

Task 1: Running Python Script and various expressions in an interactive interpreter

Aim:

To run python Script and various expressions in an interactive interpreter

- a. **Create a python program to enter two numbers and then performs and displays the results of the following operations: addition, subtraction, multiplication, and division.**

Algorithm:

- 1.Start.
- 2.Get the two numbers and store it in variable x and y.
- 3.For Addition do ; x+y and print it.
- 4.For Subtraction do ; x-y and print it.
- 5.For Multiplication do ; x*y and print it.
- 6.For Division do; x/y and print it.
- 7.Stop

Program:

```
x=int(input("Enter the First number:"))
y=int(input("Enter the Second number:"))
```

```
add = x + y
sub = x - y
pro = x * y
div = x / y
```

```
print("Addition:",add)
print("Subtraction :",sub)
print("Multiplication:",pro)
print("Division:",div)
```

Output:

```
===== RESTART: C:\Users\91979\Desktop\t.py
Enter the First number:5
Enter the Second number:6
Addition: 11
Subtraction : -1
Multiplication: 30
Division: 0.8333333333333334
|
```

b. Create a python program to enter two numbers and then performs and displays the results of the following relational expression : >, <, ==, !=, >=, <=

Algorithm:

- 1.Start
- 2.Get the the input from the user and store it in a,b&c.
- 3.Perform the relational operations(i.e, >,<.,==,!=,>=,<=).
- 4.Print the results.
- 5.Stop.

Program:

```
# Initializing the value of a, b, and c
a = int(input("Enter the First number: "))
b = int(input("Enter the Second number: "))
c = int(input("Enter the Third number: "))

# Using relational operators
print(a, ">", b, "is", a > b)
print(a, "<", b, "is", a < b)
print(c, "==", a, "is", c == a)
print(c, "!=", b, "is", c != b)
print(a, ">=", b, "is", a >= b)
print(b, "<=", a, "is", b <= a)
```

Output:

```
===== RESTART: C:\Us
Enter the First number: 5
Enter the Second number: 6
Enter the Third number: 7
5 > 6 is False
5 < 6 is True
7 == 5 is False
7 != 6 is True
5 >= 6 is False
6 <= 5 is False
```

c. Create a python program to enter three numbers and then performs and displays the results of the following Logical operations: and, or, not.

Algorithm:

- 1.Start.
- 2.Get the input from the user.
- 3.Perform the logical operations on the inputs.
- 4.Print the results.
- 5.Stop.

Program:

```
# Taking three numbers as input
a = int(input("Enter the First number: "))
b = int(input("Enter the Second number: "))
c = int(input("Enter the Third number: "))

# Performing logical operations
print("\nLogical Operations Results:")
print((a > b) and (b > c))
print((a > b) or (b > c))
print(not(a > b))
print(not(b > c))
```

Output

```
===== RESTART: C:\Use
Enter the First number: 5
Enter the Second number: 6
Enter the Third number: 7

Logical Operations Results:
False
False
True
True
```

Result

Thus, the python program to run Python Script and various expressions in an interactive interpreter was don successfully and the output was verified.

Task 2: Implement conditional, control and looping statements

AIM

To implement conditional , control and looping statements using python

- a. **You are developing a simple grade management system for a school. The system needs to determine the grade of a student based on their score in a test. The grading system follows these rules:**

If the score is 90 or above, the grade is "A".

If the score is between 80 and 89, the grade is "B".

If the score is between 70 and 79, the grade is "C".

If the score is between 60 and 69, the grade is "D".

If the score is below 60, the grade is "F".

ALGORITHM

1. Start
2. Get the input mark from the user.
3. With the use of an If-elif-else statement do
 - If the marks \geq 90 print grade "A".
 - If the mark is between 80 and 89 print grade "B".
 - If the mark is between 70 and 79 print grade "C".
 - If the mark is between 60 and 69 print grade "D".
 - If the mark is below 60, print grade "F".
4. Stop

PROGRAM

```
score =int(input("Enter the score:"))
```

```
if score $\geq$ 90:
```

```
    print("The Grade is A")
```

```
elif (score  $\leq$ 89 and score $\geq$ 80):
```

```
    print("The Grade is B")
```

```
elif(score  $\leq$ 79 and score  $\geq$ 70):
```

```
    print("The Grade is C")
```

```
elif( score <=69 and score >=60):
```

```
    print("The Grade is D")
```

```
else:
```

```
    print("The Grade is F")
```

Output:

```
===== RESTART: C:\Usr
Enter the score:60
The Grade is D
```

- b. You are developing an educational program to help young students learn about natural numbers. One of the features of the program is to display the first 10 natural numbers to the user. Write a Python program that uses a for loop to print the first 10 natural numbers.

ALGORITHM

- 1.Start.
- 2.Display "The first 10 natural numbers are:".
- 3.Use a for loop for generating the numbers.
- 4.Print the output.
- 5.Stop

PROGRAM

```
# Displaying the first 10 natural numbers
print("The first 10 natural numbers are:")
for i in range(1, 11): # Loop from 1 to 10
    print(i)
```

Output:

```
===== RESTART: C:\Users\919
The first 10 natural numbers are:
1
2
3
4
5
6
7
8
9
10
```

- c. You are working on a feature for a financial application that involves validating user input. One of the requirements is to count the total number of digits in a given number

ALGORITHM

- 1.Start.
2. get the input from the user.
- 3.Convert the integer to string using str().
- 4.Use len function to find number of digits.
- 5.Print the output.

PROGRAM

```
digit=int(input("Enter the Number:"))
string=str(digit) #since integer doesn't have len()
count=len(string)
print("The number of digits in ",digit,"is :",count)
```

Output:

```
===== RESTART: C:\Users\91979\l
```

```
Enter the Number:5
```

```
The number of digits in 5 is : 1
```

```
===== RESTART: C:\Users\91979\l
```

```
Enter the Number:55
```

```
The number of digits in 55 is : 2
```

Task 2

2.1 Develop a simple program for the Air Force to label an aircraft as military or civilian. The program is to be given the plane's observed speed in km/h (kilometer per hour). The speed will serve as its input. For planes traveling in excess of 1100 km/h, you should display them as "It's a civilian aircraft", between 500 km/h to 1100 km/h, display them as "It's a military aircraft!" and for planes traveling at more slower speed – less than 500 km/h, you should display them as an "It's a BIRD!".

Input & Output:

2

1500

It's a civilian aircraft

200

It's a BIRD!

2.2 The National Earthquake Information Center has the following criteria to determine the earthquake's damage. Here are the given richter scale criteria and their corresponding characterization. The richter scale serves as the input data and the characterization as output information. Use the ladderized if / else if / else conditional statement.

Richter Numbers (n) Characterization

$n < 5.0$ Little or no damage

$5.0 \leq n < 5.5$ Some damage

$5.5 \leq n < 6.5$ Serious damage

$6.5 \leq n < 7.5$ Disaster

higher Catastrophe

Input & Output:

2

6

Serious damage

2

Little or no damage

RESULT

Thus, the python program to implement conditional, control and looping statements was done successfully.

Task 3: Importing Python modules and packages in python programming

Aim:

To write python demonstrating importing Python modules and packages

- a. **You are tasked with developing a modular calculator application in Python. The calculator should support basic arithmetic operations: addition, subtraction, multiplication, and division. Each operation should be implemented in a separate module. Additionally, you should create a main program to handle user input, call the appropriate module, and display the results.**

Algorithm:

1. Define functions for addition, subtraction, multiplication, and division.
2. Handle division by zero by raising an error if the divisor is zero.
3. Import the module (mymath) containing these functions.
4. Initialize two numbers (a = 10, b = 5).
5. Call each function using mymath.<function_name>(a, b).

