**PERIYAR MANIAMMAI INSTITUTE OF SCIENCE & TECHNOLOGY, THANJAVUR**

## Course: B.SC (AI)

**Year – I** **Semester – 2**

## List of Experiments for Practical Exams

| Exp No | Title of the Experiments |
|---|---|
| 1 | Handling Anaconda Navigator and Jupyter Notebook(Theoretical) |
| 2 | Data Types in Python – I(Numerical, Array & List)<br>Data Types in Python – II(Tuples, Set & Dictionary) |
| 3 | Executing Conditional Statements in Python<br>Building an Expert System in Python using Conditional Statements |
| 4 | Executing For loop, While Loops and Functional Programming in Python |
| 5 | Working with Creating Modules and handling JSON File using Python |
| 6 | Creating class and object using python and Building a Expert System using Class and Objects |
| 7 | Implementation of Binary Search algorithm and Bubble sort algorithm using python |
| 8 | Implementation of Breadth First Search, Depth First Search and Bellman-Ford Algorithm in Python |
| 9 | Fundamentals of Tkinter and Build a Simple Calculator in tkinter using python |
| 10 | Build Some management systems using tkinter |
| 11 | Fundamentals of Pygame and Build a simple snake game in Python |
| 12 | Creating a star ship meteors game in Pygame |
| 13 | Fundamentals of Flask (Building a Simple Flask app) |
| 14 | Build a student Digital profile using FLASK |

## EX:1 Handling Anaconda Navigator and Jupyter Notebook(Theoretical)

**AIM:** To Handling Anaconda Navigator and Jupyter Notebook(Theoretical)

**REQUIREMENTS:** Anaconda Navigator and Jupyter Notebook

**PROCEDURE:**

1.Install Anaconda: First, you need to download and install the Anaconda distribution from the official website. Anaconda comes with many pre-installed packages and libraries that are essential for data science and machine learning.

2.Launch Anaconda Navigator: Once you have installed Anaconda, you can launch Anaconda Navigator. Anaconda Navigator provides a graphical user interface (GUI) that allows you to easily manage your projects, environments, and packages.

3.Create a new environment: You can create a new environment in Anaconda Navigator by clicking on the "Environments" tab and then clicking on "Create". You can specify the name of the environment, the Python version, and the packages you want to install.

4.Launch Jupyter Notebook: Once you have created an environment, you can launch Jupyter Notebook by clicking on the "Home" tab in Anaconda Navigator and then clicking on "Launch" under Jupyter Notebook.

5.Create a new notebook: In Jupyter Notebook, you can create a new notebook by clicking on the "New" button and then selecting "Notebook" under the "Notebook" section. You can then start writing code, adding visualizations, and writing text

6.Install packages: If you need to install additional packages, you can do so in Anaconda Navigator by clicking on the "Environments" tab and then selecting the environment you want to install the package in. You can then search for the package you want to install and click on the checkbox to install it.

7.Save and share notebooks: Jupyter Notebook allows you to save your notebooks in various formats, such as HTML, PDF, or Markdown.

In summary, Anaconda Navigator and Jupyter Notebook are two powerful tools that can help you manage your data science and machine learning projects. By following the tips above, you can easily create new environments, launch Jupyter Notebook, create new notebooks, install packages, and save and share your work with other

**RESULT:**

Thus to Handling Anaconda Navigator and Jupyter Notebook(Theoretical) is verified successfully

## EX 2: DATA TYPES IN PYTHON

AIM: To study and practice about the basic datatypes of python like numeric, array, string,List, tuples, set and dictionary using Jupyter Notebook.

**REQUIRMENTS**:

1. Jupyter Notebook

**CODING**:

#Write Coding's for Numeric, Array, String and List operations using python

**Numeric datatype:**

```
# Integer Example

num1 = 10

print("num1:", num1, type(num1))

# Float Example

num2 = 3.14

print("num2:", num2, type(num2))

# Complex Number Example

num3 = 2 + 3j

print("num3:", num3, type(num3))
```

**Array datatype:**

```
import numpy as np

arr1 = np.array([1, 2, 3])

print("arr1:", arr1, type(arr1))

arr2 = np.array([[1, 2, 3], [4, 5, 6]])

print("arr2:", arr2, type(arr2))
```

