**PRACTICAL-1**

**AIM:** To understand the overall programming architecture using Map Reduce API.

The MapReduce task is mainly divided into two phases Map Phase and Reduce Phase

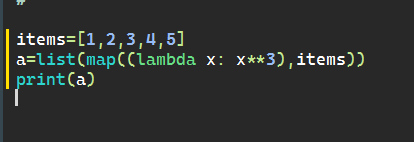
1.map(),filter(), and reduce() in python.

2.These functions are most commonly used with lambda function.

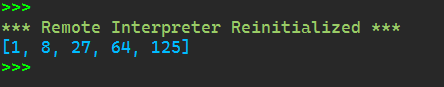
1.map():

-“A map function executes certain instructions or functionality provided to it on every item of an iterable.” The iterable could be list , tuple, set, etc.

-Syntax: map(function, iterable)



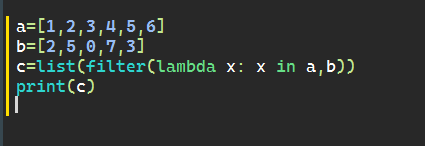
-Output:



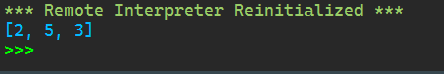
2.filter():

-“A filter function in Python tests a specific user-defined condition for a function and returns an iterable for the elements and values that satisfy the condition”

-Syntax: filter(function, iterable)



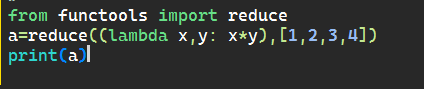
-Output:



3.reduce():

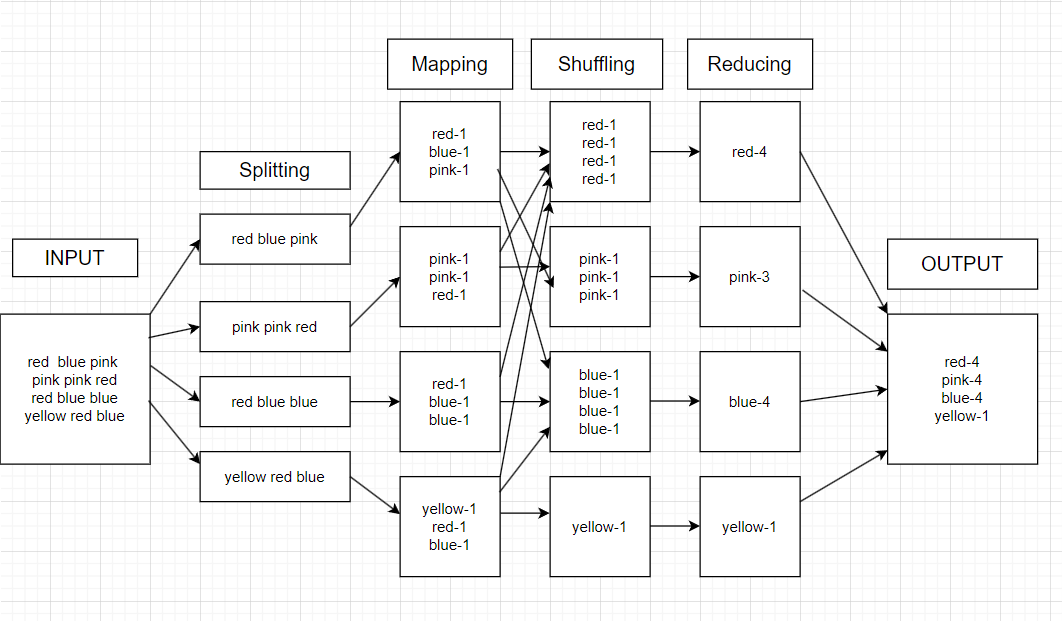
-“Reduce functions apply a function to every item of an iterable and gives back a single value as a resultant.”