6. Print the results of all operations.

Program:

```
def add(a, b):  
    return a + b  
def subtract(a, b):  
    return a - b  
def multiply(a, b):  
    return a * b  
def divide(a, b):  
    if b == 0:  
        raise ValueError("Cannot divide by zero")  
    return a / b  
  
import mymath  
  
a = 10  
b = 5  
print("Addition:", mymath.add(a, b))  
print("Subtraction:", mymath.subtract(a, b))  
print("Multiplication:", mymath.multiply(a, b))  
print("Division:", mymath.divide(a, b))
```

Output:

```
===== RESTART: C:  
Addition: 15  
Subtraction: 5  
Multiplication: 50  
Division: 2.0
```

- b. You are working on a Python project that requires you to perform various mathematical operations and geometric area calculations. To organize your code better, you decide to create a package named mypackage which includes sub packages pack1 and pack 2 with two modules: mathfunctions and areafunctions Demonstrate the use of the functions by performing a few calculations and printing the results.

Algorithm:

1. Create mathfunctions.py module:

2. Create areafunctions.py module:
3. Create __init__.py files in pack1 and pack2:
4. Create main.py:
5. Print the output as expected.

Program:

1. Create the mathfunctions.py module

```
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b == 0:
        return "Error! Division by zero."
    return a / b
```
2. Create the areafunctions.py module

```
import math
def circle_area(radius):
    return math.pi * radius * radius
def rectangle_area(length, width):
    return length * width
def triangle_area(base, height):
    return 0.5 * base * height
```
3. Create __init__.py in each package folder (pack1 and pack2)

```
from .mathfunctions import add, subtract, multiply, divide
from .areafunctions import circle_area, rectangle_area, triangle_area
```
4. Create the main.py file

```
from pack import mathfunctions
from pack import areafunctions
# Using math functions
print("Addition:", mathfunctions.add(10, 5))
print("Subtraction:", mathfunctions.subtract(10, 5))
print("Multiplication:", mathfunctions.multiply(10, 5))
print("Division:", mathfunctions.divide(10, 5))
# Using area functions
print("Circle Area (radius=7):", areafunctions.circle_area(7))
print("Rectangle Area (5x10):", areafunctions.rectangle_area(5, 10))
print("Triangle Area (base=6, height=8):", areafunctions.triangle_area(6, 8))
```

Output:

```
===== RESTART: C:/Users/91979/Desktop
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Circle Area (radius=7): 153.93804002589985
Rectangle Area (5x10): 50
Triangle Area (base=6, height=8): 24.0
```

Result:

Thus, the program for Importing Python modules and packages was successfully executed and the output was verified.

Task 4. Use various data types, List, Tuples and Dictionary in python programming**Aim:**

To use various data types, List, Tuples and Dictionary in python programming

- a. **You are working on a Python project that requires you to manage and manipulate a list of numbers. Your task is to create a Python program that demonstrates the following list operations:**
 1. **Add Elements:** Add elements to the list.
 2. **Remove Elements:** Remove specific elements from the list.
 3. **Sort Elements:** Sort the list in ascending and descending order.
 4. **Find Minimum and Maximum:** Find the minimum and maximum elements in the list.
 5. **Calculate Sum and Average:** Calculate the sum and average of the elements in the list.

ALGORITHM

1. Start

2. For adding elements to a list first create a list with name "list" and assign the values within [] brackets , inorder to add a new value use the function append().
3. For removing a specific element use "pop(index value)" or "remove(itemname)".
4. For sorting the elements use "sorted(list)" function.
5. For finding minimum value use "min(list)" and for maximum use "max(list)".
6. For sum use function "sum(list)" and for average use the formula "sum(list)/len(list)"
7. Print the output.
8. End.

PROGRAM

#Add Elements: Add elements to the list.

```
list=[10,20]
```

```
a=30
```

```
list.append(a)
```

```
print(list)
```

#Remove Elements: Remove specific elements from the list.

```
list.pop(1)#by index value
```

```
print(list)
```

```
list.remove(10)#by itemname
```

```
print(list)
```

#Sort Elements: Sort the list in ascending and descending order.

```
l=[5,8,9,15,30,89]
```

```
print(sorted(l))
```

#Find Minimum and Maximum: Find the minimum and maximum elements in the list.

```
print("The minimum value is:",min(l))
```

```
print("The maximum value is:",max(l))
```

#Calculate Sum and Average

```
print("The sum is:",sum(l))
```

```
print("The average is :",(sum(l)/len(l)))
```

OUTPUT

```
== RESTART: C:/Users/Student
[10, 20, 30]
[10, 30]
[30]
[5, 8, 9, 15, 30, 89]
The minimum value is: 5
The maximum value is: 89
The sum is: 156
The average is : 26.0
```

b. You are tasked with creating a Python program that showcases operations on tuples. Tuples are immutable sequences, similar to lists but with the key difference that they cannot be changed after creation. Your program should illustrate the following tuple operations:

1. **Create a Tuple:** Define a tuple with elements of different data types(10, 'hello', 3.14, 'world')
2. **Access Elements:** Access individual elements and slices of the tuple.
3. **Concatenate Tuples:** Combine two tuples to create a new tuple.
4. **Immutable Nature:** Attempt to modify elements of the tuple and handle the resulting error.

ALGORITHM

1. Start.
2. To create a tuple use "tuple_name=(values)".
3. To access the elements of a tuple either use the index values (tuple_name(index_value)) or the tuple slicing (tuple_name[start:end]).
4. To concatenate tuples use the operator "+" (tuple1 "+" tuple2).
5. Try to modify the tuple elements by assigning the values directly like ; tuple(index)= new_value , will result in an error as it is immutable.
6. Print the output.
7. End.

PROGRAM

```
#Create a Tuple: Define a tuple with elements of different data types(10, 'hello',
3.14, 'world')
tuple=(10, 'hello', 3.14, 'world')
print(tuple)
#Access Elements: Access individual elements and slices of the tuple.
for i in tuple:
    print(i)
print(tuple[1:3])
print(tuple[:-1])
#Concatenate Tuples: Combine two tuples to create a new tuple.
t2=(5,0.5)
t3=tuple+t2
print(t3)
#Immutable Nature: Attempt to modify elements of the tuple and handle the
resulting error.
tuple(3)="PI" #ERROR
```

OUTPUT

```
== RESTART: C:/Users/Student.D
(10, 'hello', 3.14, 'world')
10
hello
3.14
world
('hello', 3.14)
(10, 'hello', 3.14)
|
```

- c. You are tasked with creating a Python program that showcases operations on dictionaries. Dictionaries in Python are unordered collections of items. Each item is a pair consisting of a key and a value. Your program should illustrate the following dictionary operations:
1. **Create a Dictionary:** Define a dictionary with key-value pairs of different data types.({'name': 'Alice', 'age': 30, 'city': 'New York'})
 2. **Access Values:** Access values using keys.
 3. **Modify Dictionary:** Update values, add new key-value pairs, and remove existing pairs.
 4. **Iterate Over Dictionary:** Use loops to iterate over keys or values.

