**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)  Data Types in Python – II(Tuples, Set & Dictionary) |
| 3 | Executing Conditional Statements in Python  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:

num = 7 # Integer

num

type(num) #Find the datatype of num variable

num\_flt = 3.141 # Floating Values

num\_flt

type(num) #Find the datatype of num\_flt variable

num\_clx = 1+2j # Complex Number

num\_clx

type(num) #Find the datatype of num\_clx variable

List: fruits = ["Watermelon","Apple","Banana"]

fruits

type(fruits)

#Accessing List Elements

List[4]

Fruits[-1]

#Find the Length of List

len(fruits)

#Adding Elements to List

fruits.append("Orange")

#Removing Elements From the List

fruits.remove("Kiwi")

fruits.pop()

del fruits[1]

**OUTPUT:**

['Watermelon', 'Apple', 'Banana', 'Orange']

['Watermelon', 'Banana', 'Orange']

['Watermelon', 'Banana']

['Watermelon']

**RESULT:**

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

**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.”)

**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.

**RESULT:**

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

**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)

**OUTPUT:**

apple

banana

cherry

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}”)

**OUTPUT:**

Enter the number:9

The sum is 45

**RESULT:**

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

**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’]) # outputs “VIKNESHRAJ”

Print(data[‘rollno’]) # outputs 30

Print(data[‘cgps’]) # outputs 9.7

Print(data[‘phonenumber’]) # output “6380777777”

**OUTPUT:**

VIKNESHRAJ

30

9.7

6380777777

**RESULT:**

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

**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:**

from experta import \*

class Greetings(KnowledgeEngine):

@DefFacts()

def \_initial\_action(self):

yield Fact(action="greet")

@Rule(Fact(action='greet'),

NOT(Fact(name=W())))

def ask\_name(self):

self.declare(Fact(name=input("What's your name?")))

@Rule(Fact(action='greet'),

NOT(Fact(location=W())))

def ask\_location(self):

self.declare(Fact(location=input("Where are you?")))

@Rule(Fact(action='greet'),

Fact(name=MATCH.name),

Fact(location=MATCH.location))

def greet(self,name,location):

print("Hi %s! How is the weather in %s?" % (name,location))

engine = Greetings()

engine.reset()

engine.run()

**Output:**

What's your name? Vikneshraj D

Where are you? Berlin

Hi Vikneshraj D! How is the weather in Berlin?

**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.

**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)

**Output:**

The search element 11 is present at index value 10 in dataset

-1

**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)

**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]

**Result:**

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

**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**:

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')

**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:**

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',[])

**Output**:

['sam', 'Aaron', 'christine', 'Denny', 'binny', 'Elvin', '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)

**Output:**

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

**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.