**String datatype:**

```
str1 = "Hello World!"

print("str1:", str1, type(str1))
```

**List datatype:**

```
lst1 = [1, 2, 3, "four", "five"]

print("lst1:", lst1, type(lst1))
```

**Tuple datatype:**

```
tup1 = (1, 2, 3, "four", "five")

print("tup1:", tup1, type(tup1))
```

**Set datatype:**

```
set1 = {1, 2, 3, "four", "five"}

print("set1:", set1, type(set1))
```

**Dictionary datatype:**

```
dict1 = {"name": "Vikneshraj", "age": 18, "city": "Berlin"}

print("dict1:", dict1, type(dict1))
```

**RESULT:**

Thus the way we declare and execute the basic datatypes in python is verified Successfully

**Numeric datatype:**

**OUTPUT:**

**Num1: 10 <class ‘int’>**

**Num2: 3.14 <class ‘float’>**

**Num3: (2+3j) <class ‘complex’>**

**Array datatype**

**OUTPUT:**

**Arr1: [1 2 3] <class ‘numpy.ndarray’>**

**Arr2: [[1 2 3]**

**[4 5 6]] <class ‘numpy.ndarray’>**

**String datatype:**

**OUTPUT:**

**Str1: Hello World! <class ‘str’>>**

**List datatype:**

**OUTPUT:**

**Lst1: [1, 2, 3, ‘four’, ‘five’] <class ‘list’>>**

**Tuple datatype:**

**OUTPUT:**

**Tup1: (1, 2, 3, ‘four’, ‘five’) <class ‘tuple’>>**

**Set datatype:**

**OUTPUT:**

**Set1: {1, 2, 3, ‘five’, ‘four’} <class ‘set’>>**

**Dictionary datatype:**

**OUTPUT:**

**Dict1: {‘name’: ‘Vikneshraj’, ‘age’: 18, ‘city’: ‘Berlin’} <class ‘dict’>>**

## EX.3Executing Conditional Statements in Python

## Building an Expert System in Python using Conditional Statements

**AIM**: To Executing Conditional Statements in Python

Building an Expert System in Python using Conditional Statements

**REQIREMENT:**

Jupyter Notebook

**CODING:**

```
Print("Welcome to the school admission form")

Name = input("Please enter your name: ")

Age = int(input("Please enter your age: "))

Grade = int(input("Please enter the grade you are applying for: "))

If age < 5:

 Print("You are not old enough to apply for school.")

 elif age >= 5 and age <= 18:

 If grade < 1 or grade > 12:

 Print("Invalid grade level.")

 elif grade >= 1 and grade <= 6:
```

```
Print("You are eligible for elementary school.")

elif grade >= 7 and grade <= 9:

Print("You are eligible for middle school.")

elif grade >= 10 and grade <= 12:

Print("You are eligible for high school.")

else:

Print("Invalid age.")
```

**RESULT:**

Thus the way we declare and execute the Expert System in Python using Conditional Statements is verified Successfully

**OUTPUT**:

Welcome to the school admission form
Please enter your name: Vikneshraj D
Please enter your age: 16
Please enter the grade you are applying for: 10
You are eligible for high school.

## EX:4Executing For loop, While Loops and Functional Programming in Python

**AIM:** To Executing For loop, While Loops and Functional Programming in Python

**REQIREMENT:**

Jupyter Notebook

**CODING:**

**For Loop:**

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

 print(fruit)
```

**While Loop**:

```
N= input(“Enter the number:”)

Val = 0

I = 0
```

```
While I <= int(n):

 Val += i

 I += 1

Print(f" the sum is {val}")
```

**RESULT:**

Thus the way we declare and execute the For loop, While Loops and Functional Programming in Python is verified Successfully.

**For loop:**

**OUTPUT:**

apple

banana

cherry

**While Loop:**

**OUTPUT:**

Enter the number:9

The sum is 45

**EX:5Working with Creating Modules and handling JSON File**

**using Python**

**AIM:** To Create Modules and handling JSON File using Python

**REQIREMENT:**

Jupyter Notebook

**CODING:**

```
Import json

Dictionary ={

 “name” :”VIKNESHRAJ ”,

 “rollno” :30,

 “cgps” :9.7,

 “phonenumber”:”6380777345

}

With open(“sample.json”,’w’)as outline:

 Json.dump(dictionary,outline)

Import json

# open the JSON file and read its contents
```

```
With open('sample.json', 'r') as f:

 Data = json.load(f)

# access the contents of the JSON object

Print(data['name'])

Print(data['rollno'])

Print(data['cgps'])