Algorithm

1. Start the program
2. Define a dictionary with key-value pairs of different data types.
3. Retrieve values from the dictionary using their corresponding keys.
4. Modify Dictionary
5. Iterate Over Dictionary
6. Stop the program.

PROGRAM

```
#Create a Dictionary: Define a dictionary with key-value pairs of different data
types.({'name': 'Alice', 'age': 30, 'city': 'New York'})
dictionary={'name': 'Alice', 'age': 30, 'city': 'New York'}
print(dictionary)
#Access Values: Access values using keys.
print(dictionary['name'])
print(dictionary['age'])

#Modify Dictionary: Update values, add new key-value pairs, and remove
existing pairs.
dictionary['name']= "James"
print(dictionary)
dictionary.pop('city')
print(dictionary)
#Iterate Over Dictionary: Use loops to iterate over keys or values.
for k in dictionary:
    print("KEY:",k)

print(dictionary.items())
```

OUTPUT

```
== RESTART: C:/Users/Student.DESKTOP-4SNH6EA/Desktop/
{'name': 'Alice', 'age': 30, 'city': 'New York'}
Alice
30
{'name': 'James', 'age': 30, 'city': 'New York'}
{'name': 'James', 'age': 30}
KEY: name
KEY: age
dict_items([('name', 'James'), ('age', 30)])
```

RESULT

Thus, various data types, List, Tuples and Dictionary in python programming was used and verified successfully.

Task - 5.Implement various Searching and Sorting Operations in python programming

Aim:

To Implement various Searching and Sorting Operations in python programming.

5.1. A company stores employee records in a list of dictionaries, where each dictionary contains id, name, and department. Write a function find_employee_by_id that takes this list and a target employee ID as arguments and returns the dictionary of the employee with the matching ID, or None if no such employee is found.

Algorithm:

1. Input Definition:
2. Define the function find_employee_by_id that takes two parameters:
 - a. A list of dictionaries (employees), where each dictionary represents an employee record with keys id, name, and department.
 - b. An integer (target_id) representing the employee ID to be searched.
3. Iterate Through the List:

- Use a for loop to iterate through each dictionary in the employees list.
4. Check for Matching ID:

Within the loop, check if the id field of the current dictionary matches the target_id.
 5. Return Matching Record:

If a match is found, return the current dictionary.
 6. Handle No Match:

If the loop completes without finding a match, return None.

Program 5.1

```
def find_employee_by_id(employees, target_id):  
    for employee in employees:  
        if employee['id'] == target_id:  
            return employee  
    return None  
  
# Test the function  
employees = [  
    {'id': 1, 'name': 'Alice', 'department': 'HR'},  
    {'id': 2, 'name': 'Bob', 'department': 'Engineering'},  
    {'id': 3, 'name': 'Charlie', 'department': 'Sales'},  
]  
print(find_employee_by_id(employees, 2)) # Output: {'id': 2, 'name': 'Bob',  
'department': 'Engineering'}
```

Output:

```
===== RESTART: C:/Users/91979/Desktop/print1/t3.py  
{'id': 2, 'name': 'Bob', 'department': 'Engineering'}
```

5.2. You are developing a grade management system for a school. The system maintains a list of student records, where each record is represented as a dictionary containing a student's name and score. The school needs to generate a report that displays students' scores in ascending order. Your task is to implement a feature that sorts the student records by their scores using the Bubble Sort algorithm.

Algorithm:

1. Initialization:
 - Get the length of the students list and store it in n.
2. Outer Loop:
 - Iterate from i = 0 to n-1 (inclusive). This loop represents the number of passes through the list.

3. Track Swaps:

- Initialize a boolean variable swapped to False. This variable will track if any swaps are made in the current pass.

4. Inner Loop:

- Iterate from $j = 0$ to $n-i-2$ (inclusive). This loop compares adjacent elements in the list and performs swaps if necessary.

5. Compare and Swap:

- For each pair of adjacent elements (i.e., `students[j]` and `students[j+1]`):
 - Compare their score values.
 - If `students[j]['score'] > students[j+1]['score']`, swap the two elements.
 - Set swapped to True to indicate that a swap was made.

6. Early Termination:

- After each pass of the inner loop, check if swapped is False. If no swaps were made during the pass, the list is already sorted, and you can break out of the outer loop early.

7. Completion:

- The function modifies the students list in place, sorting it by score.

Program 5.2

```
def bubble_sort_scores(students):
    n = len(students)
    for i in range(n):
        # Track if any swap is made in this pass
        swapped = False
        for j in range(0, n-i-1):
            if students[j]['score'] > students[j+1]['score']:
                # Swap if the score of the current student is greater than the next
                students[j], students[j+1] = students[j+1], students[j]
                swapped = True
        # If no two elements were swapped, the list is already sorted
        if not swapped:
            break
# Example usage
students = [
    {'name': 'Alice', 'score': 88},
    {'name': 'Bob', 'score': 95},
    {'name': 'Charlie', 'score': 75},
    {'name': 'Diana', 'score': 85}
]
print("Before sorting:")
for student in students:
    print(student)
```

```
bubble_sort_scores(students)
print("\nAfter sorting:")
for student in students:
    print(student)
```

Output:

```
===== RESTART: C:/Users/91979/Desktop/print1/t3.py
Before sorting:
{'name': 'Alice', 'score': 88}
{'name': 'Bob', 'score': 95}
{'name': 'Charlie', 'score': 75}
{'name': 'Diana', 'score': 85}

After sorting:
{'name': 'Charlie', 'score': 75}
{'name': 'Diana', 'score': 85}
{'name': 'Alice', 'score': 88}
{'name': 'Bob', 'score': 95}
```

Result:

Thus, the Program for various Searching and Sorting Operations is executed and verified successfully.

Task 6.Implement various text file operation

Aim:

To write a python program Implement various text file operations

Problem 6.1:

You need to write the sentence "Error objects are thrown when runtime errors occur. The Error object can also be used as a base object for user-defined exceptions" into a text file named log.txt. Implement a function that performs this task.

Algorithm:

1. Write to a File:

- Define writefile(filename) function:
 - Open a file named "log.txt" in write mode.
 - Write the following text to the file:
 - "Error objects are thrown when runtime errors occur. The Error object can also be used as a base object for user-defined exceptions"
 - Close the file.

2. Read from a File:

- Define `readfile(filename)` function:
 - Open the file specified by filename in read mode using a with statement.
 - Read the entire content of the file.
 - Print the content.

3. Execute the Program:

- Call `writefile("write")` to write the predefined text to "log.txt".
- Call `readfile("text")` to attempt to read from a file named "text" and print its content.

Program 6.1

```
def writefile(filename):
    f=open("log.txt ","w")
    f.write("Error objects are thrown when runtime errors occur. The Error object
    can also be used as a base object for user-defined exceptions ")
    f.close()
def readfile(filename):
    with open(filename, "r") as file:
        content = file.read()
        print(content)
writefile("write")
readfile("text")
```

Output:

```
===== RESTART: C:/Users/91979/Desktop/S1L5/6a.py =====
Error objects are thrown when runtime errors occur. The Error
object can also be used as a base object for user-defined exc
eptions
```

Problem 6.2.

You have a text file log.txt containing logs of a system. Write a function that counts the number of lines containing the word "ERROR".

Algorithm:

1. Initialize Error Counter:

- Define the function `count_error_lines(filename)`:
 - Initialize `error_count` to 0.

