SriChandrasekharendraSaraswathiViswaMahavidyalaya
[SCSVMV]
[Deemed to be University U/S 3 of the UGC Act 1956]
Accredited with 'A' Grade by NAAC
Enathur,Kanchipuram–631561

Department of Computer Science and Engineering

LAB MANUAL

Course Code: BCSF183P80

Course Title: PYTHON PROGRAMMING LAB
Programs (UG/PG): UG II BECSE

Semester: III(ODD SEMESTER)
Academic Year: 2024-2025

PREPARED BY

Dr.R.PREMA
ASSISTANT PROFESSOR
DEPT.OF CSE
SCSVMV DEEMED TO BE UNIVERSITY
KANCHIPURAM TAMILNADU 631561

SriChandrasekharendraSaraswathiViswaMahavidyalaya

[SCSVMV**]**

[Deemed to be University U/S 3 of the UGC Act 1956] Accredited with 'A' Grade by NAAC

Enathur,Kanchipuram–631561

Department of Computer Science and Engineering

Course Code:	PYTHON PROGRAMMING LAB	L	T	P	C
BCSF183P80		0	0	3	2

PRE-REQUISITE

Knowledge in basics of any Programming language

OBJECTIVES

- To write ,test, and debug simple Python programs.
- Read and write data from/to files in Python.
- Represent compound data using Python lists, tuples, dictionaries.
- Use functions for structuring Python programs.
- To implement Python program swith conditionals and loops.

OUTCOMES

Upon completion of the course, students will be able to:

- Write, test, and debug simple Python programs.
- Implement Python programs with conditionals and loops.
- Develop Python programs step-wise by defining functions and calling them.
- Use Pythonlists,tuples,dictionaries for representing compound data.
- Read and write data from/to files in Python.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Mapping with Programme outcomes

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1			S									
CO2			S									
CO3				M								
CO4				M								
CO5				M								

S-STRONG,M-MEDIUM,L-LOW

LISTOFPROGRAMS

1. Compute the GCD of two numbers.
2. Find the square root of a number(Newton'smethod)
3. Exponentiation(power of a number)
4. Find the maximum of a list of numbers
5. Linear search and Binary search
6. Selection sort,Insertion sort
7. Merge sort
8. First n prime numbers
9. Multiply matrices
10. Programs that take command line arguments(wordcount)

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

11. Find the most frequent words in a text read from a file
12. Simulate elliptical or bits in Pygame
13. Simulate bouncing ball using Pygame

PLATFORM NEEDED

Python 3 interpreter for Windows/Linux

SYLLABYS PREPAREDBY

Mr.J.S.ShyamMohan, Assistant Professor/CSE

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

INTERNAL ASSESSMENT MARK SPLIT UP:

Observation: 15 Marks

Model Exam: 15 Marks

Record and Attendance: 10 Marks

Total: 40 Marks

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Installation of PyCharm

Aim: To install PyCharm Software and Packages.

Procedure:

How to install PyCharm:

Step 1) Go to website official website of JetBrains https://www.jetbrains.com/pycharm/download/ and click on the "DOWNLOAD" link of the Community section.

Step 2) After clicking on Download Click on Next

Step 3) After Click on Next, you need to choose the destination folder according to your choice.

Step 4) Choose options of installation according to your choice.

Step 5) Choose Jet Brains and Click on "Install".

Step 6) Let the installation be finished.

Step 7) After Installation completed, It will show that PyCharm is installed successfully, then click on "I want to manually reboot later". Click on Finish and then the process is completed.

Output:

The PyCharm installed.

Result:

Hence, the installation of PyCharm successful.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

EXERCISE 1:Compute the GCD of two numbers.

Objective:

To compute the **Greatest Common Divisor (GCD)** of two numbers using Python.

Theory:

The GCD (or HCF - Highest Common Factor) of two numbers is the largest number that divides both of them without leaving a remainder.

For example:

- GCD of 60 and 48 is 12 because 12 is the largest number that divides both 60 and 48.

Methods to Compute GCD:

1. **Using the Euclidean Algorithm:** The Euclidean algorithm is an efficient way to compute the GCD of two numbers. The algorithm is based on the principle:

1. GCD(a,b)=GCD(b,a%b)

   where% is the modulo operator. The process continues until b becomes 0, at which point a is the GCD.

2. **Using Python's built-in math.gcd() function:** Python has a built-in function for GCD in the math library.

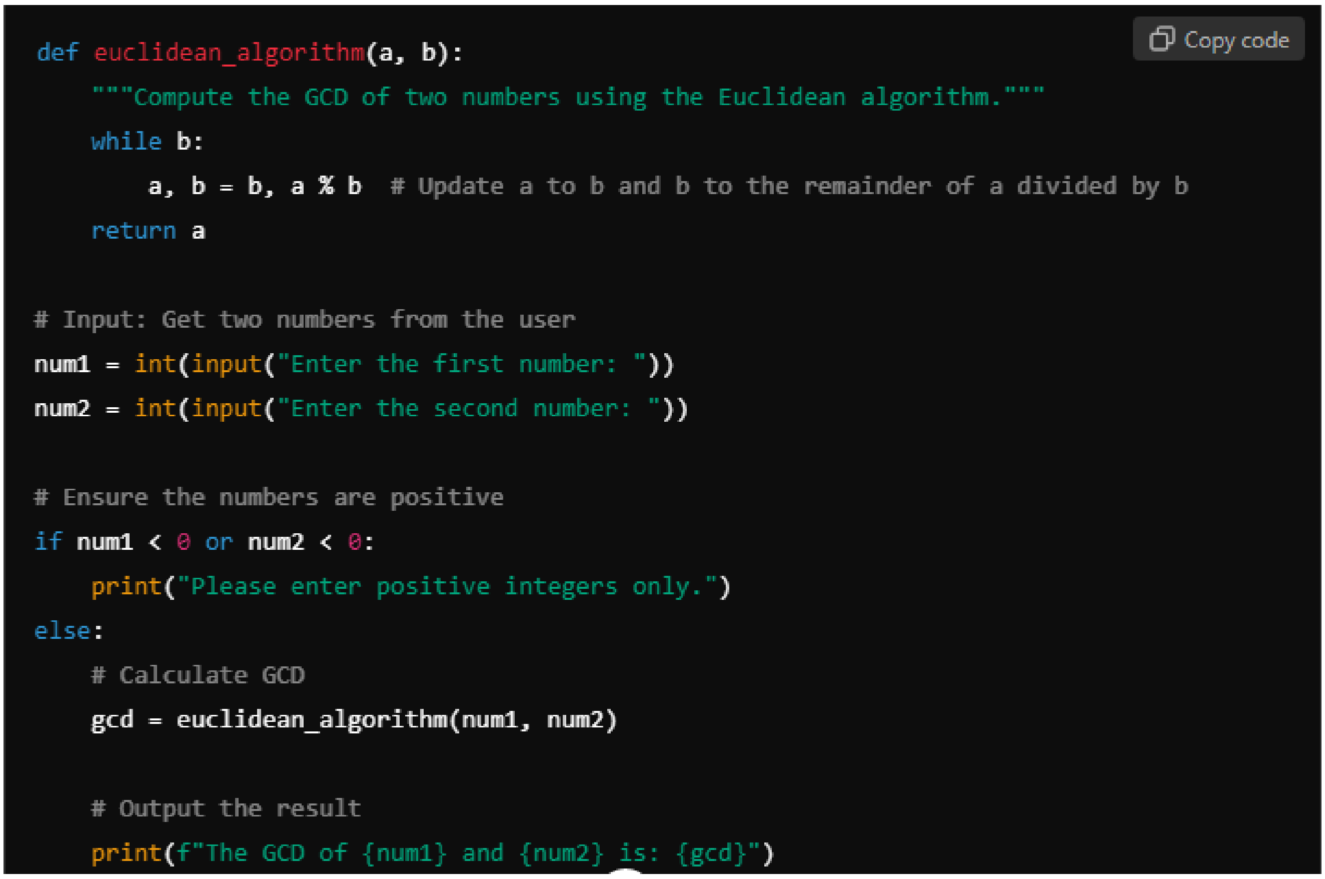
Procedure:

1. **Step 1:** Import necessary libraries (if using Python's built-in function).

2. **Step 2:** Define a function to calculate the GCD using the Euclidean algorithm.
3. **Step 3:** Prompt the user for two integers as input.
4. **Step 4:** Call the GCD function and display the result.
5. **Step 5 (optional):** You can also use Python's built-in math.gcd() function to check the result.

Code:

Method 1: Using the Euclidean Algorithm



```python
def euclidean_algorithm(a, b):
    """Compute the GCD of two numbers using the Euclidean algorithm."""
    while b:
        a, b = b, a % b  # Update a to b and b to the remainder of a divided by b
    return a

# Input: Get two numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Ensure the numbers are positive
if num1 < 0 or num2 < 0:
    print("Please enter positive integers only.")
else:
    # Calculate GCD
    gcd = euclidean_algorithm(num1, num2)

    # Output the result
    print(f"The GCD of {num1} and {num2} is: {gcd}")
```

Explanation:

1. **Function Definition**: The euclidean_ algorithm function takes two parameters, a and b.

It uses a while loop to continue until b becomes zero.
Inside the loop, the values of a and b are updated: a takes the value of b, and b takes the value of a % b, which is the remainder of the division of a by b.

2. **User Input**:

The program prompts the user to enter two integers.
It checks to ensure the numbers are positive.

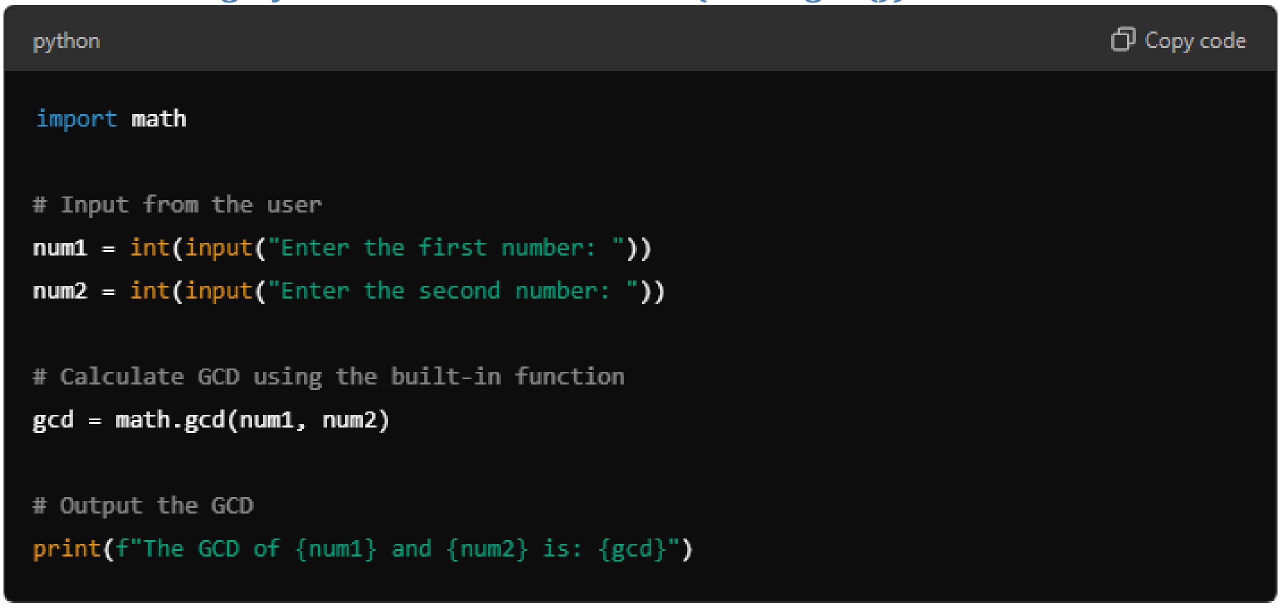
3. **Calculate GCD**:

The GCD is calculated by calling the euclidean_algorithm function with the two numbers.

4. **Output**:

The program prints the result.