-Syntax: reduce(function, iterable)



-Output:





**Practical-2**

**Aim: Write a program of Word Count in Map Reduce over HDFS.**

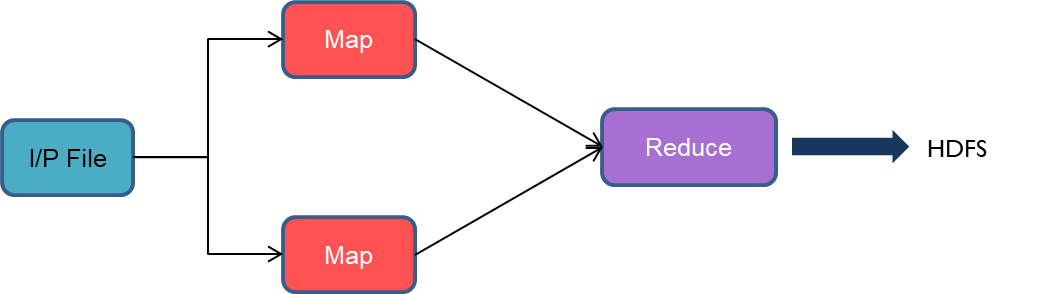
**Description:**

MapReduce is a framework for processing large datasets using a large number of computers (nodes), collectively referred to as a cluster. Processing can occur on data stored in a file system (HDFS).A method for distributing computation across multiple nodes.Each node processes the data that is stored at that node.

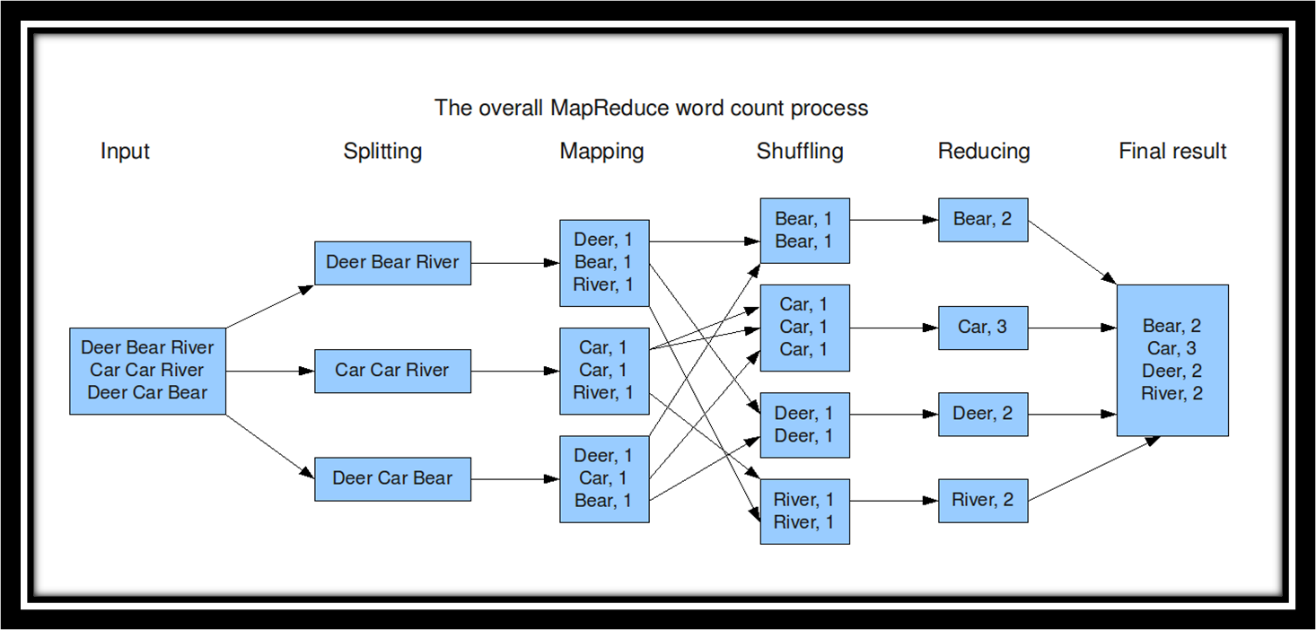
Consists of two main phases

Mapper Phase

Reduce phase



Input data set is split into independent blocks – processed in parallel. Each input split is converted in Key Value pairs. Mapper logic processes each key value pair and produces and intermediate key value pairs based on the implementation logic. Resultant key value pairs can be of different type from that of input key value pairs. The output of Mapper is passed to the reducer. Output of Mapper function is the input for Reducer. Reducer sorts the intermediate key value pairs. Applies reducer logic upon the key value pairs and produces the output in desired format.Output is stored in HDFS

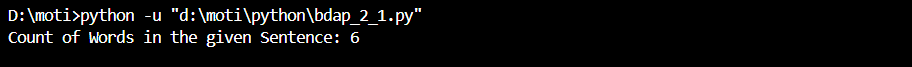


**Execution Step:**

# Quick Two Line Codes

countOfWords **=** len("Geeksforgeeks is best Computer Science Portal".split())