2. Open and Read File:

- Open the file specified by filename in read mode using a with statement.

3. Check Each Line for "ERROR":

- Loop through each line in the file:
 - If the line contains the word "ERROR", increment error_count by 1.

4. Return Error Count:

- After reading all the lines, return the value of error_count.

5. Execute the Program:

- Call count_error_lines("log.txt") to count the number of lines with the word "ERROR" in the file "log.txt".
- Print the result with the message: "Number of lines with 'ERROR': {error_lines}".

Program 6.3:

```
def count_error_lines(filename):
    error_count = 0
    with open(filename, "r") as file:
        for line in file:
            if "ERROR" in line:
                error_count += 1
    return error_count
error_lines = count_error_lines("log.txt")
print(f"Number of lines with 'ERROR': {error_lines}")
```

log.txt

"Error" objects are thrown when runtime **Error** occur.

The **Error** object can also be used as a base object for user-defined exceptions."

Output:

```
===== RESTART: C:/Users/91979/Desktop/S1L5/6a.py :
Number of lines with 'ERROR' is 2
```

Problem 6.3:

You need to write a report containing the details (Name, departments) of the employee in list. Write a Python function that writes this report to a file named employee_report.txt

Algorithm:

1. Create Employee Data:
 - Define the function write_employee_report(filename):
 - Create a list employees containing dictionaries, each with "name" and "department" keys for individual employees.
2. Open File for Writing:

- Open the file specified by filename in write mode using a with statement.
3. Write Employee Data to File:
- Loop through each employee in the employees list:
 - For each employee, format a string as "Name: {employee['name']}, Department: {employee['department']]".
 - Write the formatted string to the file, followed by a newline character (\n).
4. Execute the Program:
- Call write_employee_report("employee_report.txt") to write the employee data to the file "employee_report.txt".

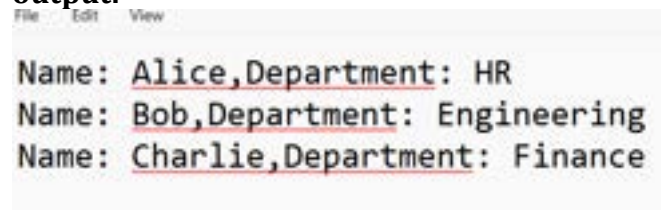
Program 6.3:

```
def write_employee_report(filename):
    employees = [
        {"name": "Alice", "department": "HR"},
        {"name": "Bob", "department": "Engineering"},
        {"name": "Charlie", "department": "Finance"}
    ]

    with open(filename, "w") as file:
        for employee in employees:
            line = f"Name: {employee['name']}, Department: {employee['department']}\n"
            file.write(line)

# Example usage:
write_employee_report("employee_report.txt")
```

output:



```
File Edit View
Name: Alice, Department: HR
Name: Bob, Department: Engineering
Name: Charlie, Department: Finance
```

Result:

Thus, the python program Implement various text file operations was successfully executed and the output was verified.

Task No: 7. Utilizing 'Functions' concepts in Python Programming.**Aim:**

To write the python program using 'Functions' concepts in Python Programming
7.1. You are developing a small Python script to analyze and manipulate a list of student grades for a class project. Write a Python program that satisfies the above requirements using the built-in functions print(), len(), type(), max(), min(), sorted(), reversed(), and range().

Algorithm:

1. Start the program
2. Print a welcome message: Outputs a simple greeting.
3. Determine and print the number of students: Uses len() to find the number of elements in the student_names list.
4. Print the type of lists: Uses type() to show the type of the student_names and student_grades lists.
5. Find and print highest and lowest grades: Uses max() and min() to determine the highest and lowest values in student_grades.
6. Print sorted list of grades: Uses sorted() to sort the grades.
7. Print reversed list of grades: Uses reversed() to reverse the sorted list and converts it to a list.
8. Generate and print a range of grade indices: Uses range() to create a list of indices from 1 to the number of students.
9. Stop

Program:

```
def analyze_student_grades():
    # Sample data
    student_names = ["Alice", "Bob", "Charlie", "Diana"]
    student_grades = [85, 92, 78, 90]

    # 1. Print a welcome message
    print("Welcome to the Student Grades Analyzer!\n")

    # 2. Determine and print the number of students
    num_students = len(student_names)
    print("Number of students:", num_students)

    # 3. Print the type of the student names list and the grades list
    print("\nType of student_names list:", type(student_names))
    print("Type of student_grades list:", type(student_grades))

    # 4. Find and print the highest and lowest grade
    highest_grade = max(student_grades)
    lowest_grade = min(student_grades)
    print("\nHighest grade:", highest_grade)
    print("Lowest grade:", lowest_grade)

    # 5. Print the list of grades sorted in ascending order
    sorted_grades = sorted(student_grades)
    print("\nSorted grades:", sorted_grades)

    # 6. Print the list of grades in reverse order
    reversed_grades = list(reversed(sorted_grades))
    print("Reversed grades:", reversed_grades)

    # 7. Generate and print a range of grade indices from 1 to the number of students
    grade_indices = list(range(1, num_students + 1))
    print("\nGrade indices from 1 to number of students:", grade_indices)

    # Run the analysis
    analyze_student_grades()
```

Output:


```

===== RESTART: C:/Users/91979/Desktop/print1/t3.py =====
Welcome to the Student Grades Analyzer!

Number of students: 4

Type of student_names list: <class 'list'>
Type of student_grades list: <class 'list'>

Highest grade: 92
Lowest grade: 78

Sorted grades: [78, 85, 90, 92]
Reversed grades: [92, 90, 85, 78]

Grade indices from 1 to number of students: [1, 2, 3, 4]

```

7.2. You are tasked with creating a small calculator application to help users perform basic arithmetic operations and greet them with a personalized message. Your application should perform the following tasks: addition, subtraction, multiplication, division.

Algorithm:

1. Start the program
2. User Input for Numbers: The program prompts the user to enter two numbers.
3. User Input for Operation: The program prompts the user to choose an arithmetic operation (addition, subtraction, multiplication, division).
4. Perform Operation: Based on the user's choice, the program performs the chosen arithmetic operation using the defined functions.
5. Display Result: The program displays the result of the operation.
6. Stop

7.2.Program:

```

def add(a, b):
    """Return the sum of two numbers."""
    return a + b
def subtract(a, b):
    """Return the difference between two numbers."""
    return a - b
def multiply(a, b):
    """Return the product of two numbers."""
    return a * b
def divide(a, b):
    """Return the quotient of two numbers. Handles division by zero."""
    if b != 0:
        return a / b
    else:
        return "Error: Division by zero"
def greet(name):
    """Return a greeting message for the user."""
    return f'Hello, {name}! Welcome to the program.'
def main():

```

```

# Demonstrating the use of user-defined functions
# Arithmetic operations
num1 = 10
num2 = 5
print("Arithmetic Operations:")
print(f"Sum of {num1} and {num2}:", add(num1, num2))
print(f"Difference between {num1} and {num2}:", subtract(num1, num2))
print(f"Product of {num1} and {num2}:", multiply(num1, num2))
print(f"Quotient of {num1} and {num2}:", divide(num1, num2))
# Greeting the user
user_name = "Alice"
print("\nGreeting:")
print(greet(user_name))

# Run the main function
if __name__ == "__main__":
    main()

```

Output:

```

===== RESTART: C:/Users/91979/Desktop/print1/t3.py
Arithmetic Operations:
Sum of 10 and 5: 15
Difference between 10 and 5: 5
Product of 10 and 5: 50
Quotient of 10 and 5: 2.0

Greeting:
Hello, Alice! Welcome to the program.