Print(data['phonenumber'])
```

**RESULT:**

Thus the way we declare and execute the Modules and handling JSON File using Python is verified Successfully.

**OUTPUT:**

VIKNESHRAJ

30

9.7

6380777777

## EX:6 Creating class and object using python and Building a Expert System using Class and Objects

**AIM:**To Create class and object using python and Building a Expert System using Class and Objects

**REQIREMENT:**

Jupyter Notebook

**CODING:**

**Expert System using Class and objects:**

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def print_details(self):

        print(f"{self.name} is {self.age} years old.")

person1 = Person("Vikneshraj", 18)

person1.print_details()



from experta import *
```

```
class meds(KnowledgeEngine):

    @DefFacts()
    def _initial_action(self):
        yield Fact(action ='load')

    @Rule(Fact(action ='load'), NOT(Fact(fulltime = W())))
    def start_quest(self):
        print("Welcome to the Medical Expert system.")
        self.declare(Fact(intro = input("please enter your name: ")))
        self.declare(Fact(fulltime= input("Do you want to enter the Medical
expert system? ")))

    @Rule(Fact(action='load'),(Fact(fulltime = 'no')))
    def exiting(self):
        print("Thank you!")

    @Rule(Fact(action = 'load'),(Fact(fulltime = 'yes')))
    def fever_check(self):
        self.declare(Fact(Fever = input("Do you have fever for the last  few
days?")))
    @Rule (Fact(action = 'load'), AND(Fact(fulltime = 'yes'),NOT(Fact(Fever =
'not sure'))))
    def cough_check(self):
        self.declare(Fact(cough = input("Do you have dry cough for the last few
days?")))

    @Rule(Fact(action='load'),AND(Fact(fulltime = 'yes'),NOT(Fact(Fever =
'not sure')),NOT(Fact(cough = 'not sure'))))
    def tired_check(self):
        self.declare(Fact(Tired = input("Have you been feeling tired?")))

    @Rule(Fact(action = 'load'),AND(Fact(fullltime='yes'),AND(Fact(Fever =
'yes'),Fact(cough = 'no'),Fact(Tired = 'no'))))
    def accept_1(self):
        print("You have fever,please take rest and have paracetanol")
```

```
   @Rule(Fact(action = 'load'),AND(Fact(fulltime='yes'),AND(Fact(Fever =
'no'),Fact(cough = 'yes'), Fact(Tired = 'n0'))))
   def accept_2(self):
      print("You just have dry cough . please gargle, steae and have lots of hot
water.")

   @Rule(Fact(action = 'load'),AND(Fact(fulltime='yes'), Fact(Fever = 'yes'),
Fact(cough = 'yes'),Fact(Tired = 'yes')))
   def accept_3(self):
      print(" You are showing symptoms of COVID-19.Please get yourself
tested and stay quarentined.")

   @Rule(Fact(action = 'load'),AND(Fact(fulltime='yes'), Fact(Fever = 'no'),
Fact(cough = 'yes'),Fact(Tired = 'yes')))
   def accept_4(self):
      print(" Please visit the doctor as you may  have a throat infection.")

   @Rule(Fact(action = 'load'),AND(Fact(fulltime='yes'), Fact(Fever = 'yes'),
Fact(cough = 'no'),Fact(Tired = 'yes')))
   def accept_5(self):
      print("You may ba having a viral infection. Take ample rest.If it presists
please visit a doctor.")
@Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'), OR(Fact(Fever =
'yes'),Fact(Fever = 'no')),
                           OR(Fact(cough = 'yes'),Fact(cough = 'no')),
                           OR(Fact(Tired = 'yes'),Fact(Tired = 'no'))))
   def adv_expt(self):
      print("You have completed the simple medical expert system.")
      self.declare(Fact(dep_dive = input("Do you want to dive deeper into the
expert system?")))

   @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive =
'no')))
   def div_reject(self):
      print("Thank you for using our expert system.")

   @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive
='yes')))
   def breath(self):
```

```
        self.declare(Fact(breathing = input("Have you been experiencing
shortness of breath?")))

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive =
'yes'),
                                OR(Fact(breathing ='yes'),Fact(brearthing = 'no'))))
    def chest_pain(self):
        self.declare(Fact(chest = input("Have you been experiencing acute chest
pain or pressure?")))

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive =
'yes'),
                                OR(Fact(breathing = 'yes'),Fact(breathing='no'),
OR(Fact(chest = 'yes'),Fact(chest = 'no')))))

    def speech_loss(self):
        self.declare(Fact(loss = input("Have you been experiencing any loss of
speech or movement?")))

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'), Fact(dep_dive =
'yes'),Fact(breathing = 'yes'),Fact(loss ='no'),Fact(chest = 'no')))
    def accept_6(self):
        print("You seem to be having shortness of breath. Even if you  are not
COVID positive, this is serious.")
        print("Go to the doctor immediately.")

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive =
'yes'),Fact(breathing = 'no'),Fact(loss = 'no'),Fact(chest='no')))

    def accept_7(self):
        print("You seem to be having either loss of speech or movement. Even if
you  are not COVID positive, this is serious.")
        print("Go to the doctor immediately.")

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive =
'yes'),Fact(breathing = 'no'),Fact(loss = 'no'),Fact(chest='yes')))

    def accept_8(self):
```

```
        print("You seem to be having  chest pain. Even if you  are not COVID positive, this is serious.")
        print("Go to the doctor immediately.")

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive = 'yes'),Fact(breathing = 'yes'),Fact(loss = 'no'),Fact(chest='yes')))

    def accept_9(self):
        print("You seem to be having  chest pain and shortness of breath. Even if you  are not COVID positive, this is serious.")
        print("Go to the doctor immediately.")


    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive = 'yes'),Fact(breathing = 'no'),Fact(loss = 'yes'),Fact(chest='yes')))

    def accept_10(self):
        print("You seem to be having  chest pain and loss of speech or motion. Even if you  are not COVID positive, this is serious.")
        print("Go to the doctor immediately.")
    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive = 'yes'),Fact(breathing = 'yes'),Fact(loss = 'yes'),Fact(chest='no')))
    def accept_11(self):
        print("You seem to be having shortness of breath and loss of speech or movement. Even if you  are not COVID positive, this is serious.")
        print("Go to the doctor immediately.")

    @Rule(Fact(action = 'load'),AND(Fact(fulltime = 'yes'),Fact(dep_dive = 'yes'),Fact(breathing = 'yes'),Fact(loss = 'yes'),Fact(chest='yes')))

    def accept_12(self):
        print("You seem to be having  chest pain and shortness of breath and loss of speech or movement. Even if you  are not COVID positive, this is serious.")
        print("Go to the doctor immediately.")

Engine = meds()
Engine.reset()
Engine.run()
```