**print**("Count of Words in the given Sentence:", countOfWords)

Output

# Quick One Line Codes:- to findout onliner program

**print**(len("Geeksforgeeks is best Computer Science Portal".split()))

Output

# Quick One Line Code with User Input

print(len(input("Enter Input:").split()))

type:2  
  
from collections import Counter

def word\_count(file\_path):

    with open(file\_path, 'r') as file:

        text = file.read()

    words = text.split()

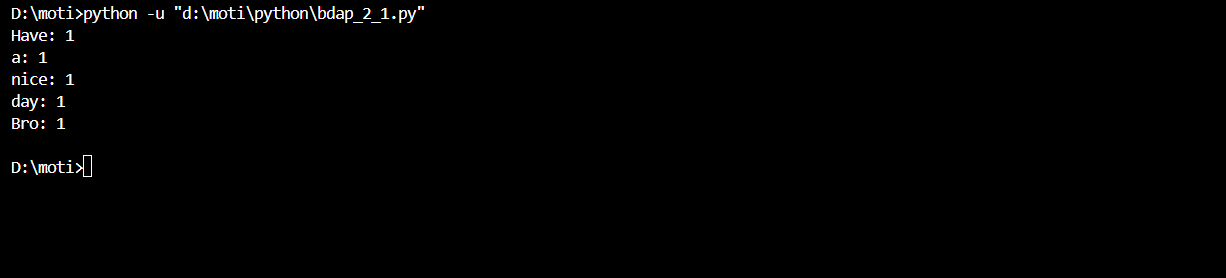
    word\_counts = Counter(words)

    for word, count in word\_counts.items():

        print(f"{word}: {count}")

file\_path = r"D:\moti\python\vasu.txt"

word\_count(file\_path)

Output

import urllib.request

import random

from operator import itemgetter

current\_word = {}

current\_count = 0

story ='http://sixty-north.com/c/t.txt'

request = urllib.request.Request(story)

response = urllib.request.urlopen(request)

each\_word = []

words = None

count = 1

same\_words ={}

word = []

"""looping the entire file"""

#Collect All the words into a list

for line in response:

    #print "Line = " , line

    line\_words = line.split()

    for word in line\_words:

        each\_word.append(word)

for words in each\_word:

    if words.lower() not in same\_words.keys():

        same\_words[words.lower()]=1

for line in response:

    line\_words =line.split()

    for word in line\_words:

        each\_word.append(word)

for words in each\_word:

    if words.lower() not in same\_words.keys():

        same\_words[words.lower()]=1

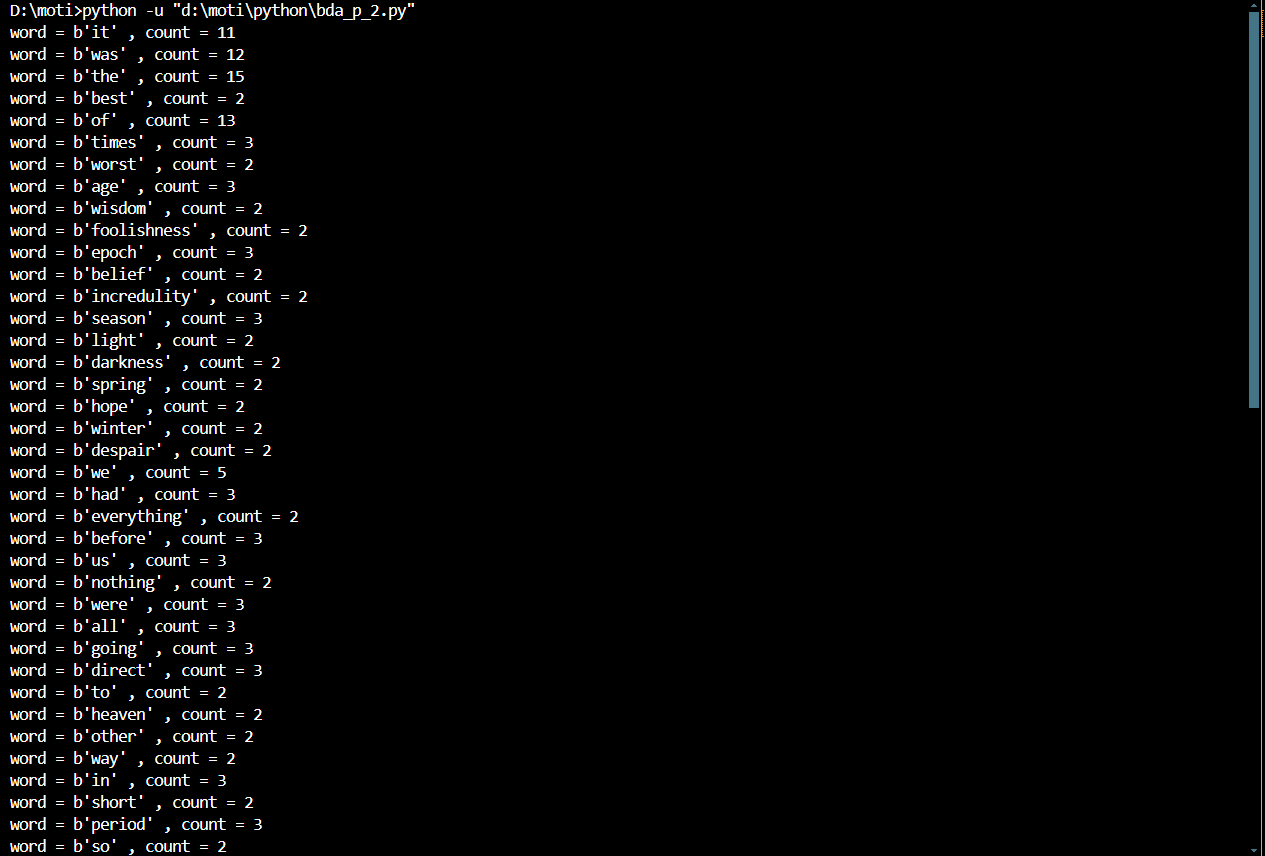
    else:

        same\_words[words.lower()]=same\_words[words.lower()]+1

for each in same\_words.keys():

    print("word =",each, ", count =",same\_words[each])

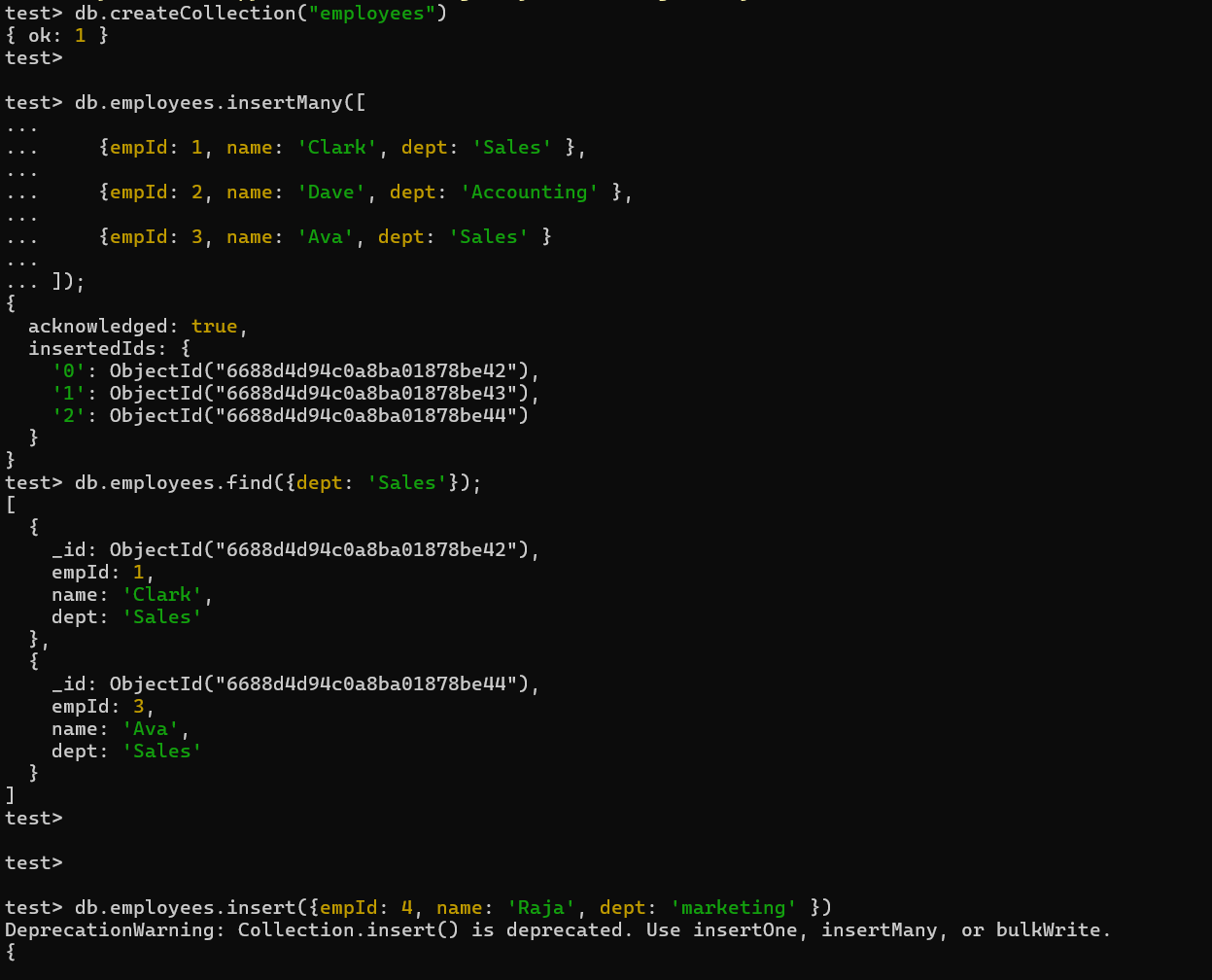
Output

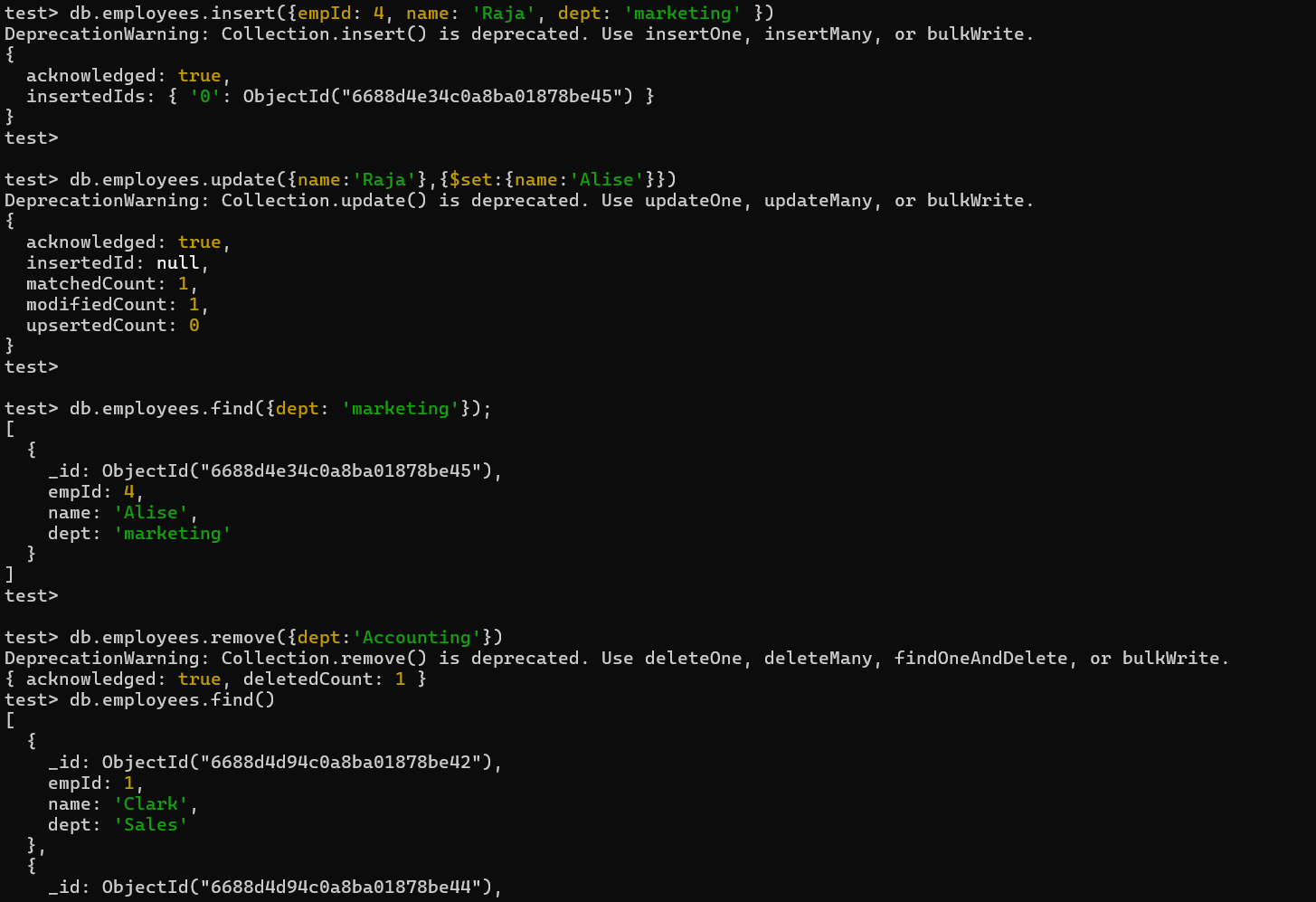