```

Result:

Thus, the python program using 'Functions' concepts was successfully executed and the output was verified.

Task 8.Implement python generator and decorators C01-K3

Aim:

Write a python program to Implement python generator and decorators

8.1 Write a Python program that includes a generator function to produce a sequence of numbers. The generator should be able to:

- a. Produce a sequence of numbers when provided with start, end, and step values.**
- b. Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are provided.**

Produce a sequence of numbers when provided with start, end, and step values.

Algorithm:

- 1. Define Generator Function:**

- Define the function `number_sequence(start, end, step=1)`.
- 2. Initialize Current Value:**
 - Set current to the value of start.
- 3. Generate Sequence:**
 - While current is less than or equal to end:
 - Yield the current value of current.
 - Increment current by step.
- 4. Get User Input:**
 - Read the starting number (start) from user input.
 - Read the ending number (end) from user input.
 - Read the step value (step) from user input.
- 5. Create Generator Object:**
 - Create a generator object by calling `number_sequence(start, end, step)` with user-provided values.
- 6. Print Generated Sequence:**
 - Iterate over the values produced by the generator object.
 - Print each value.

8.1. Program:

```
def number_sequence(start, end, step=1):
    current = start
    while current <= end:
        yield current
        current += step
start = int(input("Enter the starting number: "))
end = int(input("Enter the ending number: "))
step = int(input("Enter the step value: "))
# Create the generator
sequence_generator = number_sequence(start, end, step)
# Print the generated sequence of numbers
for number in sequence_generator:
    print(number)
```

Output:

```
Enter the starting number: 1
Enter the ending number: 50
Enter the step value: 5
1
6
```

11
16
21
26
31
36
41
46

Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are provided.

Algorithm:

1. Start Function:

- Define the function `my_generator(n)` that takes a parameter `n`.

2. Initialize Counter:

- Set value to 0.

3. Generate Values:

- While value is less than `n`:
 - Yield the current value.
 - Increment value by 1.

4. Create Generator Object:

- Call `my_generator(11)` to create a generator object.

5. Iterate and Print Values:

- For each value produced by the generator object:
 - Print value.

8.1.(b)Program:

```
def my_generator(n):  
    # initialize counter  
    value = 0  
    # loop until counter is less than n  
    while value < n:  
        # produce the current value of the counter  
        yield value  
        # increment the counter  
        value += 1  
# iterate over the generator object produced by my_generator  
for value in my_generator(3):  
    # print each value produced by generator  
    print(value)
```

Output:

0
1

8.2. *Imagine you are working on a messaging application that needs to format messages differently based on the user's preferences. Users can choose to have their messages automatically converted to uppercase (for emphasis) or to lowercase (for a softer tone). You are provided with two decorators: `uppercase_decorator` and `lowercase_decorator`. These decorators modify the behavior of the functions they decorate by converting the text to uppercase or lowercase, respectively. Write a program to implement it.*

Algorithm:

1. Create Decorators:

- Define `uppercase_decorator` to convert the result of a function to uppercase.
- Define `lowercase_decorator` to convert the result of a function to lowercase.

2. Define Functions:

- Define `shout` function to return the input text. Apply `@uppercase_decorator` to this function.
- Define `whisper` function to return the input text. Apply `@lowercase_decorator` to this function.

3. Define Greet Function:

- Define `greet` function that:
 - Accepts a function (`func`) as input.
 - Calls this function with the text "Hi, I am created by a function passed as an argument."
 - Prints the result.

4. Execute the Program:

- Call `greet(shout)` to print the greeting in uppercase.
- Call `greet(whisper)` to print the greeting in lowercase.

Program:

```
def uppercase_decorator(func):
    def wrapper(text):
        return func(text).upper()
    return wrapper

def lowercase_decorator(func):
    def wrapper(text):
        return func(text).lower()
```

```

    return wrapper

@uppercase_decorator
def shout(text):
    return text

@lowercase_decorator
def whisper(text):
    return text

def greet(func):
    greeting = func("Hi, I am created by a function passed as an argument.")
    print(greeting)

greet(shout)
greet(whisper)

```

Output:

HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT.
hi, i am created by a function passed as an argument.

Result:

Thus the python program to Implement python generator and decorators was successfully executed and the output was verified.

Task 9: Implement Exceptions and Exceptional handling in Python.

Aim:

To implement Exceptions and Exceptional handling in Python.

Problem 9.1.*You are developing a Python program that processes a list of students' grades. The program is designed to allow the user to select a grade by specifying an index number. However, you need to ensure that the program handles cases where the user inputs an index that is out of range, i.e., an index that does not exist in the list.*

Algorithm:

1. Start the program
2. Initializes a list of grades (e.g., [85, 90, 78, 92, 88]).
3. Prompts the user to enter the index of the grade they wish to view.
4. Attempts to display the grade at the specified index.
5. If the index is out of range, catches the IndexError and prints an error message,

"Invalid index. Please enter a valid index."

Program:

```
# Initialize the list of grades
grades = [85, 90, 78, 92, 88]

# Display the grades list
print("Grades List:", grades)

# Prompt the user to enter the index of the grade they want to view
try:
    index = int(input("Enter the index of the grade you want to view: "))
    # Attempt to display the grade at the specified index
    print(f"The grade at index {index} is: {grades[index]}")
except IndexError:
    # Handle the case where the index is out of range
    print("Invalid index. Please enter a valid index.")
except ValueError:
    # Handle the case where the input is not an integer
    print("Invalid input. Please enter a numerical index.")
```

Output:

```
Grades List: [85, 90, 78, 92, 88]
Enter the index of the grade you want to view: 10
Invalid index. Please enter a valid index.
```

Problem 9.2. You are developing a Python calculator program that performs basic arithmetic operations. One of the key functionalities is to divide two numbers entered by the user. However, dividing by zero is not allowed and would cause the program to crash if not handled properly.

Algorithm:

1. Start the program
2. Prompts the user to enter two numbers: a numerator and a denominator.
3. Attempts to divide the numerator by the denominator.
4. If the denominator is zero, catches the `ZeroDivisionError` and displays an error message: "Error: Division by zero is not allowed."

Program:

```
# Function to perform division
def divide_numbers():
```

```

try:
    # Prompt the user to enter the numerator
    numerator = float(input("Enter the numerator: "))
    # Prompt the user to enter the denominator
    denominator = float(input("Enter the denominator: "))
    # Attempt to perform division
    result = numerator / denominator
    print(f"Result: {result}")
except ZeroDivisionError:
    # Handle division by zero error
    print("Error: Division by zero is not allowed.")
except ValueError:
    # Handle invalid input that is not a number
    print("Error: Please enter valid numbers.")

# Call the function to execute the division operation
divide_numbers()

```

Output:

```

Enter the numerator: 10
Enter the denominator: 0
ERROR!
Error: Division by zero is not allowed.

```

Problem 9.3: You are building a Python application to determine if a person is eligible to vote based on their age. According to the rules, only individuals who are 18 years or older are allowed to vote. To enforce this rule, you decide to create a custom exception called `InvalidAgeException`, which will be raised whenever an age below 18 is entered.

