1 2 3 4]
```

② Program to find an item, the largest, the smallest item in a tuple and to print & display items in a tuple.

```
tup = input ("Enter a Tuple :")
tup = tuple (map (int, tup.split()))
print ("Maximum Element : ", max (tup))
print ("Minimum Element : ", min (tup))
print ("Tuple : ", tup)
print ("Searching an element")
x = int (input ("Enter the Key :"))
if x in tup:
    print (f" Element {x} found at ", tup.index (x)+1)
else:
    print (f" Element {x} not found ")
```

Output :-

```
Enter a Tuple : 1 2 3 4 5
Maximum Element : 5
Minimum Element : 1
Tuple : (1, 2, 3, 4, 5)
Searching an element
Enter the Key : 6
Element 6 not found
```

③ Program to count number of words in a sentence.

```
str = input ("Enter a Sentence :")
my_dict = {}
words = str.lower().split()
for word in words:
    my_dict[word] = words.count(word)
print (my_dict)
```

Output :-

Enter a Sentence : Lion is King of Jungle and it is carnivore

{'Lion' : 1, 'is' : 2, 'King' : 1, 'of' : 1, 'jungle' : 1, 'and' : 1, 'it' : 1, 'carnivore' : 1 }

④ Program to convert list to string and list to tuple

```
words = ["Sun", "rises", "in", "the", "east"]
sentence = " "
sentence = sentence.join(words)
print ("String :", sentence)
tup = tuple(words)
print ("Tuple :", tup)
```

Output :-

Sun rises in the east

('Sun', 'rises', 'in', 'the', 'east')

① Simple Python program using conditional statements, looping, performing operations such as search, insert, update, delete, display and sorting on datatypes like list, tuple, set, dictionary.

```
def search (data):
    X = input ("Enter search element : ")
    for i in range (len (data)):
        if data [i] == X :
            print (f" Element found at {i}")
                return
    print ("Element not found ")
def sort (data):
    for i in range (len (data)):
        flag = False
            for j in range (len (data) - i - 1):
                if data [j] > data [j+1]:
                    data [j], data [j+1] = data [j+1], data [j]
                    flag = True
        if not flag:
            return
```

```
def update (data, key):
    if key not in data:
        print ("Key not found")
    else:
        x = input ("Enter a new value:")
        data[key] = x

list_ = []
def list_op():
    print ("List Operations :")
    print ("1. Insert \n 2. Delete \n 3. Update \n 4. Search \n 5. Sort \n")
    print ("6. Display \n 7. Return to main")
    while True:
        ch = int (input ("Enter Choice :"))
        if ch == 1:
            x = input ("Enter element :")
            list.append (ele)
        elif ch == 2:
            ele = input ("Enter element to be deleted:")
            try:
                list_.remove (ele)
            except:
                print (f"Element {ele} not found")
```

```
        elif ch==3:
            index = int(input("Enter index :"))
            update(list_, index)
        elif ch==4:
            search(list_)
        elif ch==5:
            sort(list_)
        elif ch==6:
            print(list_)
        elif ch==7:
            break
        else:
            print("Invalid Choice")
tup = Tuple()
def tuple_op():
    print("Enter the tuple:")
    tup = tuple(i for i in input().split())
    print("1. Display \n 2. Search \n 3. Exit")
    while True:
        ch = int(input("Enter the choice:"))
        if ch==1:
            print(tup)
```

```
    elif ch==2:
        search(tup)
    elif ch==3:
        break
    else:
        print("Invalid choice")

myset = set()
def set_op():
    print("1.Add\n2.Remove\n3.Search\n4.Display\n5.Exit")
    while True:
        ch = int(input("Enter the choice : "))
        if ch==1:
            myset.add(input("Enter Element"))
        elif ch==2:
            try:
                myset.remove(input("Enter Element: ")
            except:
                print("Element not found")
        elif ch==3:
            search(myset)
        elif ch==4:
            print(myset)
        elif ch==5:
            return
        else:
            print("Invalid choice")
```

```
dict_ = dict()
def dict_operation():
    print ("1. Insert \n2. Update \n3. Delete \n4. Search \n5. Display \n6. Exit")
    While True:
        ch = int (input ("Enter the Choice :"))
        if ch == 1 :
            Key = input ("Enter Key :")
            value = input ("Enter Value :")
            dict [Key] = value
        elif ch = = 2:
            Key = input ("Enter the Key :")
            update (dict_ , Key)
        elif ch = = 3:
            Key = input ("Enter Key to Delete :")
            if Key in dict_ :
                del dict_ [Key]
            else :
                print ("Key not found")
        elif ch == 4:
            ele = input ("Enter the value :")
            for Key, value in dict_items():
                if value = = ele:
                    print(Key)
```