Method 2: Using Python's Built-in Function (math.gcd())



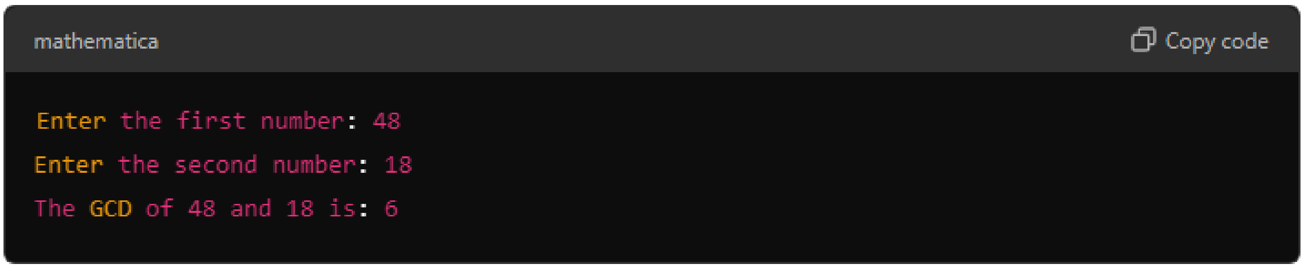
```python
import math

# Input from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Calculate GCD using the built-in function
gcd = math.gcd(num1, num2)

# Output the GCD
print(f"The GCD of {num1} and {num2} is: {gcd}")
```

Explanation:

Built-in math.gcd()

If you use Python's built-in function, you simply import the math module and use math.gcd(num1, num2) to get the GCD.

Example Output:



```
Enter the first number: 48
Enter the second number: 18
The GCD of 48 and 18 is: 6
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

- The Euclidean algorithm is an efficient method to compute the GCD.
- Python's math.gcd() is a built-in solution that simplifies the task even further.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

2.Find the square root of a number(Newton'smethod)

Objective:

To find the square root of a number using **Newton's Method** (also called **Newton-Raphson Method**) in Python.

Theory:

Newton's method is an iterative approach to find the square root of a number. It refines an initial guess by repeatedly applying the formula:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$$

Where:

- xnx_nxn is the current guess.
- S is the number whose square root is to be found.
- xn+1x_{n+1}xn+1 is the next guess (which will be closer to the actual square root).

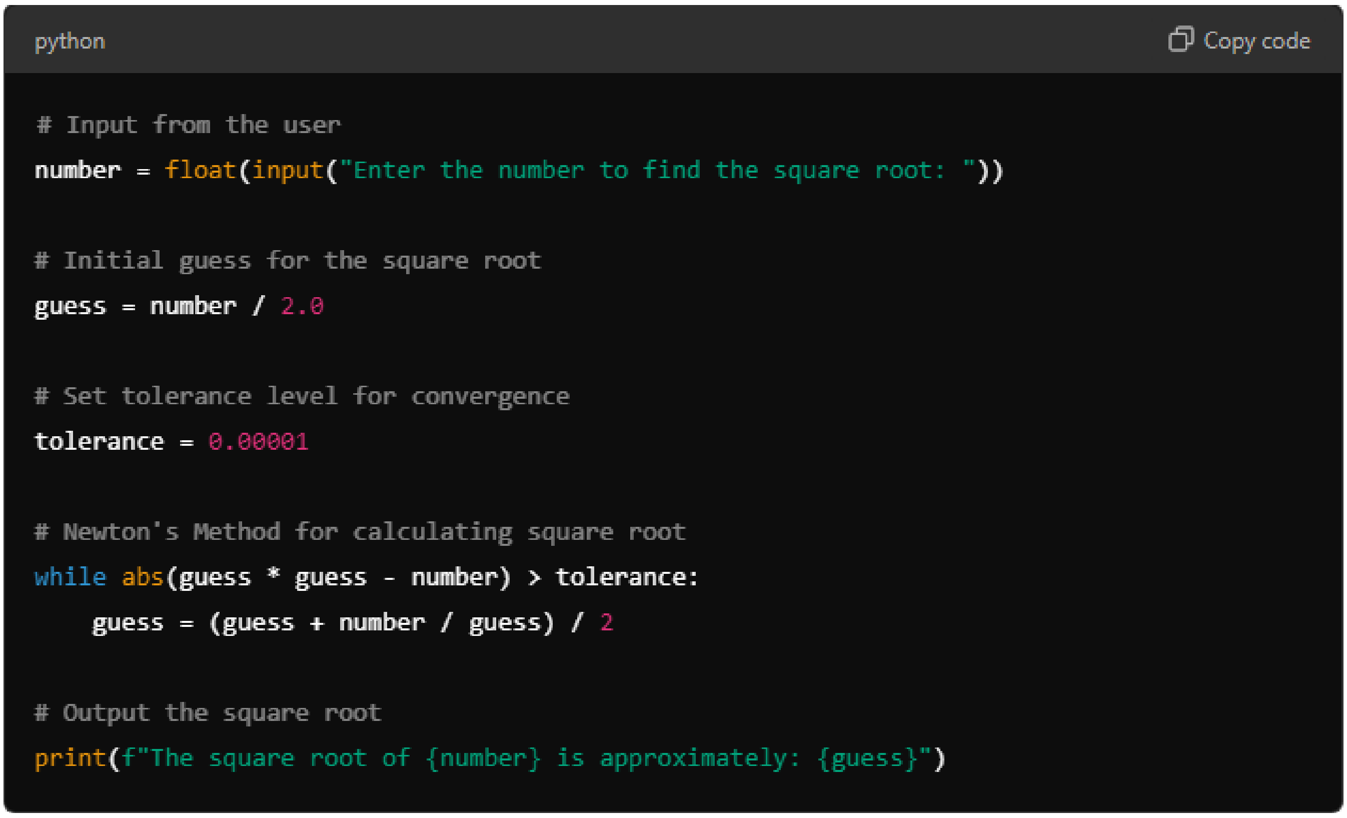
The iteration continues until the difference between the current and previous guesses becomes small enough (converges to a certain level of precision).

Procedure:

1. **Step 1:** Initialize a guess value (for instance, the number divided by 2).

2. **Step 2:** Implement the iterative formula using a loop. Stop the loop when the difference between successive guesses is smaller than a pre-set tolerance (e.g., 0.00001).
3. **Step 3:** Test the algorithm with a user input value.
4. **Step 4:** Compare the result with Python's built-in square root function math.sqrt() to verify accuracy.

Code:



```python
# Input from the user
number = float(input("Enter the number to find the square root: "))

# Initial guess for the square root
guess = number / 2.0

# Set tolerance level for convergence
tolerance = 0.00001

# Newton's Method for calculating square root
while abs(guess * guess - number) > tolerance:
    guess = (guess + number / guess) / 2

# Output the square root
print(f"The square root of {number} is approximately: {guess}")
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

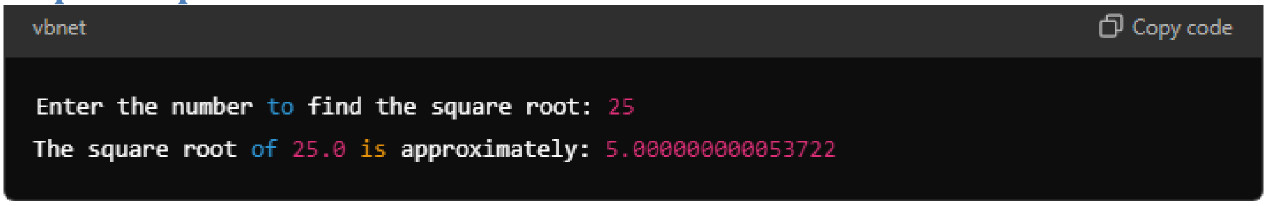
Explanation:

1. **Initial Guess**: The algorithm starts with an initial guess for the square root (typically number / 2).
2. **Iterative Improvement**: In each iteration, we improve the guess using the formula

$$\text{new guess} = \frac{1}{2} \times \quad \text{guess} + \frac{\text{number}}{\text{guess}}$$

Convergence: The loop continues until the difference between guess * guess and the original number is smaller than the defined tolerance (e.g., 0.00001).

Example Output:

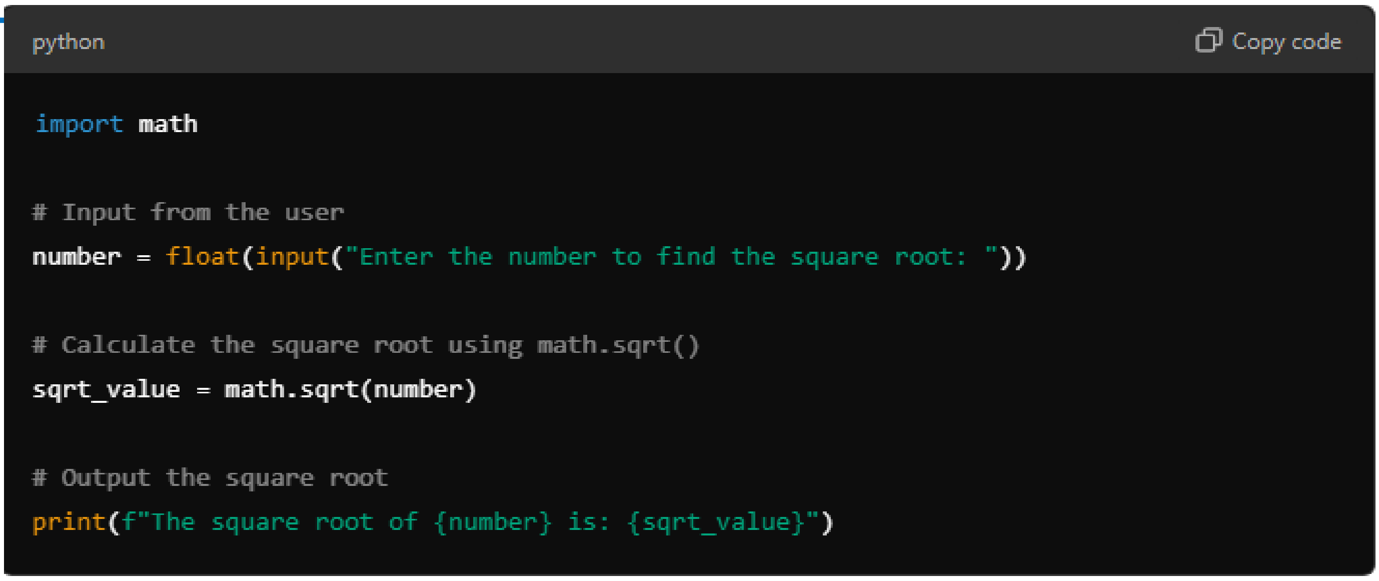


```
Enter the number to find the square root: 25
The square root of 25.0 is approximately: 5.000000000053722
```

Verification Using Built-in Python Function:

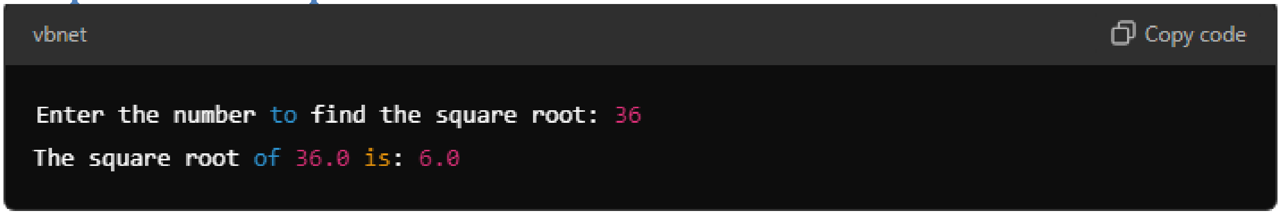
You can verify the result using Python's built-in math.sqrt() function. Here's an additional step to include in your code:

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu



```python
import math

# Input from the user
number = float(input("Enter the number to find the square root: "))

# Calculate the square root using math.sqrt()
sqrt_value = math.sqrt(number)

# Output the square root
print(f"The square root of {number} is: {sqrt_value}")
```

Example of Full Output:



```
Enter the number to find the square root: 36
The square root of 36.0 is: 6.0
```

Explanation:

- The math.sqrt() function calculates the square root of a given number.
- The function handles the computation directly and efficiently without needing iterative or custom methods.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

- **Newton's Method** provides a fast, iterative approach to finding the square root of a number.The method converges quickly to an accurate result with a mall tolerance.You can compare it with Python's built-in math.sqrt() function to check the accuracy of the result.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

3. Exponentiation (Power of a Number)

Objective:

To calculate the exponentiation of a number (i.e., the result of raising one number to the power of another) using Python.

Theory:

Exponentiation is the operation of raising one number, called the **base**, to the power of another number, called the **exponent**. For example:

$$a^b = a \times a \times \cdots \times a \quad \text{(b times)}$$

Where:

- a is the **base**.
- b is the **exponent**.