**Result:**

Thus the way we declare and execute the class and object using python and Building a Expert System using Class and Objects is verified Successfully.

## Class and objects:

**Output:**

Vikneshraj D is 18 years old

## Expert System using Class and object:

**Output:**

Welcome to the Medical Expert system.
please enter your name : Vikneshraj D
Do you want to enter the Medical expert system? yes
Do you have fever for the last few days? yes
Do you have dry cough for the last few days? yes
Have you been feeling tired? yes
You are showing symptoms of COVID-19.Please get yourself tested and stay quarantined.
You have completed the simple medical expert system.
Do you want to dive deeper into the expert system? yes
Have you been experiencing shortness of breath? yes
Have you been experiencing acute chest pain or pressure? yes
Have you been experiencing any loss of speech or movement? yes
You seem to be having chest pain and shortness of breath and loss of speech or movement. Even if you are not COVID positive, this is serious.
Go to the doctor immediately.

**EX:7Implementation of Binary Search algorithm and Bubble sort algorithm using python**

**AIM**: To Implementation of Binary Search algorithm and Bubble sort algorithm using python

**REQIREMENT:**

Jupyter Notebook

**CODING:**

**Binary Search Algorithm:**

```
data =
[110,11,22,33,44,55,66,10,20,30,40,1,2,3,4,5,6,7,8,9,12,13,14,15]

data.sort()

print(data)

elem=int(input("enter the search element :"))

def binary_search(data, elem):

   low = 0

   high = len(data) - 1

   while low <= high:

      middle = (low + high) // 2
```

```
        if data[middle] == elem:
            print(f"The search element {elem} is present at index value
{middle} in dataset")
            break
        elif data[middle] > elem:
            high = middle - 1
        else:
            low = middle + 1
    if data[middle] != elem:
        print(f"The search element {elem} is not present in the dataset")
    return -1
binary_search(data,elem)
```

**Bubble Sort Algorithm:**

```
# Bubble sort in python

def bubblesort(data):

  # loop to access each array in element
  for i in range(len(data)):
```

```
    # loop to compare array element
    for j in range(0,len(data) - i - 1):

      #compare two adjecent element
       if data[j] > data [j + 1]:

          # swapping element if element are not in the intemded order
          temp = data[j]
          data[j] = data[j+1]
          data[j+1] = temp

data = [-1,10,1,2,33,45,66,77,11,23,43,111,112]

print('Before sorting the Array in Ascending Order :')
print(data)

bubblesort(data)

print("After Before sorting the array in Ascending Order:")
print(data)
```