```
    elif ch==5:
        print(dict_)
    elif ch==6:
        break
    else:
        print("Invalid Choice")

while True:
    print("1.List\n2.Tuples\n3.Set\n4. Dictionary\n5.Exit")
    ch = int(input("Enter the Choice:"))
    if ch==1:
        list_op()
    elif ch==2:
        tuple_op()
    elif ch==3:
        set_op()
    elif ch==4:
        dict_operation()
    elif ch==5:
        return
    else:
        print("Invalid Choice")
```

② Visualize the n-dimensional data using Scatter Plots, box plot, heat maps, contour plots, 3D surface plots using python packages.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('xyzdata.csv')
X = data['x']
Y = data['y']
Z = data['z']

# Scatter Plot
plt.scatter(X,Y)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter Plot')
plt.show()

# Box Plot
sns.boxplot(data=data)
plt.title('Box Plot')
plt.show()
```

Heat Map

```
sns.heatmap(data.corr(), annot = True)
plt.title('Heat Map')
plt.show()
```

Contour Plot

```
plt.tricontour(X, Y, Z, levels = 20, cmap = 'jet')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Contour Plot')
plt.show()
```

3D Surface Plot

```
ax = plt.axes(projection = '3d')
ax.set_title('3D Surface Plot')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.plot_trisurf(X, Y, Z, cmap = 'jet')
plt.show()
```

3.a) Write a program to implement the Best First Search (BFS) algorithm

```
def best_first_search(graph, start, goal, heuristic, path=[]):
    open_list = [(0, start)]
    closed_list = set()
    closed_list.add(start)

    while open_list:
        open_list.sort(key=lambda x: heuristic[x[1]], reverse=True)
        cost, node = open_list.pop()
        path.append(node)

    if node == goal:
        return cost, path

    closed_list.add(node)
    for neighbour, neighbour_cost in graph[node]:
        if neighbour not in closed_list:
            closed_list.add(node)
            open_list.append((cost+neighbour_cost, neighbour))

    return None
```

```
graph = {
    'A': [('B', 11), ('C', 14), ('D', 7)],
    'B': [('A', 11), ('E', 15)],
    'C': [('A', 14), ('E', 8), ('D', 18), ('F', 10)],
    'D': [('A', 7), ('F', 25), ('C', 18)],
    'E': [('B', 15), ('C', 8), ('H', 9)],
    'F': [('G', 20), ('C', 10), ('D', 25)],
    'G': [],
    'H': [('E', 9), ('G', 10)]
}

start, goal = 'A', 'G'

heuristic = {'A': 40, 'B': 32, 'C': 25, 'D': 35, 'E': 19, 'F': 17, 'G': 0, 'H': 10}

result = best_first_search(graph, start, goal, heuristic)

if result:
    print(f"Minimum cost from {start} to {goal} is {result[1]}")
    print(f"Cost: {result[0]}")
else:
    print(f"No path from {start} to {goal}")
```

3.b) Write a program to implement the A* algorithm

```
def h(n):
    H = {'A': 3, 'B': 4, 'C': 2, 'D': 6, 'G': 0, 'S': 5}
    return H[n]

def a_star_algorithm(graph, start, goal):
    open_list = [start]
    closed_list = set()
    g = {start: 0}
    parents = {start: start}
    while open_list:
        open_list.sort(key = lambda v: g[v] + h[v], reverse = True)
        n = open_list.pop()
        if n == goal:
            reconst_path = []
            while parents[n] != n:
                reconst_path.append(n)
                n = parents[n]
            reconst_path.append(start)
            reconst_path.reverse()
            print(f"Path found: {reconst_path}")
            return reconst_path
```

```
for (m, weight) in graph[n]:
  # if m is first visited, add it to open_list & note its parent
    if m not in open_list and m not in closed_list:
        open_list.append(m)
        parents[m] = n
        g[m] = g[n] + weight
    # otherwise, check if its quicker to first visit n, then m
    # and if it is, update parent and g data
    # and if node was in closed_list, move it to open_list
    else:
        if g[m] > g[n] + weight:
            g[m] = g[n] + weight
            parents[m] = n
            if m in closed_list:
                closed_list.remove(m)
                open_list.append(m)
    # Node's neighbour visited, put node to
    closed_list.add(n)

print('Path does not exist!')
return None
```

```
graph = {
    'S' : [('A', 1), ('G', 10)],
    'A' : [('B', 2), ('C', 1)],
    'B' : [('D', 5)],
    'C' : [('D', 3), ('G', 4)],
    'D' : [('G', 2)]
}

a_star_algorithm(graph, 'S', 'G')
```