**Lab Manual: Advanced Python Programming**

**Class M.Sc. CS/IT**

**Note: Below programs run on python interpreter and write it down with proper comments and output.**

## Experiment 1: Implementation of Advanced Data Structures in Python

**Objective:** Implement and analyze sets, dictionaries, heaps, and graphs.

**Program:**

```
# Sets and Dictionaries
my_set = {1, 2, 3, 4}
my_dict = {'a': 1, 'b': 2, 'c': 3}
print("Set Operations:", my_set)
print("Dictionary Operations:", my_dict)

# MinHeap and MaxHeap
import heapq
heap = []
heapq.heappush(heap, 10)
heapq.heappush(heap, 5)
heapq.heappush(heap, 15)
print("MinHeap:", [heapq.heappop(heap) for _ in range(len(heap))])

# Graph using adjacency list
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],
    'C': ['A', 'D'],
    'D': ['B', 'C']
}
print("Graph Representation:", graph)
```

**Output: -**

## Experiment 2: Algorithm Design and Analysis

**Program:**

```
# Recursive Binary Search
def binary_search(arr, low, high, key):
    if high >= low:
        mid = (high + low) // 2
        if arr[mid] == key:
```

```
            return mid
        elif arr[mid] > key:
            return binary_search(arr, low, mid - 1, key)
        else:
            return binary_search(arr, mid + 1, high, key)
    return -1

arr = [2, 3, 4, 10, 40]
key = 10
result = binary_search(arr, 0, len(arr)-1, key)
print("Element found at index:" if result != -1 else "Element not found")
```

**Output: -**

## Experiment 3: Time and Space Complexity Analysis

**Program:**

```
import time

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

data = [64, 34, 25, 12, 22, 11, 90]
start_time = time.time()
bubble_sort(data)
end_time = time.time()
print("Sorted array:", data)
print("Time Complexity:", end_time - start_time, "seconds")
```

**Output: -**

## Experiment 4: Object-Oriented Programming in Python

**Program:**

```
class Animal:
    def __init__(self, name):
        self.name = name
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        return "Bark"

class Cat(Animal):
```

```
    def make_sound(self):
        return "Meow"

dog = Dog("Rex")
cat = Cat("Whiskers")
print(dog.name, "says", dog.make_sound())
print(cat.name, "says", cat.make_sound())
```

**Output: -**

## Experiment 5: Design Patterns in Python

**Program:**

```
class Singleton:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Singleton, cls).__new__(cls)
        return cls._instance

obj1 = Singleton()
obj2 = Singleton()
print("Objects are the same:", obj1 is obj2)
```

**Output: -**

## Experiment 6: Functional Programming in Python

**Program:**

```
from functools import reduce
nums = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x*x, nums))
even_nums = list(filter(lambda x: x % 2 == 0, nums))
sum_nums = reduce(lambda x, y: x + y, nums)
print("Squared:", squared)
print("Even Numbers:", even_nums)
print("Sum:", sum_nums)
```

**Output: -**

## Experiment 7: Concurrency and Parallelism in Python

### Program:

```
import threading

def print_numbers():
    for i in range(5):
        print(i)

t1 = threading.Thread(target=print_numbers)
t1.start()
t1.join()
print("Thread execution completed")
```

### Output: -

## Experiment 8: Database Connectivity in Python

### Program:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY
KEY, name TEXT)''')
cursor.execute("INSERT INTO students (name) VALUES ('John Doe')")
conn.commit()
cursor.execute("SELECT * FROM students")
print(cursor.fetchall())
conn.close()
```

### Output: -

## Experiment 9: Object-Relational Mapping (ORM) in Python

**Program:**

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine('sqlite:///students.db')
Base = declarative_base()

class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String)

Base.metadata.create_all(engine)
Session = sessionmaker(bind=engine)
session = Session()
new_student = Student(name='Jane Doe')
session.add(new_student)
session.commit()
print("Student added successfully")
```

**Output: -**

## Experiment 10: Case Study Implementation

## Go to the below link and perform the case study given here.

## Create your own repository on GitHub and share the repository link in the book and write down the code .

**https://www.kaggle.com/code/devraai/temperature-data-analysis-prediction**