Ex.No: 1	PYTHON PROGRAM TO ILLUSTRATE RECURSIVE FUNCTION
Date :	

AIM:

To implement a recursive function in Python to calculate the factorial of a number.

PROCEDURE:

Step 1: Define a base case: If n≤1n \leq 1n≤1, return 1.

Step 2: Define the recursive case: Return n×factorial(n−1)n \times \text{factorial}(n-1)n×factorial(n−1).

Step 3: Take user input and call the recursive function.

Step 4: Display the result.

Source Code :

```
def factorial(n):
    return 1 if n <= 1 else n * factorial(n - 1)
num = int(input("Enter a number: "))
print(f"The factorial of {num} is {factorial(num)}")
```

Output:

```
Enter a number: 5
The factorial of 5 is 120
```

Result :

Thus the python program to illustrate recursive function is excuted successfully and verified.

EX.No: 2	A PYTHON PROGRAM TO MAP IDS TO EMPLOYEE NAMES USING DICTIONARIES AND GENERATE 100 LINEARLY SPACED NUMBERS FROM 1 TO 25
Date :	

AIM:

To write a Python program that maps employee IDs to names using dictionaries and generates 100 linearly spaced numbers between 1 and 25.

PROCEDURE:

Step 1 : Use a dictionary to map employee IDs (keys) to employee names (values).

Step 2: Use the numpy linspace function to generate 100 evenly spaced numbers between 1 and 25.

Step 3: Print the employee dictionary and the generated numbers.

Step 4: Display the result.

Source Code :

```
import numpy as np

employees = {101: "Alice", 102: "Bob", 103: "Charlie"}

print("Employee Dictionary:", employees)

numbers = np.linspace(1, 25, 100)

print("Linearly spaced numbers:", numbers)
```

Output :

```
Employee Dictionary: {101: 'Alice', 102: 'Bob', 103: 'Charlie'}
Linearly spaced numbers: [ 1.          1.24242424  1.48484848  1.72727273  1.96969697
2.21212121
  2.45454545  2.6969697   2.93939394  3.18181818  3.42424242  3.66666667
  3.90909091  4.15151515  4.39393939  4.63636364  4.87878788  5.12121212
  5.36363636  5.60606061  5.84848485  6.09090909  6.33333333  6.57575758
  6.81818182  7.06060606  7.3030303   7.54545455  7.78787879  8.03030303
  8.27272727  8.51515152  8.75757576  9.          9.24242424  9.48484848
  9.72727273  9.96969697 10.21212121 10.45454545 10.6969697  10.93939394
 11.18181818 11.42424242 11.66666667 11.90909091 12.15151515 12.39393939
 12.63636364 12.87878788 13.12121212 13.36363636 13.60606061 13.84848485
 14.09090909 14.33333333 14.57575758 14.81818182 15.06060606 15.3030303
 15.54545455 15.78787879 16.03030303 16.27272727 16.51515152 16.75757576
 17.         17.24242424 17.48484848 17.72727273 17.96969697 18.21212121
 18.45454545 18.6969697  18.93939394 19.18181818 19.42424242 19.66666667
 19.90909091 20.15151515 20.39393939 20.63636364 20.87878788 21.12121212
 21.36363636 21.60606061 21.84848485 22.09090909 22.33333333 22.57575758
 22.81818182 23.06060606 23.3030303  23.54545455 23.78787879 24.03030303
 24.27272727 24.51515152 24.75757576 25.        ]
```

Result :

Thus the a pytrhon program to map IDs to employee names using dictionaries and generate 100 linearly spaced numbers from 1 to 25 is executed successfully and verified.

Ex.No:3	PYTHON PROGRAM TO PRINT PYRAMID PATTERN BY TAKING NUMBER OF LINES AS INPUT FROM USER
Date:	

AIM:

To write a Python program that prints a pyramid pattern based on the number of lines provided by the user.

PROCEDURE:

Step 1: Prompt the user to enter the number of lines for the pyramid.

Step 2: Use a loop to iterate through each line. For each line:

- ❖ Print spaces to center-align the pyramid.
- ❖ Print stars (*) in increasing order to form the pyramid shape.

Step 3: Print the pyramid pattern.

Step 4: Display the result.

Source Code :

```
lines = int(input("Enter the number of lines for the pyramid: "))
for i in range(1, lines + 1):
    print(" " * (lines - i), end="")
    print("*" * (2 * i - 1))
```

Output:

```
Enter the number of lines for the pyramid: 5
    *
   ***
  *****
 *******
*********
```

Result:

Thus the python program to print pyramid pattern by taking number of lines as input from user is executed successfully and verified

Ex.No:4	PYTHON PROGRAM TO MANAGE STUDENT INFORMATION IN UNIVERSITIES BY COLLECTING AND STORING DETAILS SUCH AS ID,NAME,ADDRESS AND CONTACT DETAILS
Date:	

AIM:

To design a Python program for managing university student information, including ID, name, address, contact details, scores, and grade evaluation with recommendations.