Algorithm:

1. Define the custom exception.
2. Prompt the user for input.
3. Check if the age is below 18.
4. Raise an exception if the condition is met.
5. Handle the exception with a custom error message.

Program:

```

# define Python user-defined exceptions
class InvalidAgeException(Exception):
    "Raised when the input value is less than 18"

```



```

    pass
# you need to guess this number
number = 18

try:
    input_num = int(input("Enter a number: "))
    if input_num < number:
        raise InvalidAgeException
    else:
        print("Eligible to Vote")

except InvalidAgeException:
    print("Exception occurred: Invalid Age")

```

Output:

Enter a number: 15
Exception occurred: Invalid Age

Result:

Thus the program for Implement Exceptions and Exceptional handling is executed and verified successfully

Task 10. Use Matplotlib module for plotting in python

Aim:

To use Matplotlib module for plotting in python.

Problem 10.1. Write a Python programming to display a bar chart of the popularity of programming Languages.

Sample data:

Programming languages: Java, Python, PHP, JavaScript, C#, C++
Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

Sample Output:



Algorithm:

1. Define two lists for programming languages and their popularity respectively
2. Find the maximum popularity value in the list
3. Define a scaling factor to scale the bar heights within a certain limit (e.g. 50 characters)
4. For each language and popularity pair, calculate the bar height as the popularity value scaled by the scaling factor
5. Print the chart using a loop to iterate over the programming language list:
 - a. Print the language name and a separator character (e.g. "|")
 - b. Use a loop to print the bar chart by printing the bar character (e.g. "*") a number of times equal to the bar height
 - c. Print the popularity value with a separator character
 - d. Print a newline character

Program:

#pip install matplotlib

```
import matplotlib.pyplot as plt
```

```
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
```

```
plt.bar(languages, popularity, color='b')
plt.title('Popularity of Programming Languages')
plt.xlabel('Programming Languages')
plt.ylabel('Popularity')
plt.show()
```

Output:



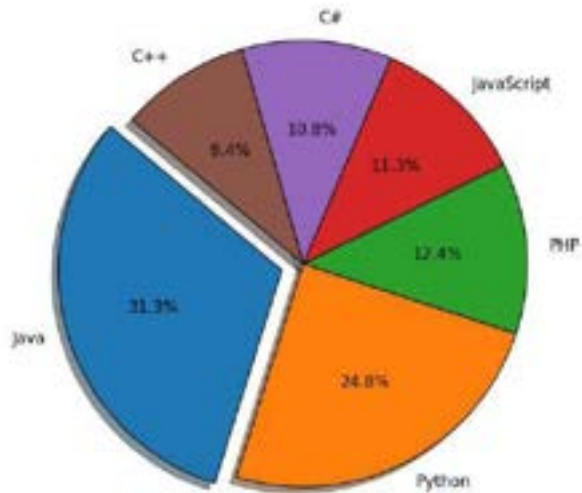
Problem 10.2. Write a Python programming to create a pie chart of the popularity of programming Languages.

Sample data:

Programming languages: Java, Python, PHP, JavaScript, C#, C++

Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

Sample Output:



Algorithm:

1. Create a list of Programming Languages and Popularity
2. Create a pie chart using the matplotlib library
3. Set the title and legend for the pie chart
4. Show the pie chart

Program:

```
import matplotlib.pyplot as plt

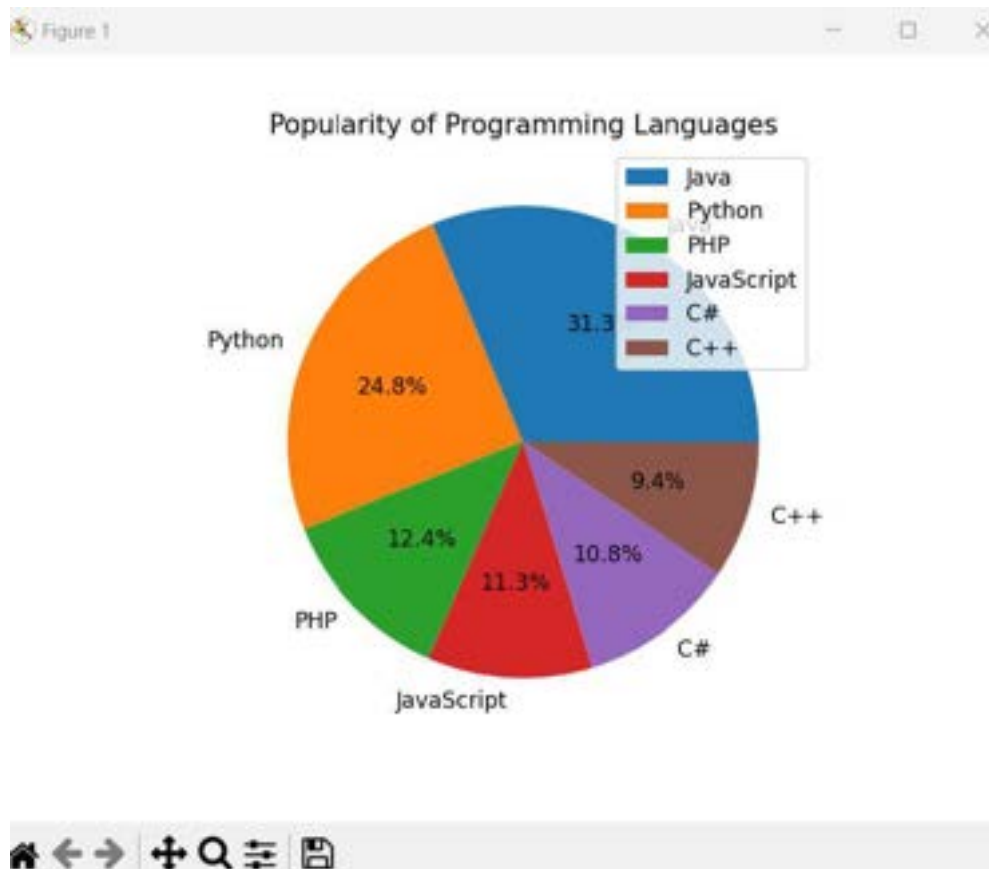
# Step 1
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]

# Step 2
plt.pie(popularity, labels=languages, autopct='%1.1f%%')
```

```
# Step 3
plt.title('Popularity of Programming Languages')
plt.legend(languages, loc="best")
```

```
# Step 4
plt.show()
```

Output:



Result: Thus the python program use Matplotlib module for plotting is executed and verified successful.

Task 11.Use Tkinter module for UI design

Aim:

To use Tkinter module for UI design

Problem 11.1. Write a Python GUI program to create a label and change the label font style (font name, bold, size) using tkinter module.

**Algorithm:**

1. Import tkinter module
2. Create a main window
3. Create a label with desired text
4. Add the label to the main window using pack() method
5. Define a function to change font style
6. Create a button to call the function when clicked
7. Add the button to the main window using pack() method
8. Start the main loop.