****

**Practical-3**

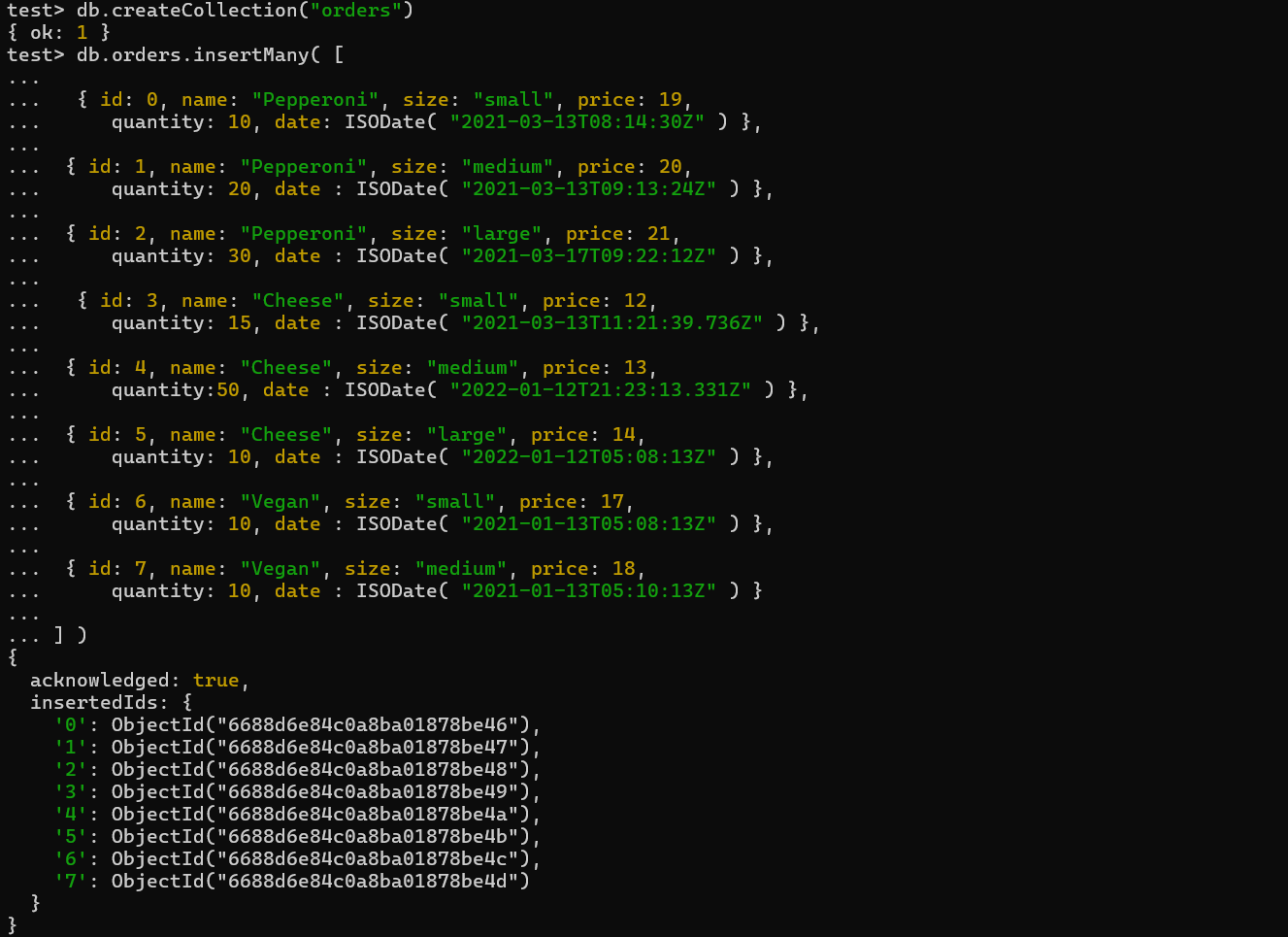
Aim:- Basic CRUD operations in MongoDB.

