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:

***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:

Result:

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

Task 8.Implement python generator and decorators CO1-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:

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:

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:

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