Program:

```
import tkinter as tk

# Function to change font style
def change_font():
    label.config(font=("Arial", 18, "bold"))

# Create main window
root = tk.Tk()

# Create label with desired text
label = tk.Label(root, text="Hello, World!", font=("Helvetica", 14))

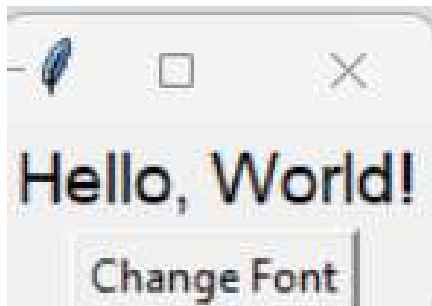
# Add label to main window
label.pack()

# Create button to change font style
button = tk.Button(root, text="Change Font", command=change_font)

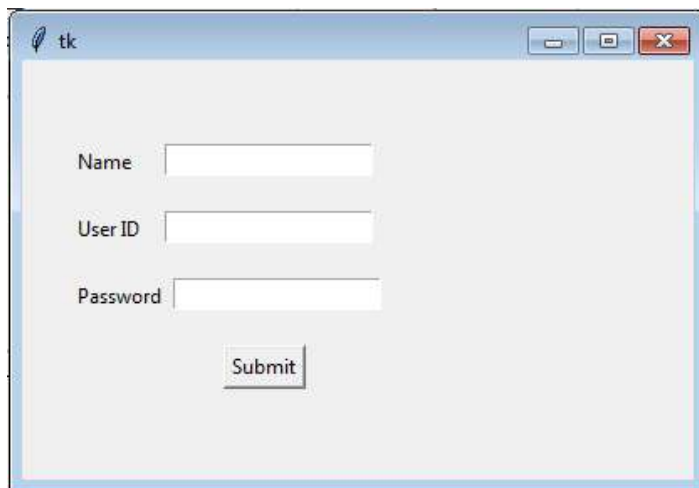
# Add button to main window
button.pack()

# Start the main loop
root.mainloop()
```

Output:



Task 11.2: Write a Python GUI program to create three single line text-box to accept a value from the user using tkinter module.



Algorithm:

1. Import the tkinter module
2. Create the main window
3. Add labels and text-boxes to the main window
4. Set the size of the text-boxes
5. Create a button to submit the values entered in the text-boxes
6. Get the values entered in the text-boxes when the button is clicked
7. Close the main window when the button is clicked

Program:

```
import tkinter as tk
```

```
# Create the main window
```

```
root = tk.Tk()
```

```
root.title("Text-Box Input")
```

```
# Create labels and text-boxes
```

```
label1 = tk.Label(root, text="Enter value 1:")
```

```

entry1 = tk.Entry(root)

label2 = tk.Label(root, text="Enter value 2:")
entry2 = tk.Entry(root)

label3 = tk.Label(root, text="Enter value 3:")
entry3 = tk.Entry(root)

# Set the size of the text-boxes
entry1.config(width=30)
entry2.config(width=30)

entry3.config(width=30)

# Create a function to get the values entered in the text-boxes
def get_values():
    val1 = entry1.get()
    val2 = entry2.get()
    val3 = entry3.get()
    print("Value 1:", val1)
    print("Value 2:", val2)
    print("Value 3:", val3)

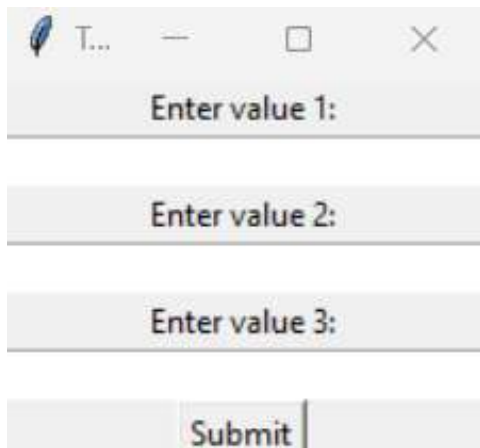
# Create a button to submit the values entered in the text-boxes
submit_button = tk.Button(root, text="Submit", command=get_values)

# Add the labels, text-boxes, and button to the main window
label1.pack()
entry1.pack()
label2.pack()
entry2.pack()
label3.pack()
entry3.pack()
submit_button.pack()

# Run the main event loop
root.mainloop()

```

Output:



The image shows a screenshot of a Tkinter window titled "T...". The window contains three input fields, each with a label "Enter value 1:", "Enter value 2:", and "Enter value 3:" respectively. Below these fields is a "Submit" button. The window has a standard title bar with a feather icon, a minus sign, a square icon, and a close button (X).

Result: Thus the Program using Tkinter module for UI design was executed and verified successfully.

Task 12. Simulate Gaming concepts using Pygame C05-K5

Aim:

To Simulate Gaming concepts using Pygame

SnakeGame:

Problem 1. Write a python program to create a snakeGame using pygame package.

Conditions:

1. Set the window size
2. Create a snake
3. Make the snake to move in the directions when left, right, down and up key is pressed
4. When the snake hits the fruit, increase the score by 10
5. If the snake hits the window, Game over

Sample Output:



Algorithm:

1. Import pygame package and initialize it
2. Define the window size and title
3. Create a Snake class which initializes the snake position, color, and movement
4. Create a Fruit class which initializes the fruit position and color
5. Create a function to check if the snake collides with the fruit and increase the score
6. Create a function to check if the snake collides with the window and end the game
7. Create a function to update the snake position based on the user input
8. Create a function to update the game display and draw the snake and fruit
9. Create a game loop to continuously update the game display, snake position, and check for collisions
10. End the game if the user quits or the snake collides with the window

Program:

```
# importing libraries
import pygame
```

```
import time
import random

snake_speed = 15

# Window size
window_x = 720
window_y = 480

# defining colors
black = pygame.Color(0, 0, 0)
white = pygame.Color(255, 255, 255)
red = pygame.Color(255, 0, 0)
green = pygame.Color(0, 255, 0)
blue = pygame.Color(0, 0, 255)

# Initialising pygame
pygame.init()

# Initialise game window
pygame.display.set_caption('GeeksforGeeks Snakes')
game_window = pygame.display.set_mode((window_x, window_y))

# FPS (frames per second) controller
fps = pygame.time.Clock()

# defining snake default position
snake_position = [100, 50]

# defining first 4 blocks of snake body
snake_body = [[100, 50],
               [90, 50],
               [80, 50],
               [70, 50]
              ]

# fruit position
fruit_position = [random.randrange(1, (window_x//10)) * 10,
                  random.randrange(1, (window_y//10)) * 10]

fruit_spawn = True

# setting default snake direction towards
# right
direction = 'RIGHT'
change_to = direction

# initial score
score = 0
```

```

# displaying Score function
def show_score(choice, color, font, size):

    # creating font object score_font
    score_font = pygame.font.SysFont(font, size)

    # create the display surface object
    # score_surface
    score_surface = score_font.render('Score : ' + str(score), True, color)

    # create a rectangular object for the text
    # surface object
    score_rect = score_surface.get_rect()

    # displaying text
    game_window.blit(score_surface, score_rect)

# game over function
def game_over():

    # creating font object my_font
    my_font = pygame.font.SysFont('times new roman', 50)

    # creating a text surface on which text
    # will be drawn
    game_over_surface = my_font.render(
        'Your Score is : ' + str(score), True, red)

    # create a rectangular object for the text
    # surface object
    game_over_rect = game_over_surface.get_rect()

    # setting position of the text
    game_over_rect.midtop = (window_x/2, window_y/4)

    # blit will draw the text on screen
    game_window.blit(game_over_surface, game_over_rect)
    pygame.display.flip()

    # after 2 seconds we will quit the program
    time.sleep(2)

    # deactivating pygame library
    pygame.quit()

    # quit the program
    quit()

# Main Function