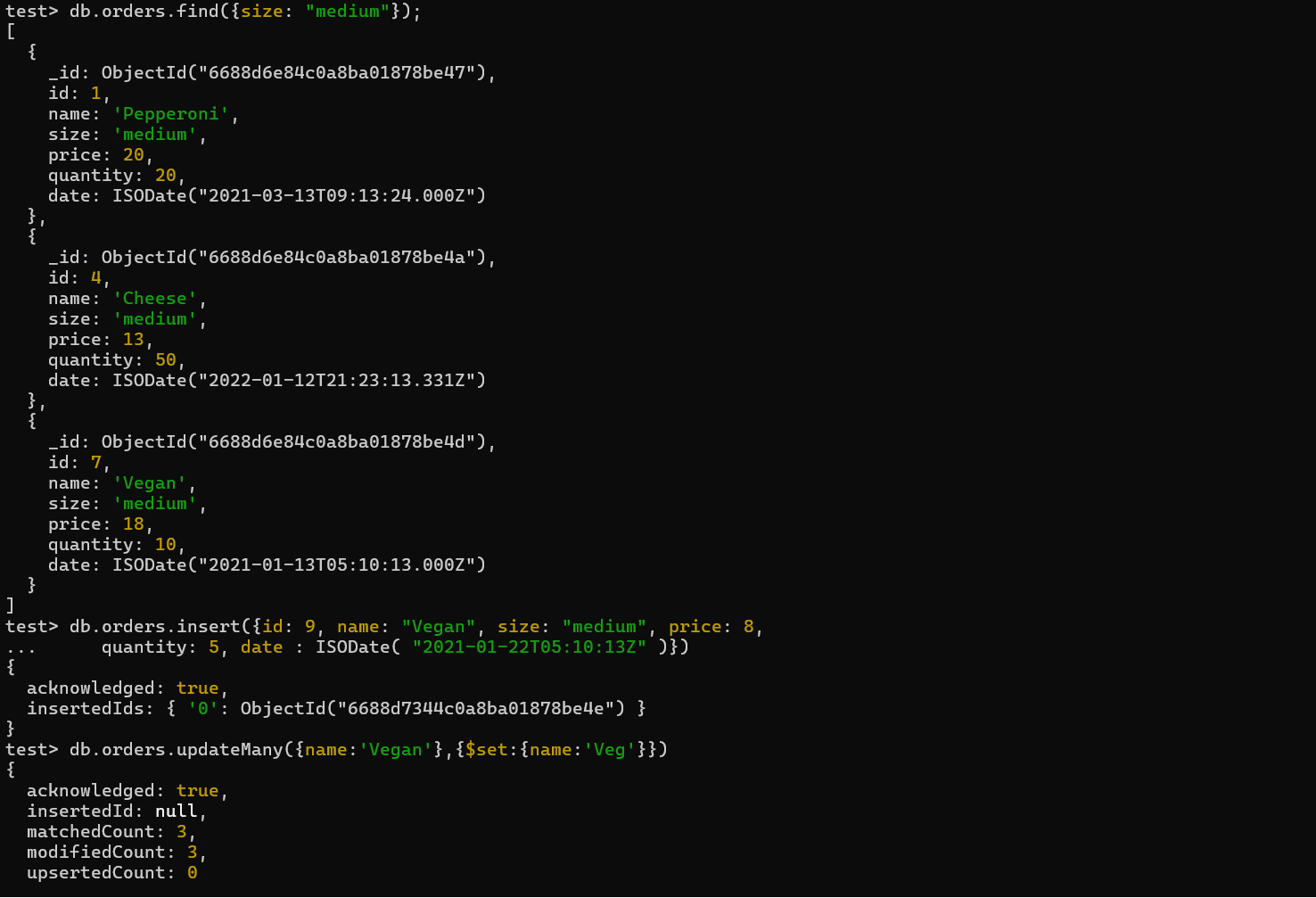
program 1



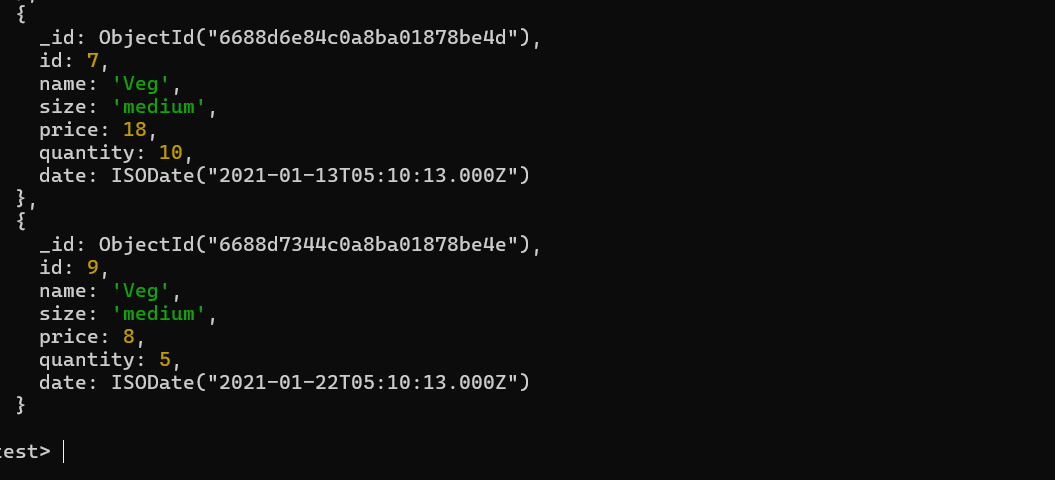


program 2









**Practical-4**

Aim : Store the basic information about students such as roll number, name, date of birth and address of student using various collection types such as list, set and map.

Lists

A list of elements of type t has type sp.list[t]. For example, [1,2,3] has type sp.list[sp.int].

Sets

A set containing elements of type t is represented as {..} and has the type sp.set[t]. For instance, the set {1,2,3} is of type sp.set[sp.int]. To create an empty set, use set().

Maps

A Map that takes element of type k to elements of type v has type sp.map[k,v].SmartPy maps are similar to python’s Dictionaries. sp.map({‘a’:65},{‘b’:66}).

**Using Lists**

# List of students with basic information

students\_list = [

{"roll\_number": 1, "name": "Aladdin", "date\_of\_birth": "1863-01-01",

"address": "Baggdad"},