PROCEDURE:

Step 1: Define a Student class with attributes for ID, name, address, contact, and score.

Step 2: Implement methods to evaluate grades and provide recommendations.

Step 3 : Create a menu-driven interface to add, view, and manage student records.

Step 4 : Store student information in a dictionary for quick retrieval.

Step 5 : Display the result.

Source Code:

```
class Student:
    def __init__(self, id, name, address, contact, score):
        self.id, self.name, self.address, self.contact, self.score = id, name, address, contact, score
        self.grade = self.get_grade()
        self.recommendation = self.get_recommendation()

    def get_grade(self):
        return 'A' if self.score >= 90 else 'B' if self.score >= 80 else 'C' if self.score >= 70 else 'D' if
self.score >= 60 else 'F'

    def get_recommendation(self):
        return {"A": "Excellent!", "B": "Good job!", "C": "Focus on improving.", "D": "Needs
improvement.", "F": "Seek guidance."}[self.grade]

    def display(self):
        print(f"ID: {self.id}, Name: {self.name}, Grade: {self.grade}, Recommendation:
{self.recommendation}")



students = {}

while True:
    choice = input("\n1. Add Student 2. Display Info 3. Exit: ")
    if choice == '1':
        id = input("ID: "); name = input("Name: "); addr = input("Address: "); contact =
input("Contact: ")
        score = float(input("Score: "))
        students[id] = Student(id, name, addr, contact, score)
    elif choice == '2':
        id = input("Enter Student ID: ")
        students.get(id, Student(id, "Not Found", "-", "-", 0)).display()
    elif choice == '3':
        break
    else:
        print("Invalid choice!")
```

Output:

```
--- Student Management System ---
1. Add Student
2. Display Student Info
3. Exit
Enter your choice: 1
Enter Student ID: 101
Enter Name: Alice
Enter Address: 123 Elm St
Enter Contact: 9876543210
Enter Score (0-100): 92
Student added successfully!

--- Student Management System ---
1. Add Student
2. Display Student Info
3. Exit
Enter your choice: 2
Enter Student ID to display info: 101
```

```
ID: 101, Name: Alice, Address: 123 Elm St, Contact: 9876543210, Score: 92, Grade: A
Recommendation: Excellent! Keep up the great work.

--- Student Management System ---
1. Add Student
2. Display Student Info
3. Exit
Enter your choice: 3
Exiting the system. Goodbye!
```

Result :

Thus the Python program for managing university student information, including ID, name, address, contact details, scores, and grade evaluation with recommendations is executed Successfully verified.

Ex.No:5	PYTHON PROGRAM TO ILLUSTRATE SINGLE AND MULTIPLE INHERITANCE
Date:	

AIM:

To demonstrate the concepts of Single Inheritance and Multiple Inheritance in Python by creating classes that inherit properties and behaviors from one or more parent classes.

PROCEDURE:

Single Inheritance:

Step 1: Define the Parent Class: Create a class (e.g., Animal) with some basic methods or attributes.

Step 2: Define the Child Class: Create a class (e.g., Dog) that inherits from the parent class using the syntax class ChildClass(ParentClass).

Step 3: Access Parent Methods: In the child class, call the inherited methods from the parent class to demonstrate single inheritance.

Step 4 : Display the result.

Multiple Inheritance:

Step 1: Define Multiple Parent Classes: Create more than one parent class (e.g., Animal and Pet), each with its own methods or attributes.

Step 2: Define the Child Class: Create a class (e.g., Dog) that inherits from both parent classes using the syntax class ChildClass(ParentClass1, ParentClass2):.

Step 3: Access Methods from Both Parents: In the child class, call the inherited methods from both parent classes to demonstrate multiple inheritance.

Step 4 : Display the result.

Source Code:

Single Inheritance:

```
class Animal:
    def speak(self):
        print("Animal makes a sound")

class Dog(Animal):
    def bark(self):
        print("Dog barks")

dog = Dog()
dog.speak()
dog.bark()
```

Multiple Inheritance:

```
class Animal:
    def speak(self):
        print("Animal makes a sound")

class Pet:
    def play(self):
        print("Pet plays with a ball")

class Dog(Animal, Pet):
    def bark(self):
        print("Dog barks")
```

```
dog = Dog()

dog.speak()

dog.play()

dog.bark()
```

Output:

For Single Inheritance:

```
Animal makes a sound
Dog barks
```

For Multiple Inheritance:

```
Animal makes a sound
Pet plays with a ball
Dog barks
```

Result:

Thus the python program to illustrate single and multiple inheritance are executed successfully and verified

Ex.No:6	PYTHON PROGRAM TO CREATE AN ARRAY ORF DIMENSION (3,5) WITH SCORES OF 5 STUDENTS IN 3 EXAMS AND PERFORMS THE FOLLOWING OPERATIONS FOR ACCESS,EDIT,SEARCH AND UPDATE ELEMENTS
Date:	

AIM:

To create a Python program that manages the scores of 5 students in 3 exams using a 2D array (3x5). The program will provide functionalities to access, edit, search, and update individual scores.