Python provides multiple ways to calculate exponentiation:

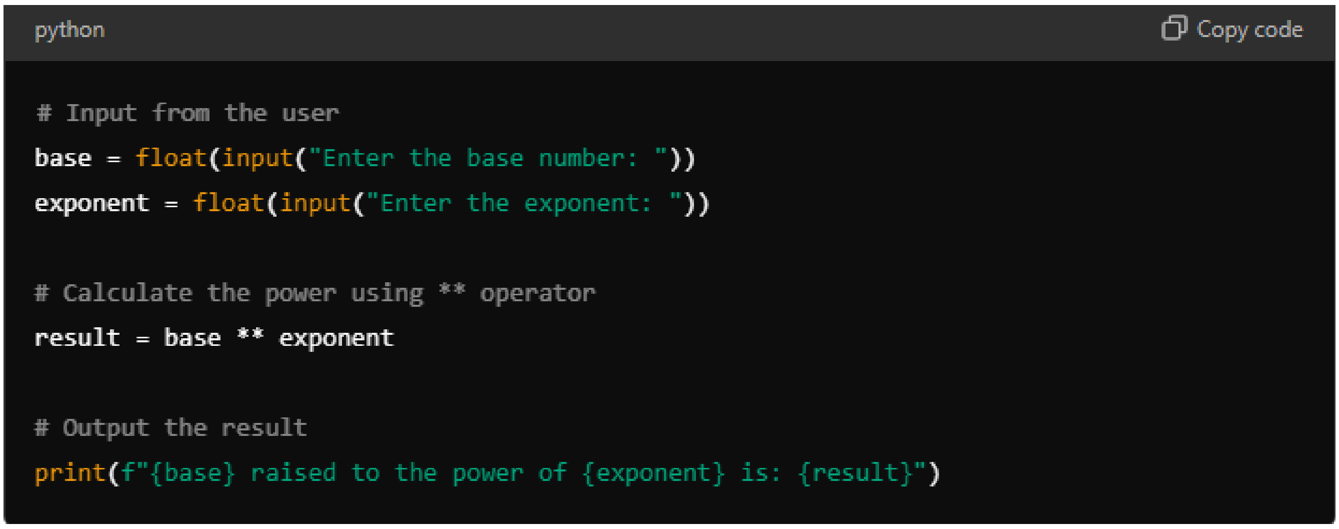
1. Using the ** operator.
2. Using the math.pow() function.
3. Using a custom iterative or recursive method.

Procedure:

1. **Step 1:** Accept the base and exponent as input from the user.
2. **Step 2:** Implement exponentiation using different approaches:
   - Using the ** operator.
   - Using the built-in math.pow() function.
   - Implementing an iterative approach (optional).
3. **Step 3:** Print the result for each method.
4. **Step 4:** Test the code with different inputs to ensure correctness.

Code:

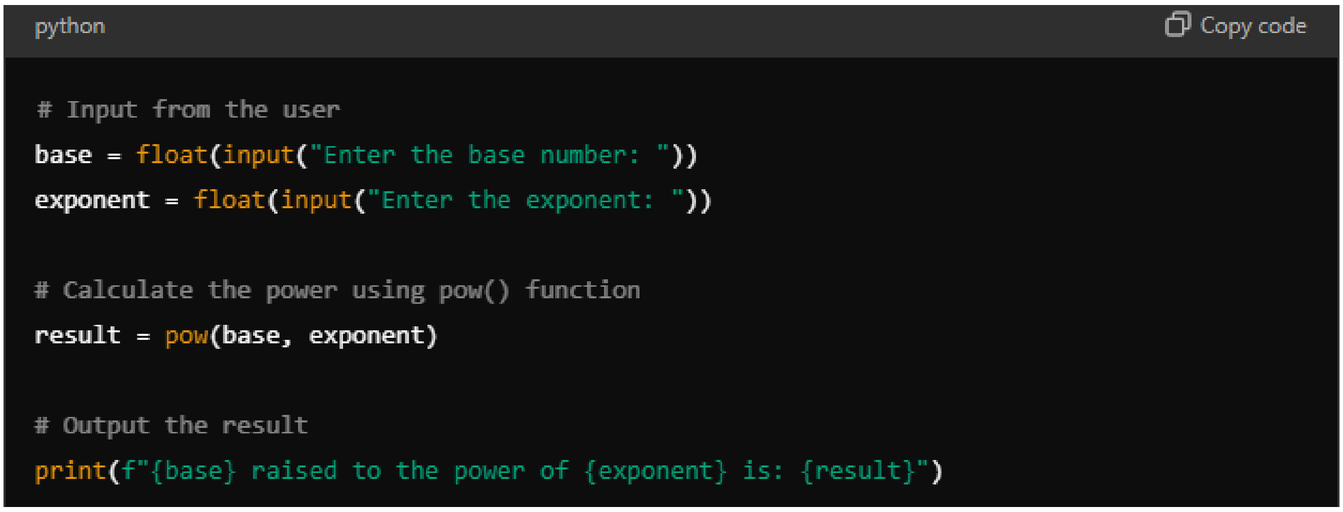
*Method 1: Using the ** Operator*



python Copy code

```python
# Input from the user
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))

# Calculate the power using ** operator
result = base ** exponent

# Output the result
print(f"{base} raised to the power of {exponent} is: {result}")
```

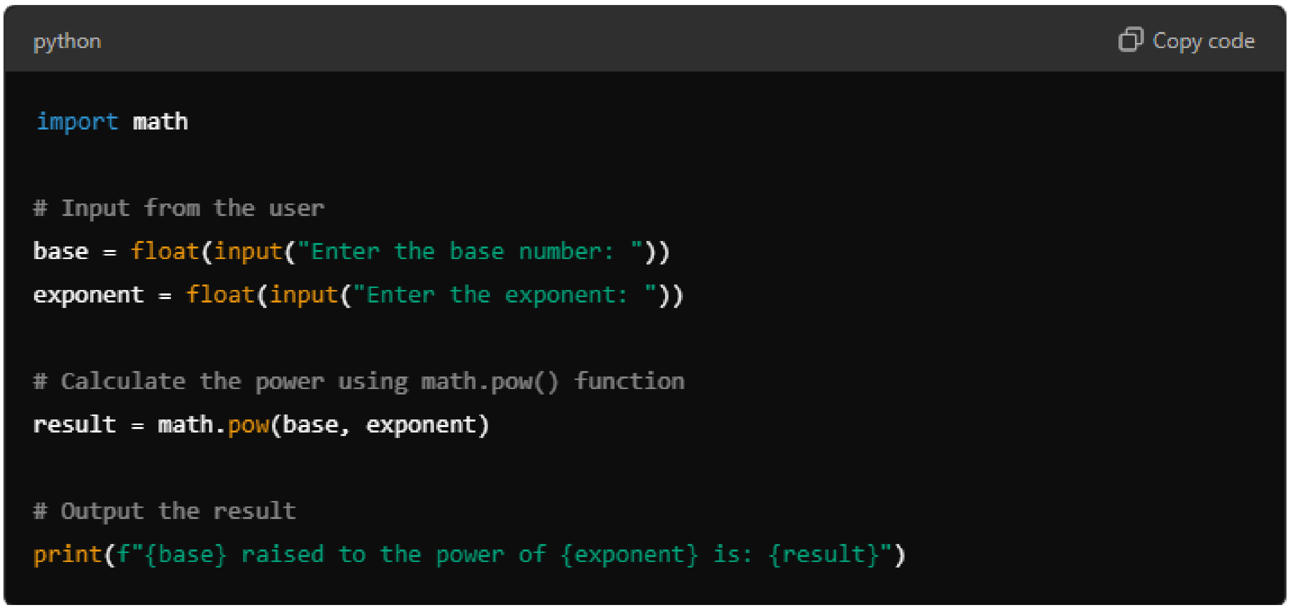
Method 2: Using math.pow()



python Copy code

```python
# Input from the user
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))

# Calculate the power using pow() function
result = pow(base, exponent)

# Output the result
print(f"{base} raised to the power of {exponent} is: {result}")
```

Method 3: Custom Iterative Function (without using built-in operators)



python Copy code

```python
import math

# Input from the user
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))

# Calculate the power using math.pow() function
result = math.pow(base, exponent)

# Output the result
print(f"{base} raised to the power of {exponent} is: {result}")
```

Procedure Explanation:

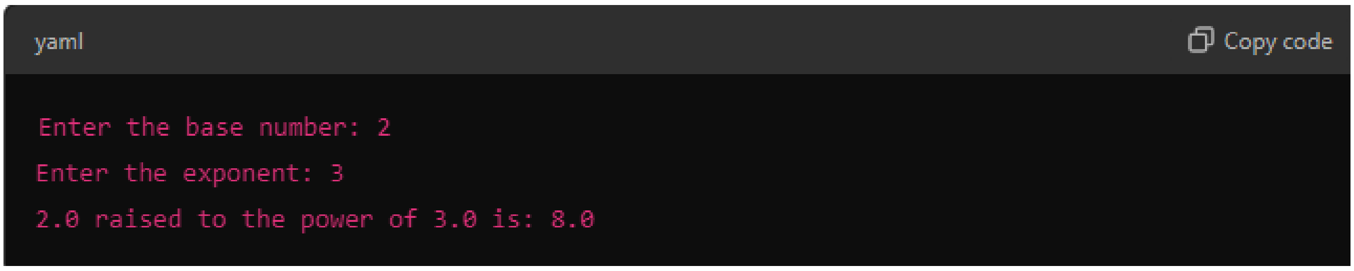
1. **Step 1: Input from the user**
   The program prompts the user to input the **base** and **exponent**. These are stored in the variables base and exponent.
2. **Step 2: Calculation using different methods**
   ○

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

- **Method 1**: Uses the ** operator, which is the simplest and most Pythonic way to calculate the power of a number.
- **Method 2**: Uses Python's built-in math.pow() function. This function performs exponentiation and is part of the math module.
- **Method 3**: Implements an **iterative** approach to calculate the power manually by multiplying the base by itself in a loop exponent number of times.

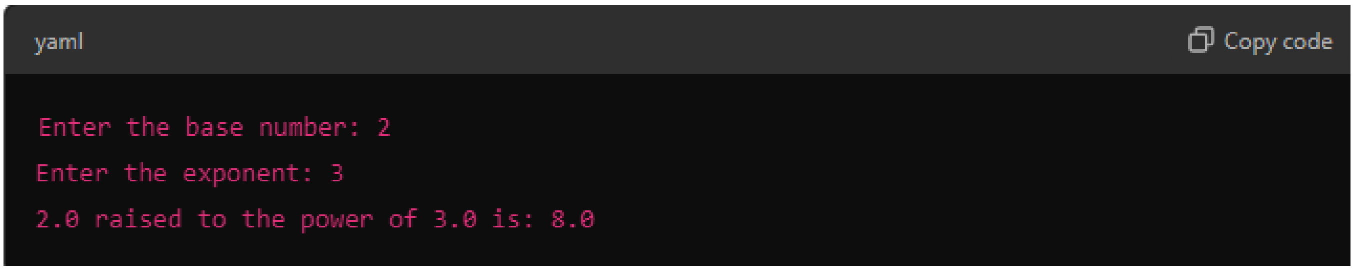
3. **Step 3: Output the result**
   The program prints the result of each method.