{"roll\_number": 2, "name": "Bibbojaan", "date\_of\_birth": "1900-05-20",

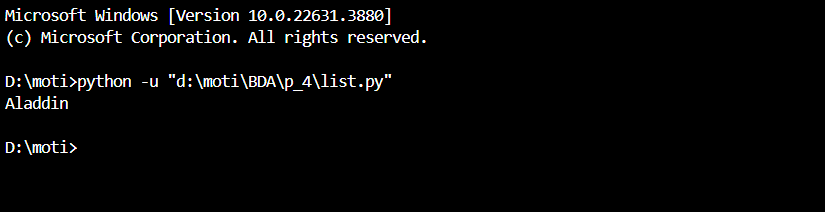
"address": "Lahore"},

{"roll\_number": 3, "name": "Chudail", "date\_of\_birth": "1200-03-10",

"address": "Pipal ka ped"}

print (students\_list [0] ["name"])

Output:-



**Using Sets**

# Set of student names (assuming names are unique)

students\_set = {

"Rasmika Mandana",

"Dimpti Dimri",

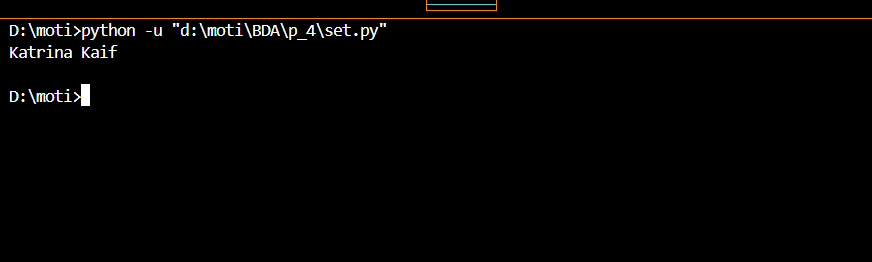
"Katrina Kaif"

}

# Example usage:

if "Katrina Kaif" in students\_set:

print("Katrina Kaif")

Output:-

**Using Maps**

# Dictionary mapping roll numbers to student information

students\_dict = {

1: {"name": "A", "date\_of\_birth": "2000-01-15", "address": "123 Main

St"},

2: {"name": "B", "date\_of\_birth": "1999-05-20", "address": "456 Elm

St"},

3: {"name": "C", "date\_of\_birth": "2001-03-10", "address": "789 0ak

St"}

# Example usage:

print (students\_dict[1] ["address"]) # Output: 123 Main St

Output:-