PROCEDURE:

Step 1: Use the numpy library to create a 3x5 array, where each row represents an exam and each column represents a student.

Step 2: Define a function access(exam, student) to retrieve the score of a specific student in a specific exam by indexing the array.

Step 3: Define a function edit(exam, student, new_score) to modify the score of a specific student in a specific exam by directly updating the array element.

Step 4: Define a function search(score) to search for a given score in the array and display its location (exam and student).

Step 5: Define a function update(exam, student, new_score) to update a specific student's score in a specific exam.

Step 6: Implement a loop that displays a menu allowing the user to choose an operation (access, edit, search, update, display, or exit).

Step 7: Prompt the user for input and perform the corresponding operation.

Step 8: Use print(scores) to display the entire array of scores when requested.

Step 9: Display the result.

Source Code:

```
import numpy as np
scores = np.array([[85, 90, 78, 92, 88],
                   [76, 85, 84, 89, 91],
                   [88, 92, 79, 85, 87]])

def access(exam, student):
    return scores[exam, student]

def edit(exam, student, new_score):
    scores[exam, student] = new_score

def search(score):
    result = np.where(scores == score)
    if result[0].size > 0:
        print(f"Score {score} found at Exam {result[0][0]+1}, Student {result[1][0]+1}")
    else:
        print(f"Score {score} not found.")

def update(exam, student, new_score):
    scores[exam, student] = new_score

while True:
    print("\n1. Access Score 2. Edit Score 3. Search Score 4. Update Score 5. Display 6. Exit")
    choice = input("Choice: ")
    if choice == '1':
        exam, student = map(int, input("Exam (1-3), Student (1-5): ").split())
        print(f"Score: {access(exam-1, student-1)}")
    elif choice == '2':
        exam, student, new_score = map(int, input("Exam (1-3), Student (1-5), New Score: ").split())
        edit(exam-1, student-1, new_score)
    elif choice == '3':
        score = int(input("Enter score to search: "))
        search(score)
    elif choice == '4':
        exam, student, new_score = map(int, input("Exam (1-3), Student (1-5), New Score: ").split())
        update(exam-1, student-1, new_score)
    elif choice == '5':
        print(scores)
    elif choice == '6':
        break
    else:
        print("Invalid choice!")
```

Output:

```
1. Access Score 2. Edit Score 3. Search Score 4. Update Score 5. Display 6. Exit
Choice: 1
Exam (1-3), Student (1-5): 1 2
Score: 90
```

Result:

Thus the Python program to create an array of dimension (3,5) with scores of 5 students in 3 exams and performs the following operations for access, edit, search and update elements are executed successfully and verified

Ex.No:7	PYTHON PROGRAM TO CALCULATE SUM OF NUMBERS BY PASSING THEM AS ARGUMENT TO FUNCTION AND RETURN THE RESULT
Date:	

AIM:

To create a Python program that calculates the sum of numbers by passing them as arguments to a function and returns the result.

PROCEDURE:

Step 1:Create a function calculate_sum(*args) that accepts a variable number of arguments using *args.

Step 2:Inside the function, use the built-in sum() function to calculate the sum of the numbers passed as arguments.

Step 3:Call the calculate_sum() function with the desired numbers as arguments.

Step 4:The function will return the sum of the numbers, which can be printed or used elsewhere in the program.

Step 5:Print the returned sum to display the result.

Source Code:

```
def calculate_sum(*args):
    return sum(args)
result = calculate_sum(1, 2, 3, 4, 5)
print("Sum:", result)
```

Output:

```
Sum: 15
```

Result:

Thus the python program to calculate sum of numbers by passing them as argument to function and return the result are executed successfully and verified

Ex.No:8	DIFFERENCE BETWEEN STRUCTURED AND UNSTRUCTURED ARRAYS USING AN EXAMPLE
Date:	

AIM:

To understand the difference between structured arrays and unstructured arrays in Python, and demonstrate their use with examples.

PROCEDURE:

Structured Arrays;

Step 1:A structured array stores data of different types (e.g., integers, floats, strings) in a single array, where each element can have multiple fields with different data types.

Step 2: Display the result.

Unstructured Arrays;

Step 1:An unstructured array stores data of the same type (e.g., integers, floats) in a simple, contiguous block of memory.

Step 2: Display the result

Source Code:

Structured Array

```
import numpy as np

structured_array = np.array([('Alice', 20, 'A'), ('Bob', 22, 'B')],
                            dtype=[('Name', 'U10'), ('Age', 'i4'), ('Grade', 'U1')])
print("Structured Array:", structured_array)
```

Unstructured Array:

```
import numpy as np

unstructured_array = np.array([10, 20, 30, 40, 50])
print("Unstructured Array:", unstructured_array)
```

Ouput:

```
Structured Array: [('Alice', 20, 'A') ('Bob', 22, 'B')]

Unstructured Array: [10 20 30 40 50]
```

Result:

Thus the difference between structured and unstructured arrays using an example is executed successfully and verified