Example Output:Method 1 (** operator):


yaml Copy code

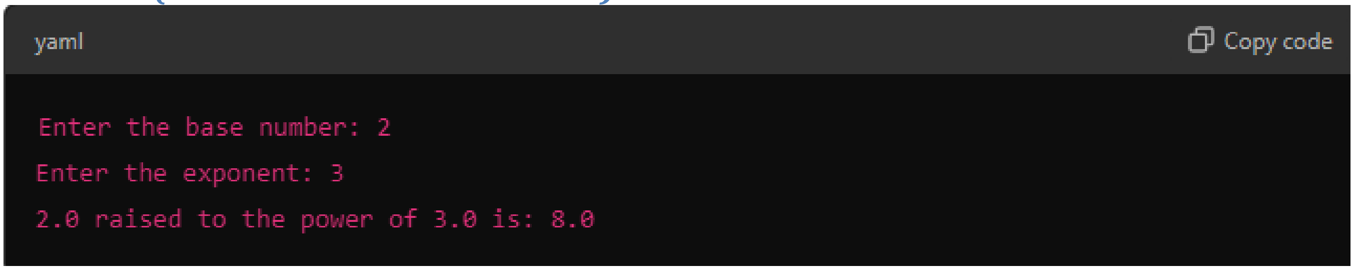
```
Enter the base number: 2
Enter the exponent: 3
2.0 raised to the power of 3.0 is: 8.0
```

Method 2 (math.pow()):


yaml Copy code

```
Enter the base number: 2
Enter the exponent: 3
2.0 raised to the power of 3.0 is: 8.0
```

Method 3 (Custom Iterative Function):


yaml Copy code

```
Enter the base number: 2
Enter the exponent: 3
2.0 raised to the power of 3.0 is: 8.0
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

- The ** **operator** is the simplest and most efficient way to compute the power of a number in Python.
- The **math.pow()** function is another built-in way to calculate exponentiation.
- A **custom iterative approach** allows for a deeper understanding of the mechanics of exponentiation, particularly useful when implementing similar functionality in languages without built-in exponentiation functions.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

4. Find the Maximum of a List of Numbers

Objective:

To find the maximum value in a list of numbers using Python.

Theory:

The **maximum** value in a list of numbers is the largest element in that list. Python provides various ways to find the maximum value in a list:

1. Using the built-in max() function.
2. Implementing a custom function to iterate through the list and find the maximum manually.

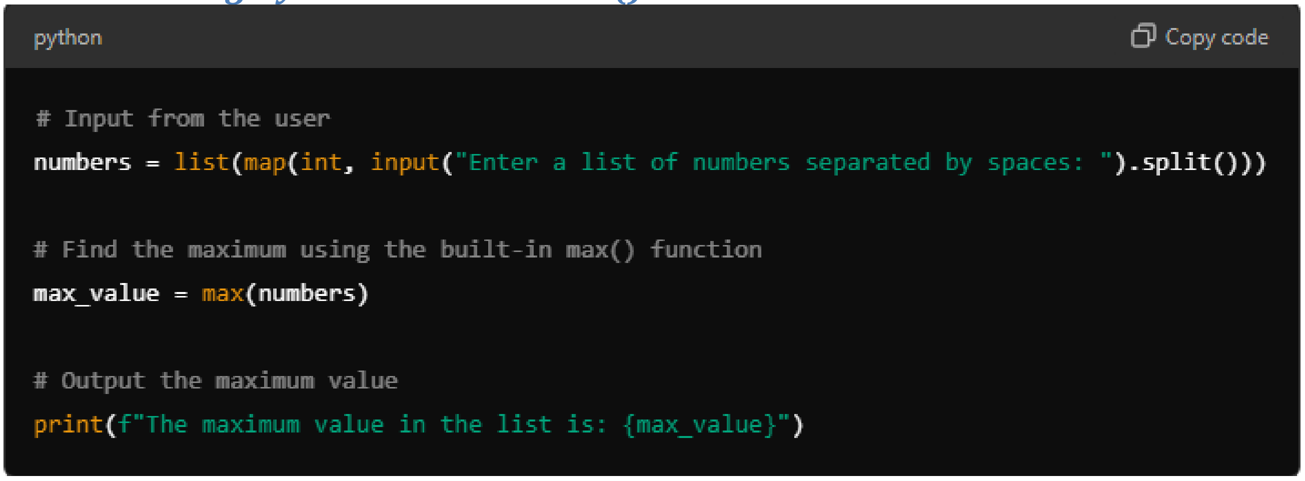
Procedure:

1. **Step 1:** Create or accept a list of numbers from the user.
2. **Step 2:** Implement the logic to find the maximum of the list using:
   - Python's built-in max() function.
   - A custom iterative function.
3. **Step 3:** Display the result.
4. **Step 4:** Test the code with different sets of numbers.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

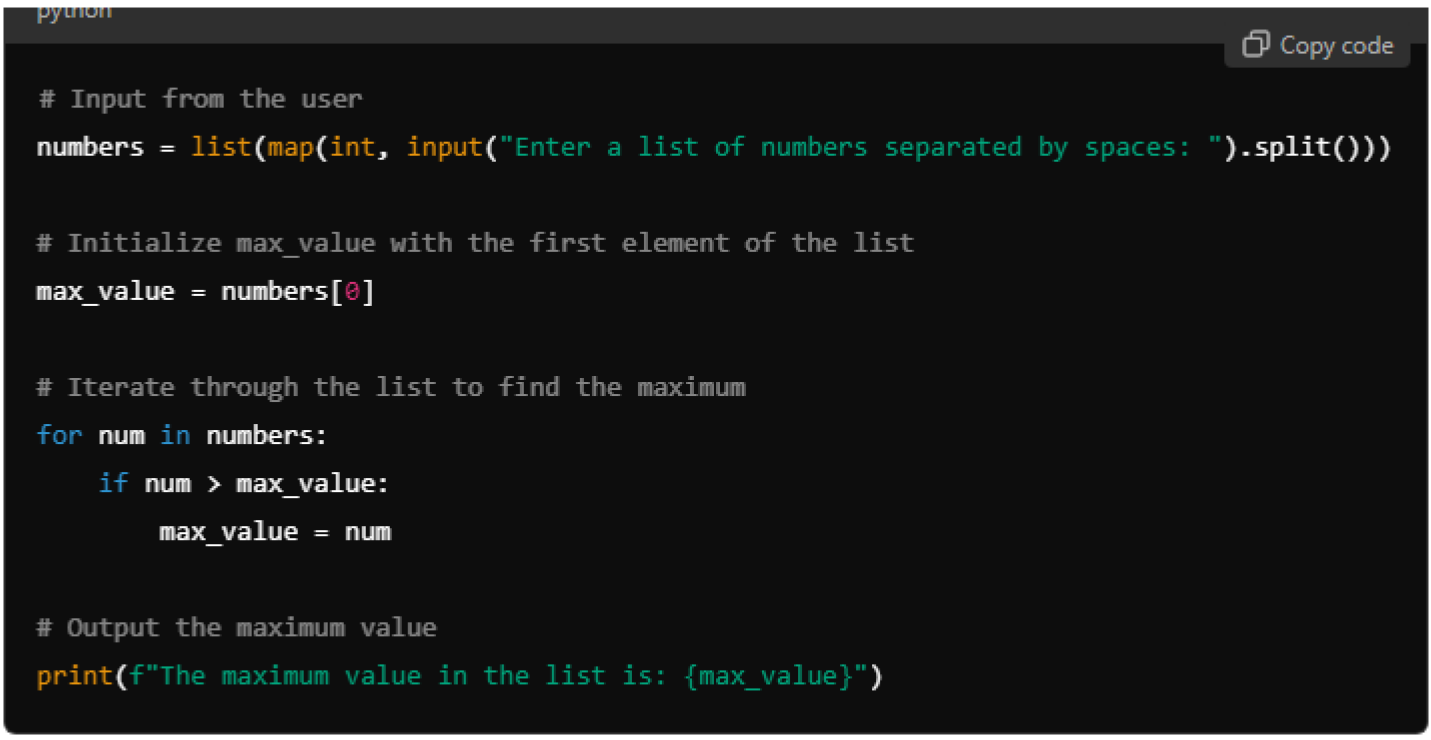
Code:

Method 1: Using Python's Built-in max() Function


python
Copy code

```python
# Input from the user
numbers = list(map(int, input("Enter a list of numbers separated by spaces: ").split()))

# Find the maximum using the built-in max() function
max_value = max(numbers)

# Output the maximum value
print(f"The maximum value in the list is: {max_value}")
```

Method 2: Custom Iterative Function



```python
# Input from the user
numbers = list(map(int, input("Enter a list of numbers separated by spaces: ").split()))

# Initialize max_value with the first element of the list
max_value = numbers[0]

# Iterate through the list to find the maximum
for num in numbers:
    if num > max_value:
        max_value = num

# Output the maximum value
print(f"The maximum value in the list is: {max_value}")
```

Procedure Explanation:

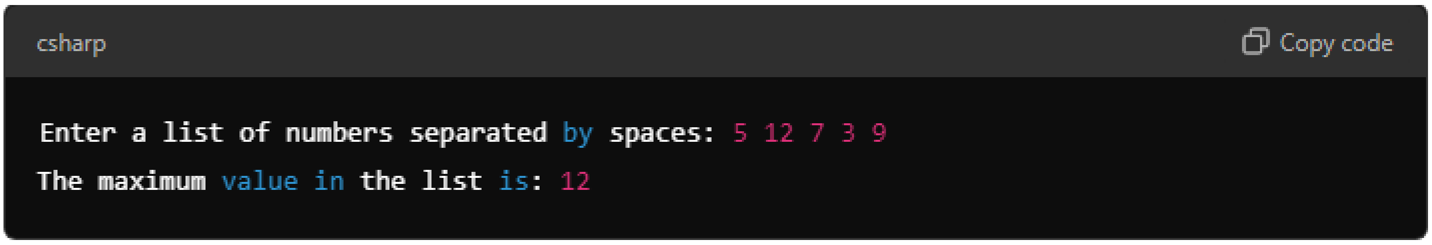
1. **Step 1: Input the List of Numbers**
   The program prompts the user to input a list of numbers separated by spaces. The input() function reads the input as a string, and map(float, input().split()) splits the string into individual numbers and converts them to float. This list is then stored in the variable numbers.
2. **Step 2: Finding the Maximum Using Built-in Function**
   - The max() function is used to directly find and return the largest number in the list.
3. **Step 3: Finding the Maximum Using a Custom Function**
   - In this approach, a function find_max() is defined, where the list is iterated over, and the maximum value is updated as each element is compared.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

4. **Step 4: Output the Result** The result is printed using the print() function, displaying the maximum value found in the list.

Example Output:

Method 1 (max() function):