**Result:**

Thus the way we declare and execute the Binary Search algorithm and Bubble sort algorithm using python is verified Successfully

## Binary Search Algorithm:

**Output:**

The search element 11 is present at index value 10 in data
-1

## Bobble Sort Algorithm:

**Output:**

Before sorting the Array in Ascending Order :
[-1, 10, 1, 2, 33, 45, 66, 77, 11, 23, 43, 111, 112]
After Before sorting the array in Ascending Order:
[-1, 1, 2, 10, 11, 23, 33, 43, 45, 66, 77, 111, 112]

**EX:8 Implementation of Breadth First Search, Depth First Search and BellmanFord Algorithm in Python Program: Breadth First Search:**

**AIM:** To Implementation of Breadth First Search, Depth First Search and BellmanFord Algorithm in Python Program: Breadth First Search

**REQUIREMENT**: Jupyter Notebook

**CODING**:

**Breadth First Search:**

```
graph={
   'sam':['Aaron','binny'],
   'Aaron':['sam','christine','Denny'],
   'binny':['Elvin','Flin'],
   'christine':['Aaron'],
   'Denny':['Aaron'],
   'Elvin':['binny','Gini'],
   'Flin':['binny'],
   'Gini':['Elvin']
}
```

```
from collections import deque
def bfs(graph,start,goal):
   visited=[]
   queue=deque([start])
   while queue:
      node=queue.popleft()
      if node not in visited:
         visited.append(node)
         print("I have visited:",node)
```

```
        neighbours=graph[node]
        if node==goal:
            return("I have reached the goal, this is my traversed path:",visited)
        for neighbour in neighbours:
            queue.append(neighbour)


bfs(graph,'binny','Flin')
```

**Depth First Search:**

```
graph={
    'sam':['Aaron','binny'],
    'Aaron':['sam','christine','Denny'],
    'binny':['Elvin','Flin'],
    'christine':['Aaron'],
    'Denny':['Aaron'],
    'Elvin':['binny','Gini'],
    'Flin':['binny'],
    'Gini':['Elvin']
}


def dfs(graph,start,goal,visited):

    stack = [start]

    while stack:

        node = stack.pop()

        if node not in visited:

            visited.append(node)
```

```
        if node==goal:
            print(visited)
        for neighbors in graph[node]:
            dfs(graph,start,goal,visited)
dfs(graph,'sam','Gini',[])
```

**Bellman-Ford Algorithm:**

```
import sys
# Initialize distances and previous vertices
def initialize(graph, source):
    distances = {}
    previous = {}
    for vertex in graph:
        distances[vertex] = sys.maxsize
        previous[vertex] = None
    distances[source] = 0
    return distances, previous
def bellman_ford(graph, source):
```

```
    distances, previous = initialize(graph, source)

    # Relax edges N-1 times

    for i in range(len(graph) - 1):

        for u in graph:

            for v in graph[u]:

                if distances[u] + 1 < distances[v]:

                    distances[v] = distances[u] + 1

                    previous[v] = u

    # Check for negative weight cycles

    for u in graph:

        for v in graph[u]:

            assert distances[v] <= distances[u] + 1, "Negative weight cycle
detected."

    return distances, previous

# Test the algorithm with the given graph

graph = {

    'sam': ['Aaron', 'binny'],
```

```
    'Aaron': ['sam', 'christine', 'Denny'],
    'binny': ['Elvin', 'Flin'],
    'christine': ['Aaron'],
    'Denny': ['Aaron'],
    'Elvin': ['binny', 'Gini'],
    'Flin': ['binny'],
    'Gini': ['Elvin']
}
distances, previous = bellman_ford(graph, 'sam')
print(distances)
```

**Result**:

Thus the way we declare and execute the Breadth First Search, Depth First Search and BellmanFord Algorithm in Python Program:
Breadth First Search is verified Successfully.

**Breadth First Search:**

**Output:**

I have visited: binny

I have visited: Elvin

I have visited: Flin

('I have reached the goal, this is my traversed path:',
['binny', 'Elvin', 'Flin'])

**Depth First Search:**

**Output**:

['sam', 'Aaron', 'christine', 'Denny', 'binny', 'Elvin', 'Gini']

**Bellman-Ford Algorithm:**

**Output:**

{'sam': 0, 'Aaron': 1, 'binny': 1, 'christine': 2, 'Denny': 2,
'Elvin': 2, 'Flin': 2, 'Gini': 3}