```

while True:

```
# handling key events
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_UP:
            change_to = 'UP'
        if event.key == pygame.K_DOWN:
            change_to = 'DOWN'
        if event.key == pygame.K_LEFT:
            change_to = 'LEFT'
        if event.key == pygame.K_RIGHT:
            change_to = 'RIGHT'

# If two keys pressed simultaneously
# we don't want snake to move into two
# directions simultaneously
if change_to == 'UP' and direction != 'DOWN':
    direction = 'UP'
if change_to == 'DOWN' and direction != 'UP':
    direction = 'DOWN'
if change_to == 'LEFT' and direction != 'RIGHT':
    direction = 'LEFT'
if change_to == 'RIGHT' and direction != 'LEFT':
    direction = 'RIGHT'

# Moving the snake
if direction == 'UP':
    snake_position[1] -= 10
if direction == 'DOWN':
    snake_position[1] += 10
if direction == 'LEFT':
    snake_position[0] -= 10
if direction == 'RIGHT':
    snake_position[0] += 10

# Snake body growing mechanism
# if fruits and snakes collide then scores
# will be incremented by 10
snake_body.insert(0, list(snake_position))
if snake_position[0] == fruit_position[0] and snake_position[1] == fruit_position[1]:
    score += 10
    fruit_spawn = False
else:
    snake_body.pop()

if not fruit_spawn:
    fruit_position = [random.randrange(1, (window_x//10)) * 10,
                     random.randrange(1, (window_y//10)) * 10]
```

```

fruit_spawn = True
game_window.fill(black)

for pos in snake_body:
    pygame.draw.rect(game_window, green,
                      pygame.Rect(pos[0], pos[1], 10, 10))
pygame.draw.rect(game_window, white, pygame.Rect(
    fruit_position[0], fruit_position[1], 10, 10))
# Game Over conditions
if snake_position[0] < 0 or snake_position[0] > window_x-10:
    game_over()
if snake_position[1] < 0 or snake_position[1] > window_y-10:
    game_over()
# Touching the snake body
for block in snake_body[1:]:
    if snake_position[0] == block[0] and snake_position[1] == block[1]:
        game_over()
# displaying score continuously
show_score(1, white, 'times new roman', 20)

# Refresh game screen
pygame.display.update()

# Frame Per Second /Refresh Rate
fps.tick(snake_speed)

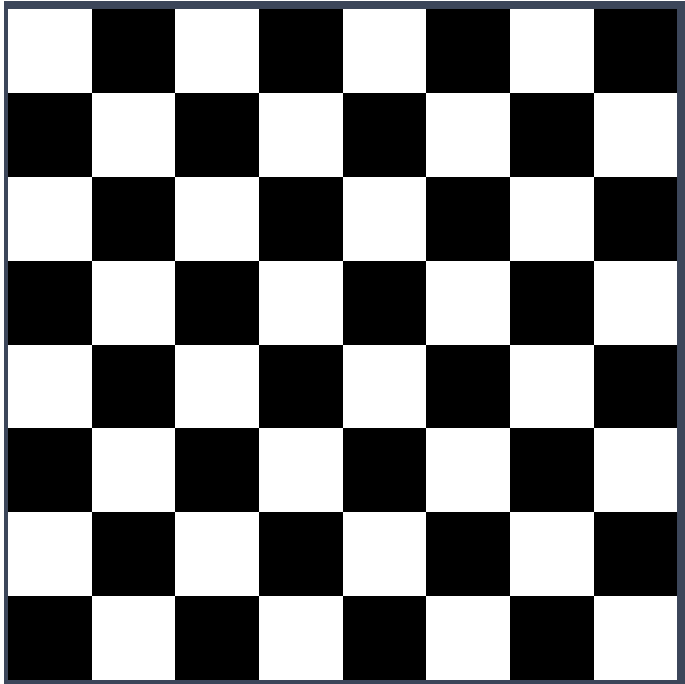
```

Output



Problem 2. Write a python program to Develop a chess board using pygame.

Sample output:

**Algorithm:**

1. Import pygame and initialize it.
2. Set screen size and title.
3. Define colors for the board and pieces.

Define a function to draw the board by looping over rows and columns and drawing squares of different colors.

4. Define a function to draw the pieces on the board by loading images for each piece and placing them on the corresponding square.
5. Define the initial state of the board as a list of lists containing the pieces.
6. Draw the board and pieces on the screen.
7. Start the game loop.

Program:

```
import pygame
```

```
# Initialize pygame
pygame.init()
```

```
# Set screen size and title
screen_size = (640, 640)
screen = pygame.display.set_mode(screen_size)
pygame.display.set_caption('Chess Board')
```

```
# Define colors
black = (0, 0, 0)
white = (255, 255, 255)
```

```

brown = (153, 76, 0)

# Define function to draw the board
def draw_board():
    for row in range(8):
        for col in range(8):
            square_color = white if (row + col) % 2 == 0 else brown
            square_rect = pygame.Rect(col * 80, row * 80, 80, 80)
            pygame.draw.rect(screen, square_color, square_rect)

# Define function to draw the pieces
def draw_pieces(board):
    piece_images = {
        'r': pygame.image.load('images/rook.png'),
        'n': pygame.image.load('images/knight.png'),
        'b': pygame.image.load('images/bishop.png'),
        'q': pygame.image.load('images/queen.png'),
        'k': pygame.image.load('images/king.png'),
        'p': pygame.image.load('images/pawn.png')
    }
    for row in range(8):
        for col in range(8):
            piece = board[row][col]
            if piece != '.':
                piece_image = piece_images[piece]
                piece_rect = pygame.Rect(col * 80, row * 80, 80, 80)
                screen.blit(piece_image, piece_rect)

# Define initial state of the board
board = [
    ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'],
    ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],
    [',', ',', ',', ',', ',', ',', ',', ','],
    [',', ',', ',', ',', ',', ',', ',', ','],
    [',', ',', ',', ',', ',', ',', ',', ','],
    [',', ',', ',', ',', ',', ',', ',', ','],
    [',', ',', ',', ',', ',', ',', ',', ','],
    ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],
    ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R']
]

# Draw board and pieces
draw_board()
draw_pieces(board)

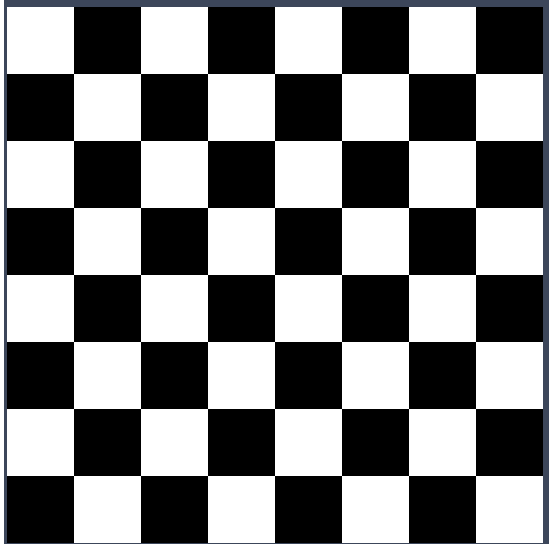
# Start game loop
while True:
    for event in pygame.event.get():

```



```
if event.type == pygame.QUIT:  
    pygame.quit()  
    quit()  
pygame.display.update()
```

Output:



Result:

Thus the program for pygame is executed and verified successfully.