```
Enter a list of numbers separated by spaces: 5 12 7 3 9
The maximum value in the list is: 12
```

Method 2 (Custom Iterative Function):

Enter numbers separated by spaces: 5.2 8.1 9.5 4.3
The maximum value in the list is: 9.5

Conclusion:

- The built-in **max() function** is the easiest and most efficient way to find the maximum value in a list of numbers.
- A **custom iterative function** allows for manual control of the logic and is useful for understanding how the process works under the hood. This method can be useful in environments or languages without built-in functions for finding the maximum value.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

5.Linear Search and Binary Search

Objective:

To implement both **Linear Search** and **Binary Search** algorithms to search for an element in a list of numbers using Python.

Theory:

1. Linear Search:

Linear search is a simple search algorithm that checks every element in a list sequentially until the target element is found or the end of the list is reached.

- **Time Complexity:**O(n) (worst-case scenario where n is the length of the list).

2. Binary Search:

Binary search is a more efficient algorithm that works on **sorted lists**. It repeatedly divides the search interval in half. If the target value is smaller than the middle element, it searches the left half; otherwise, it searches the right half.

- **Time Complexity:**O(log n) (where n is the number of elements in the list).
- **Prerequisite:** The list must be **sorted**.

Procedure:

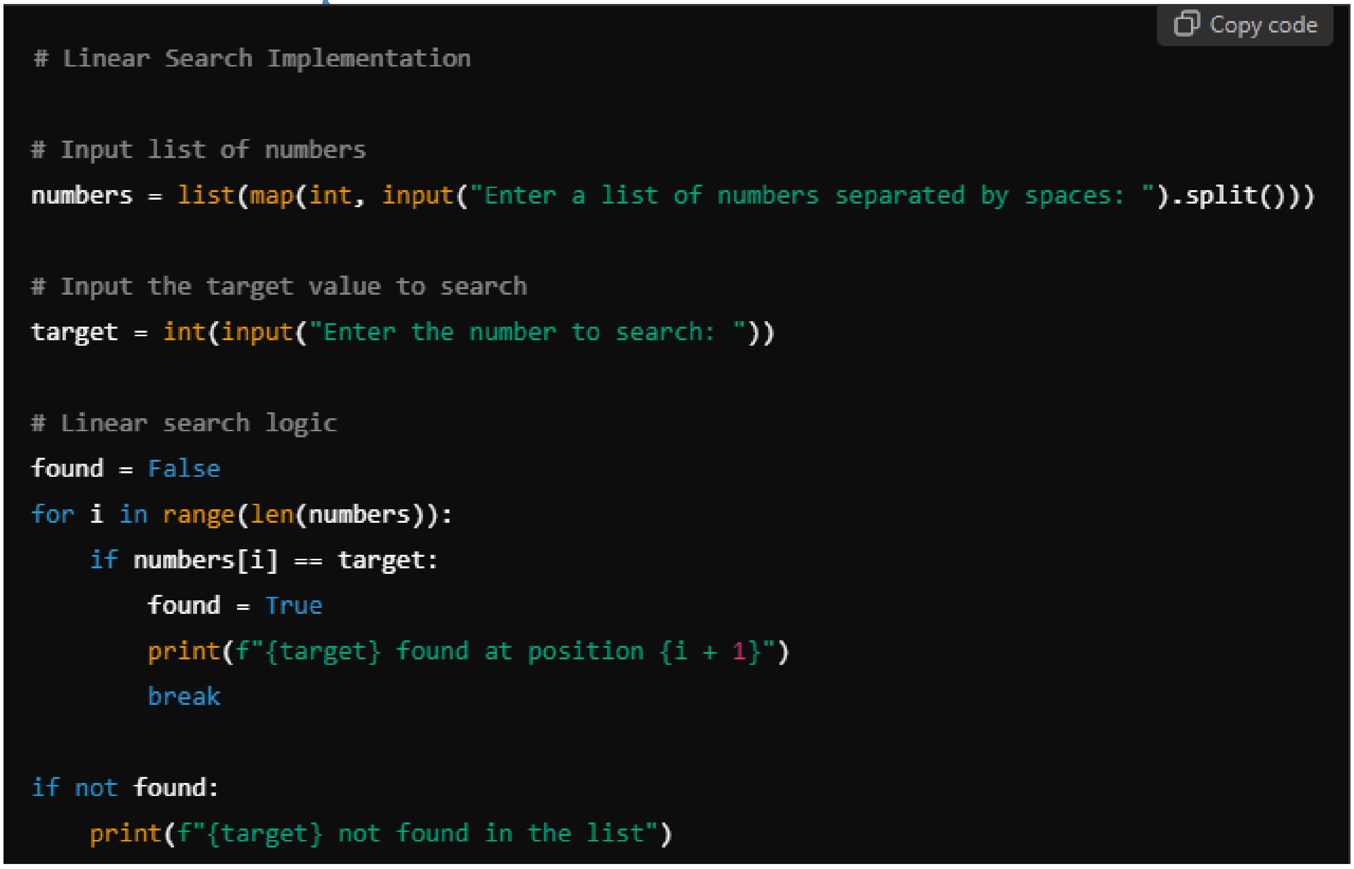
1. **Step 1:** Accept the list of numbers and the target element to search.
2. **Step 2:** Implement Linear Search.
3. **Step 3:** Implement Binary Search (the list must be sorted before applying binary search).
4. **Step 4:** Print whether the target element was found and its index.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

5. **Step 5:** Test both search methods with different inputs.

Code:

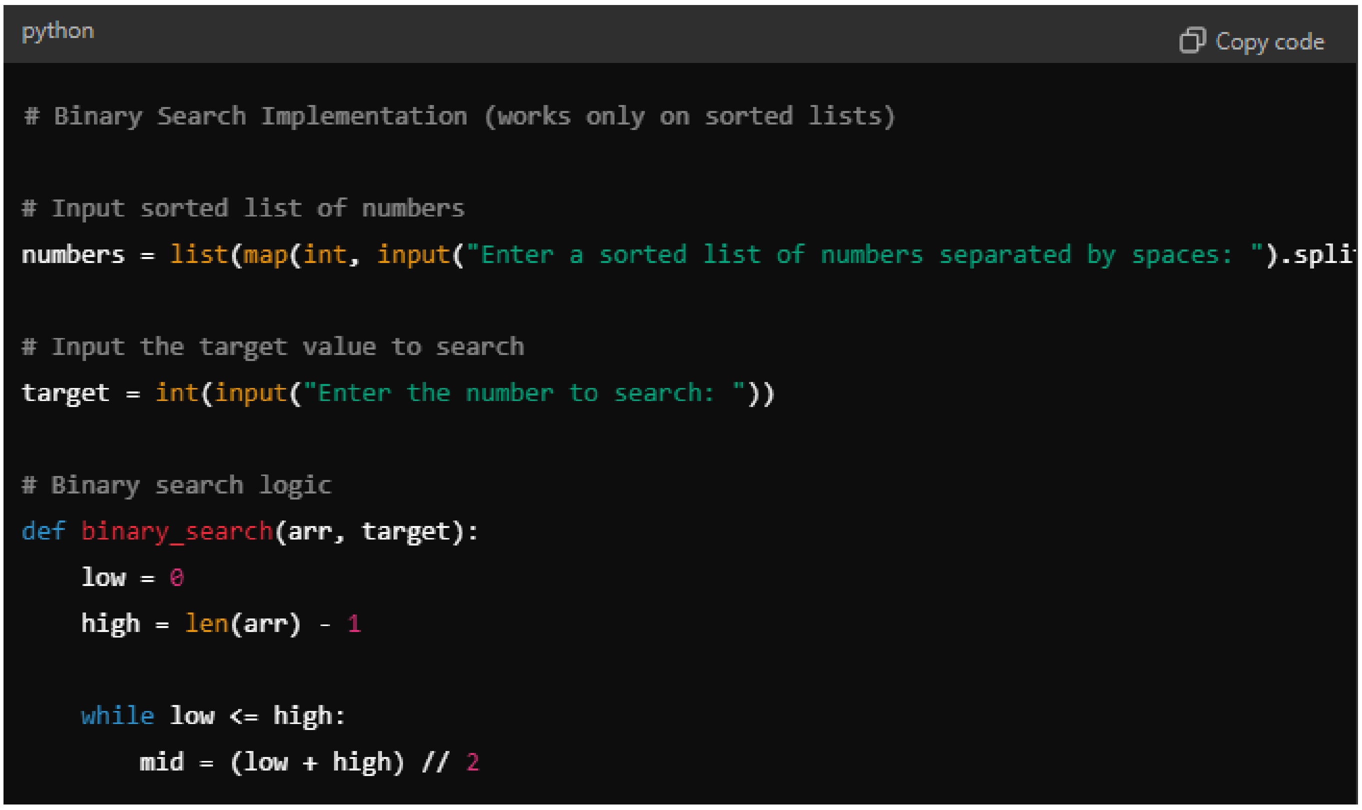
1. Linear Search Implementation:



Copy code

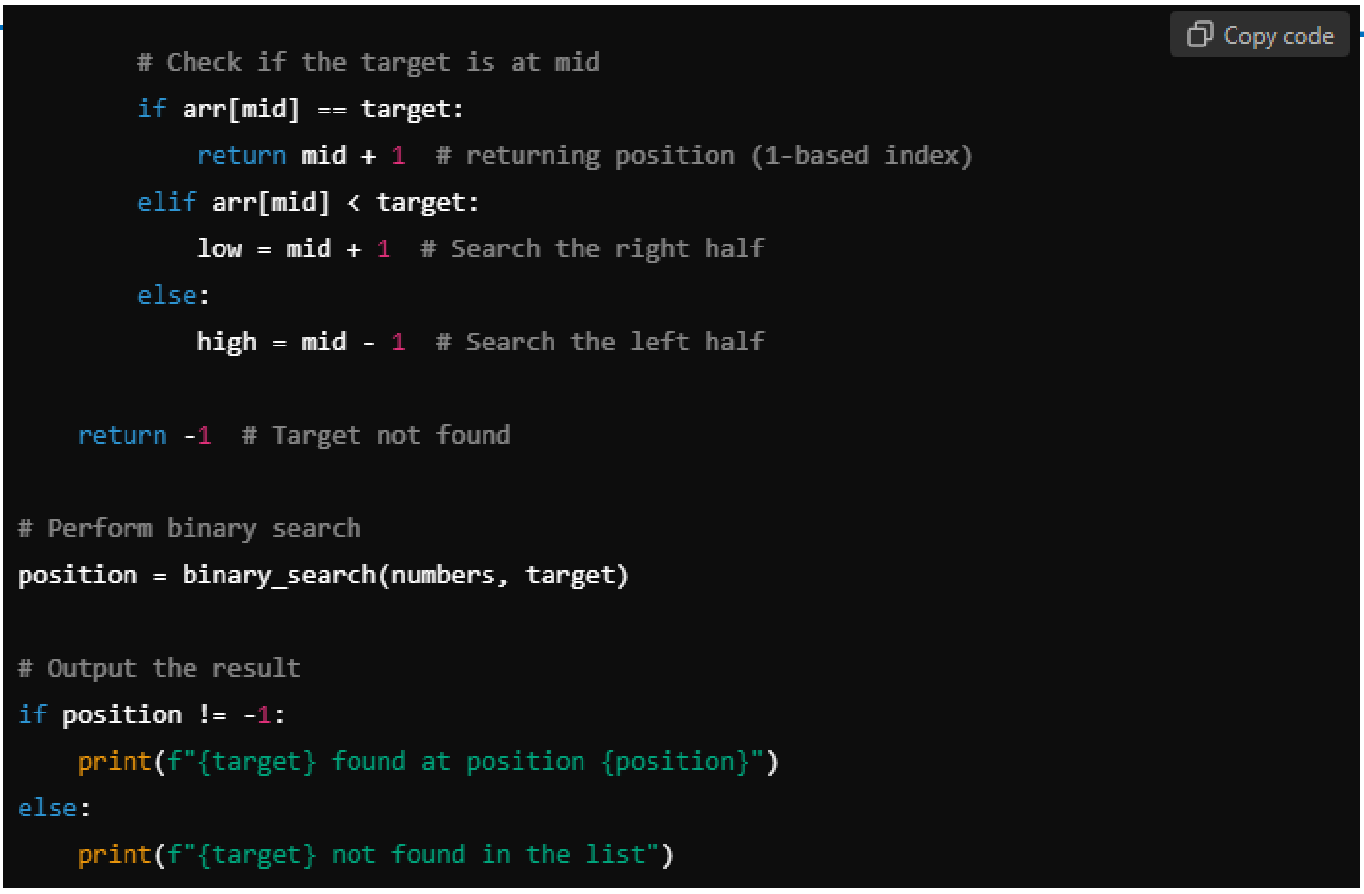
```python
# Linear Search Implementation

# Input list of numbers
numbers = list(map(int, input("Enter a list of numbers separated by spaces: ").split()))

# Input the target value to search
target = int(input("Enter the number to search: "))

# Linear search logic
found = False
for i in range(len(numbers)):
    if numbers[i] == target:
        found = True
        print(f"{target} found at position {i + 1}")
        break

if not found:
    print(f"{target} not found in the list")
```

3. Binary Search Implementation:


python
Copy code

```python
# Binary Search Implementation (works only on sorted lists)

# Input sorted list of numbers
numbers = list(map(int, input("Enter a sorted list of numbers separated by spaces: ").spli

# Input the target value to search
target = int(input("Enter the number to search: "))

# Binary search logic
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu



```python
        # Check if the target is at mid
        if arr[mid] == target:
            return mid + 1  # returning position (1-based index)
        elif arr[mid] < target:
            low = mid + 1  # Search the right half
        else:
            high = mid - 1  # Search the left half

    return -1  # Target not found

# Perform binary search
position = binary_search(numbers, target)

# Output the result
if position != -1:
    print(f"{target} found at position {position}")
else:
    print(f"{target} not found in the list")
```

Procedure Explanation:

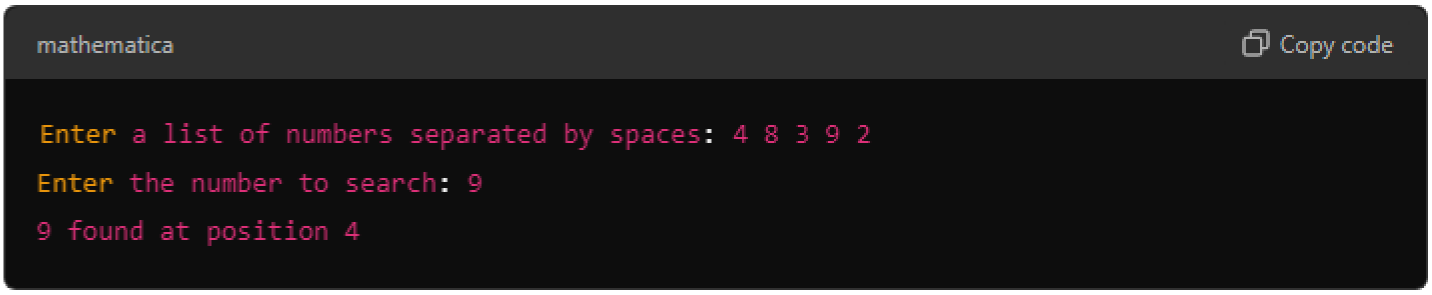
Linear Search:

1. **Step 1:** Define the function linear_search(arr, target) that iterates over the list. If the element is found, the index is returned.
2. **Step 2:** The user inputs a list of numbers and the target value to search for.
3. **Step 3:** Call the linear_search() function to check if the target is in the list.
4. **Step 4:** If the target is found, the function returns the index of the element. Otherwise, it returns -1.

Binary Search:

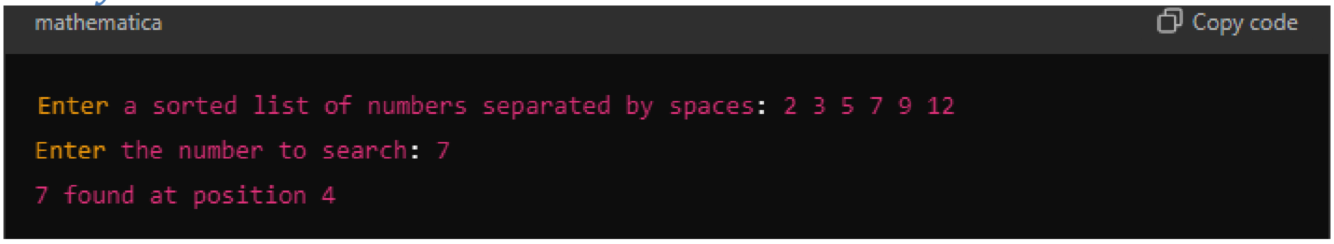
1. **Step 1:** Define the function binary_search(arr, target). This function works on a sorted list. It compares the middle element of the list to the target value and reduces the search range based on whether the target is smaller or larger than the middle element.
2. **Step 2-4:** The process continues until the target is found, or the search range becomes empty.
3. **Step 5:** The user inputs a **sorted list** and the target value.
4. **Step 6:** The binary_search() function is called to perform the search.
5. **Step 7:** If the target is found, the function returns the index; otherwise, it returns -1.

Example Output:



```
Enter a list of numbers separated by spaces: 4 8 3 9 2
Enter the number to search: 9
9 found at position 4
```

Binary Search:



```
Enter a sorted list of numbers separated by spaces: 2 3 5 7 9 12
Enter the number to search: 7
7 found at position 4
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

- **Linear Search** is simple and works on **unsorted** lists, but it's less efficient for large lists due to its O(n) time complexity.
- **Binary Search** is faster with O(log n) complexity but requires the list to be **sorted** beforehand. If the list is unsorted, you must sort it before using binary search.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

6. Python Lab Exercise: Selection Sort and Insertion Sort

Objective:

To implement **Selection Sort** and **Insertion Sort** algorithms to sort a list of numbers in Python.

Theory:

1. Selection Sort:

Selection Sort is a simple comparison-based sorting algorithm. The list is divided into two parts: the sorted part and the unsorted part. The algorithm repeatedly selects the smallest (or largest) element from the unsorted part and swaps it with the leftmost unsorted element, moving the boundary of the sorted part one element to the right.

- **Time Complexity:**$O(n^2)$ in all cases (where n is the number of elements).
- **Algorithm Steps:**
    1. Find the minimum element in the unsorted part of the list.
    2. Swap it with the first unsorted element.
    3. Move the boundary of the sorted part to the right.
    4. Repeat until the list is sorted.

2. Insertion Sort:

Insertion Sort builds the sorted list one element at a time. It picks an element from the unsorted part and places it in the correct position in the sorted part by comparing it with the already sorted elements.

- **Time Complexity:**$O(n^2)$ in the worst case, O(n) in the best case (already sorted list).
- **Algorithm Steps:**

1. Start with the second element (as the first element is considered sorted).
2. Compare the element with the elements before it.
3. Insert it into its correct position in the sorted part of the list.
4. Repeat until the entire list is sorted.

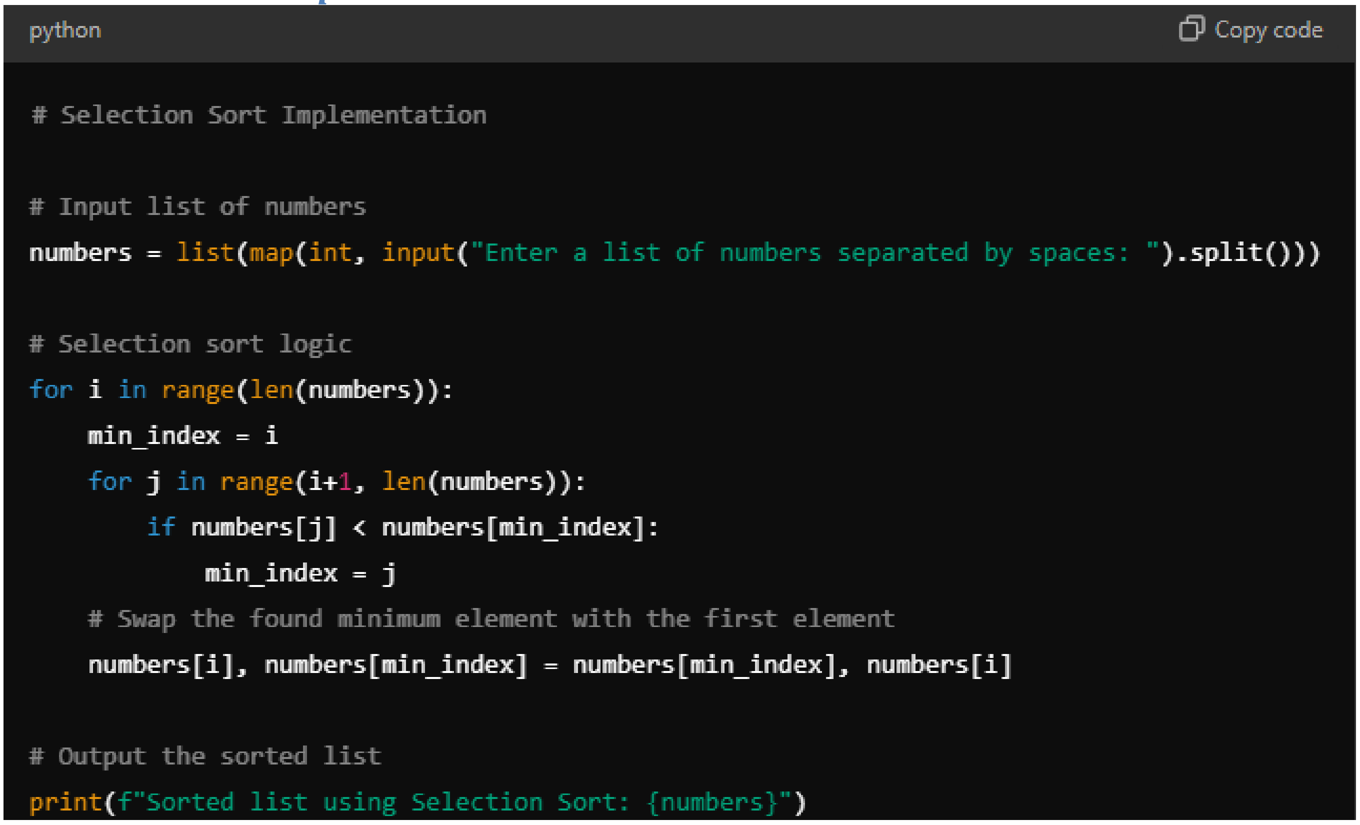
Procedure:

1. **Step 1:** Accept a list of numbers from the user.
2. **Step 2:** Implement Selection Sort.
3. **Step 3:** Implement Insertion Sort.
4. **Step 4:** Display the sorted list for both algorithms.
5. **Step 5:** Test both sorting algorithms with different inputs.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Code:

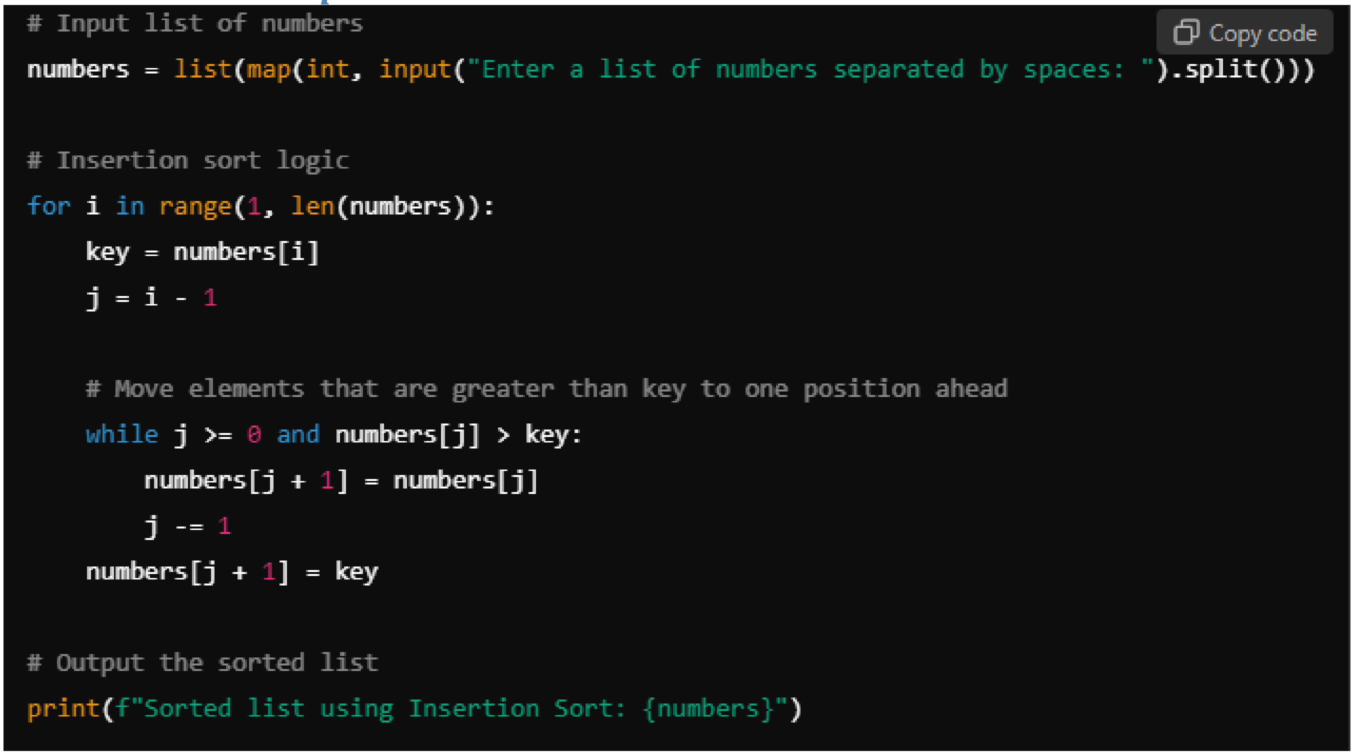
1. Selection Sort Implementation:


python Copy code

```python
# Selection Sort Implementation

# Input list of numbers
numbers = list(map(int, input("Enter a list of numbers separated by spaces: ").split()))

# Selection sort logic
for i in range(len(numbers)):
    min_index = i
    for j in range(i+1, len(numbers)):
        if numbers[j] < numbers[min_index]:
            min_index = j
    # Swap the found minimum element with the first element
    numbers[i], numbers[min_index] = numbers[min_index], numbers[i]

# Output the sorted list
print(f"Sorted list using Selection Sort: {numbers}")
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

2. Insertion Sort Implementation:



```python
# Input list of numbers
numbers = list(map(int, input("Enter a list of numbers separated by spaces: ").split()))

# Insertion sort logic
for i in range(1, len(numbers)):
    key = numbers[i]
    j = i - 1

    # Move elements that are greater than key to one position ahead
    while j >= 0 and numbers[j] > key:
        numbers[j + 1] = numbers[j]
        j -= 1
    numbers[j + 1] = key

# Output the sorted list
print(f"Sorted list using Insertion Sort: {numbers}")
```

Procedure Explanation:

Selection Sort:

1. **Step 1:** Define the function selection_sort(arr). The function iterates over the list to find the smallest element in the unsorted portion.
2. **Step 2:** The inner loop starts from the current element i and finds the minimum value in the unsorted part of the list.
3. **Step 3:** The minimum value is swapped with the first unsorted element.
4. **Step 4:** Input is taken as a list of numbers from the user.
5. **Step 5:** The sorted list is printed.

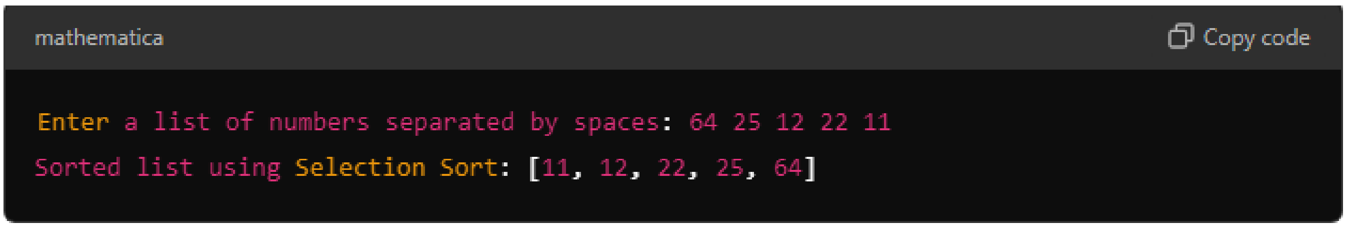
Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Insertion Sort:

1. **Step 1:** Define the function insertion_sort(arr). The function starts by assuming the first element is sorted.
2. **Step 2:** For each element from index 1 to the end, it compares the current element (key) to the elements before it.
3. **Step 3:** It shifts elements that are greater than the key to the right and inserts the key at its correct position.
4. **Step 4:** The list of numbers is taken as input.
5. **Step 5:** The sorted list is printed.

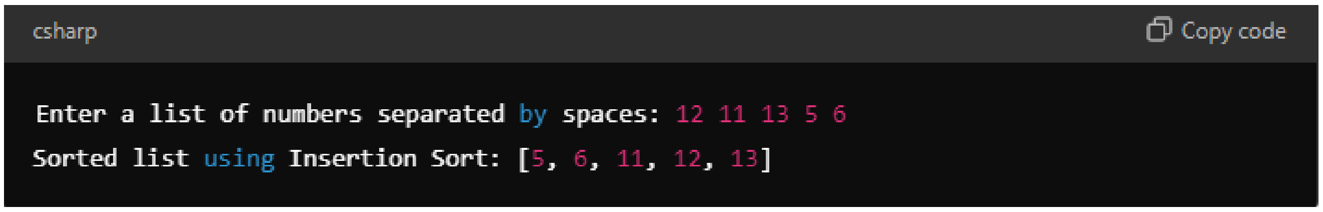
Example Output:

Selection Sort:



```
Enter a list of numbers separated by spaces: 64 25 12 22 11
Sorted list using Selection Sort: [11, 12, 22, 25, 64]
```

Insertion Sort:



```
Enter a list of numbers separated by spaces: 12 11 13 5 6
Sorted list using Insertion Sort: [5, 6, 11, 12, 13]
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

- **Selection Sort** is simple and efficient for small datasets, but it performs poorly on large lists due to its $O(n^2)$ time complexity.
- **Insertion Sort** is efficient for nearly sorted or small datasets and has a time complexity of $O(n^2)$ in the worst case but $O(n)$ in the best case.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

7. Python Lab Exercise: Merge Sort

Objective:

To implement the **Merge Sort** algorithm using Python to sort a list of numbers.

Theory:

Merge Sort is a divide-and-conquer algorithm that splits the list into smaller sublists until each sublist contains a single element (or is empty), and then merges those sublists to produce new sorted sublists until there is only one sublist remaining.

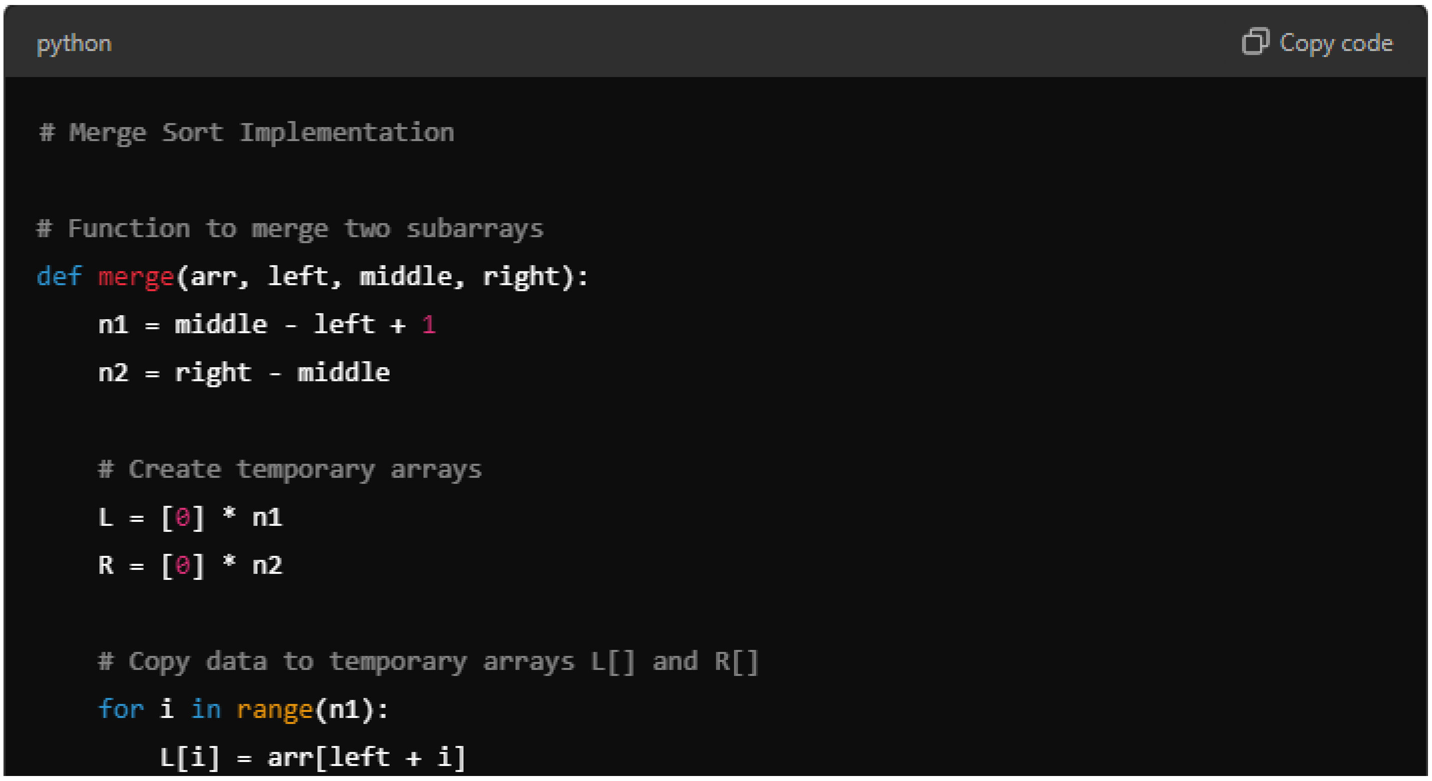
- **Time Complexity:**O(n log n) in all cases (where n is the number of elements).
- **Algorithm Steps:**
  1. If the list is empty or has one element, it is already sorted.
  2. Divide the unsorted list into two approximately equal halves.
  3. Recursively apply the merge sort to each half.
  4. Merge the sorted halves to produce the sorted list.

Procedure:

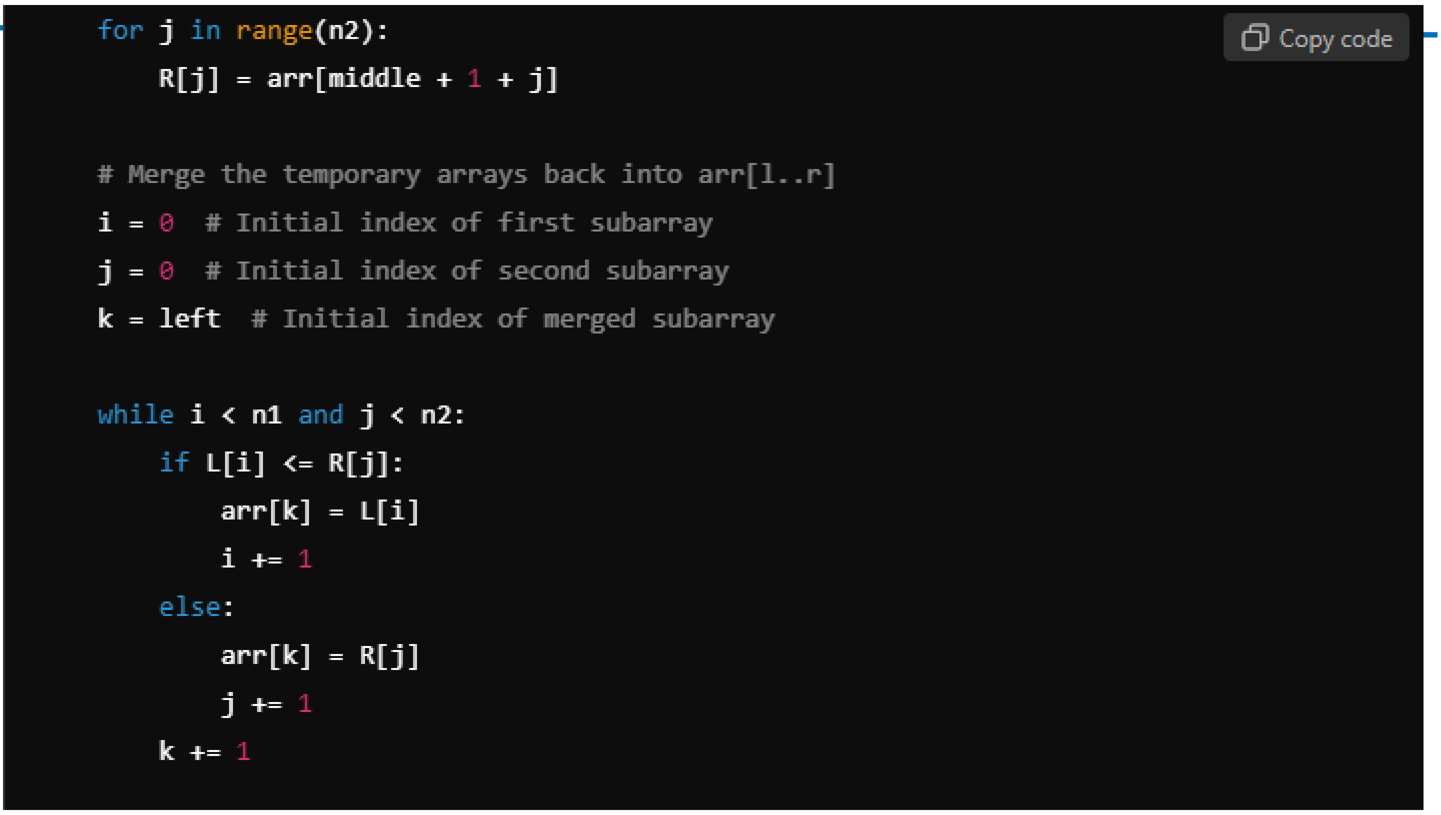
1. **Step 1:** Accept a list of numbers from the user.
2. **Step 2:** Implement the Merge Sort algorithm.
3. **Step 3:** Print the sorted list.
4. **Step 4:** Test the code with different inputs.

Code:

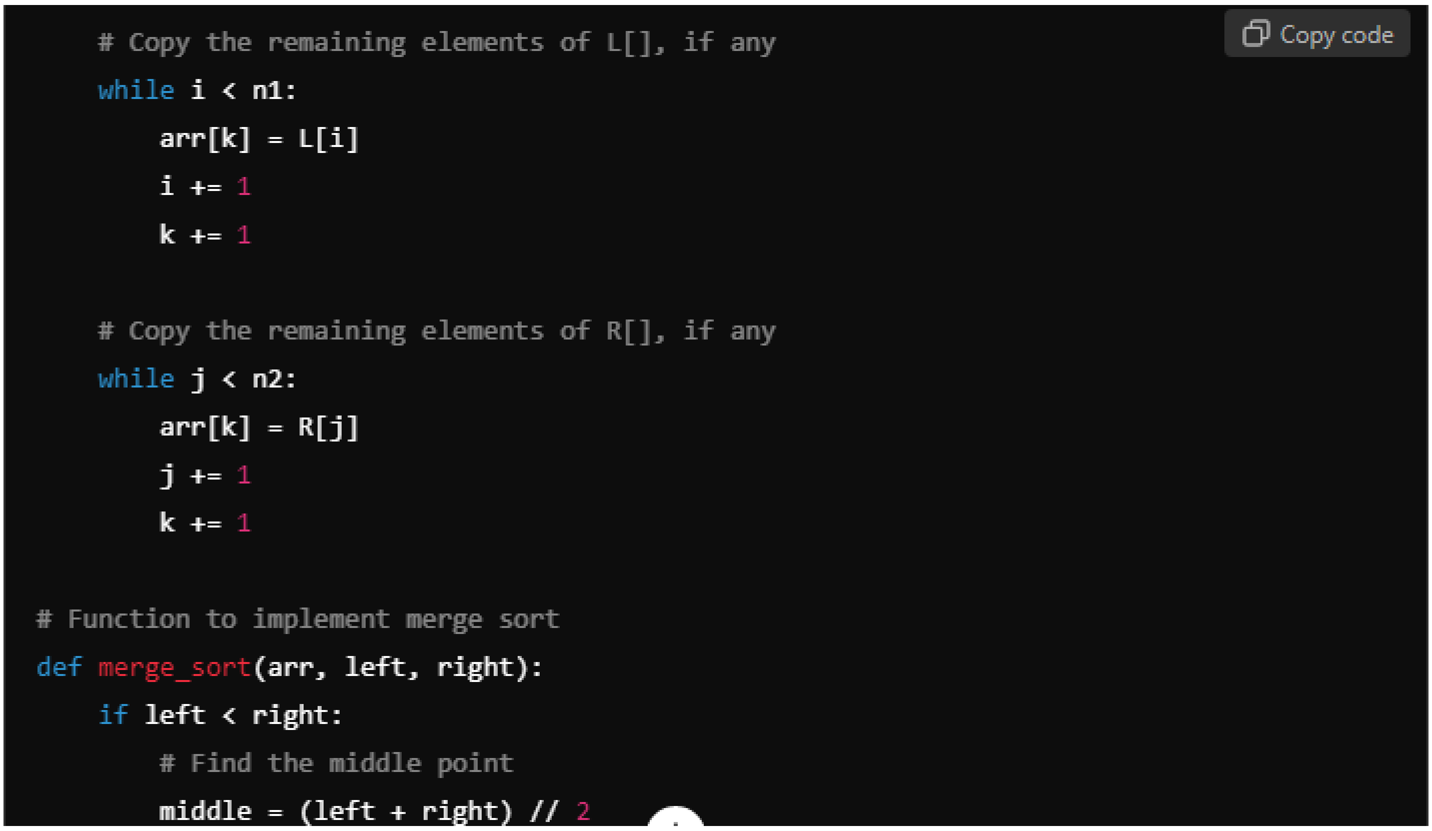
Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu



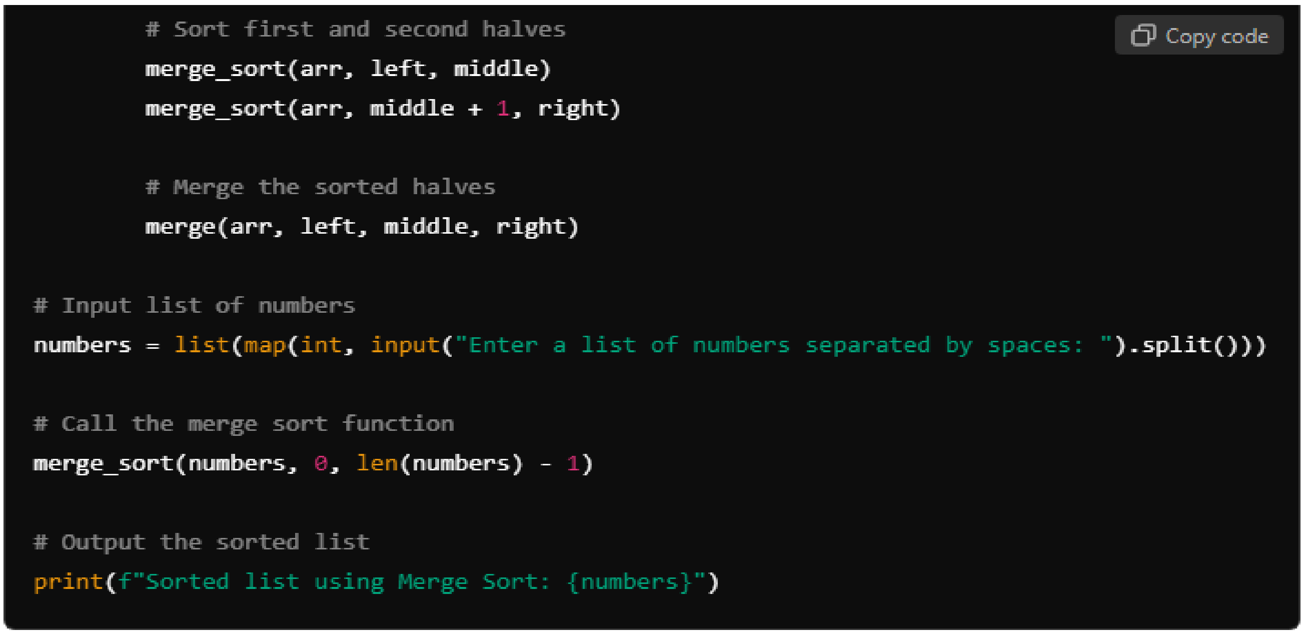
```python
# Merge Sort Implementation

# Function to merge two subarrays
def merge(arr, left, middle, right):
    n1 = middle - left + 1
    n2 = right - middle

    # Create temporary arrays
    L = [0] * n1
    R = [0] * n2

    # Copy data to temporary arrays L[] and R[]
    for i in range(n1):
        L[i] = arr[left + i]
```



```
    for j in range(n2):
        R[j] = arr[middle + 1 + j]

    # Merge the temporary arrays back into arr[l..r]
    i = 0  # Initial index of first subarray
    j = 0  # Initial index of second subarray
    k = left  # Initial index of merged subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
```


Copy code

```
    # Copy the remaining elements of L[], if any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# Function to implement merge sort
def merge_sort(arr, left, right):
    if left < right:
        # Find the middle point
        middle = (left + right) // 2
```


Copy code

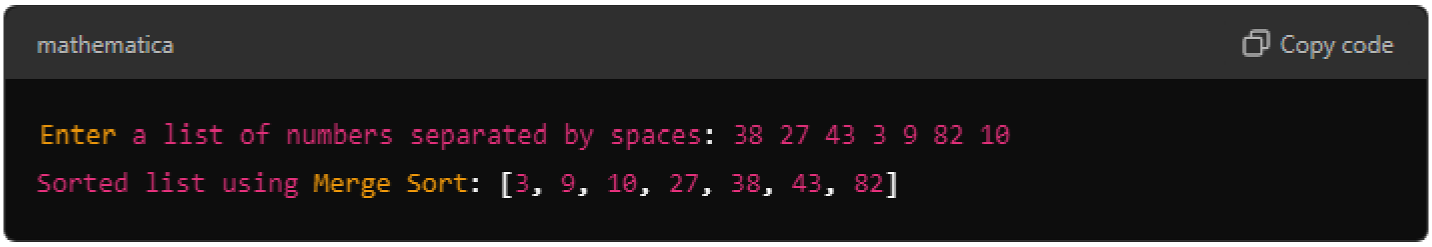
```
        # Sort first and second halves
        merge_sort(arr, left, middle)
        merge_sort(arr, middle + 1, right)

        # Merge the sorted halves
        merge(arr, left, middle, right)

# Input list of numbers
numbers = list(map(int, input("Enter a list of numbers separated by spaces: ").split()))

# Call the merge sort function
merge_sort(numbers, 0, len(numbers) - 1)

# Output the sorted list
print(f"Sorted list using Merge Sort: {numbers}")
```

Procedure Explanation:

1. **Step 1:** Define the merge(left, right) function, which merges two sorted lists into one sorted list.
    - The function uses two pointers to track the current index of each half. It compares elements and appends the smaller one to the sorted_list.
    - Any remaining elements from either half are added to the sorted_list after the main loop.
2. **Step 4:** Define the merge_sort(arr) function to implement the merge sort algorithm.
    - If the list has one or no elements, it is returned as is.
    - The list is split into two halves using slicing.
    - The merge_sort function is called recursively for both halves.
3. **Step 7:** Accept a list of numbers from the user.
4. **Step 8:** Call the merge_sort function with the input list and print the sorted list.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Example Output:



```
Enter a list of numbers separated by spaces: 38 27 43 3 9 82 10
Sorted list using Merge Sort: [3, 9, 10, 27, 38, 43, 82]
```

Conclusion:

- **Merge Sort** is an efficient, stable sorting algorithm that is particularly useful for large datasets. It consistently performs at O(n log n) time complexity, making it one of the most efficient sorting algorithms available.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

8.Python Lab Exercise: First n Prime Numbers

Objective:

To find and print the first nnn prime numbers using Python.

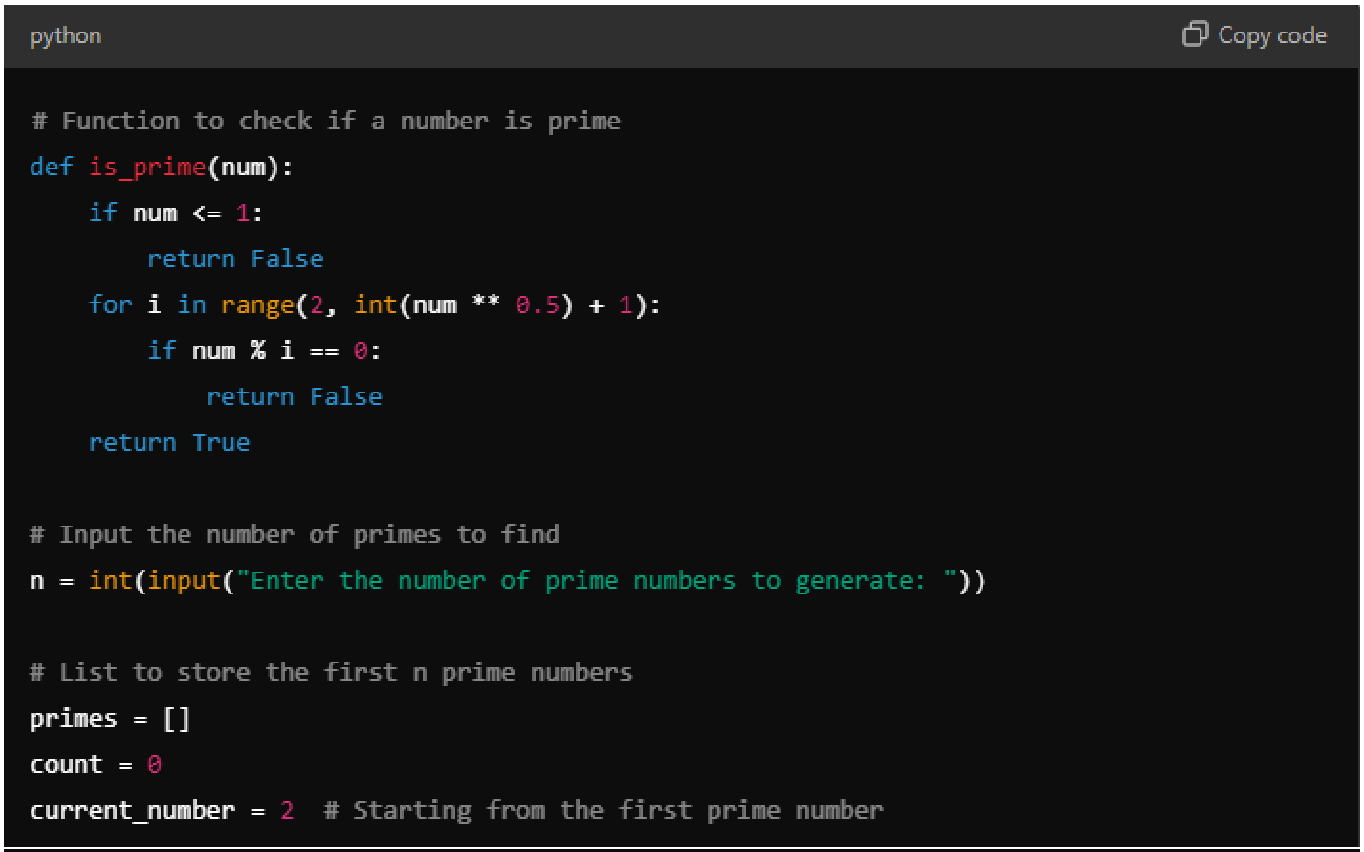
Theory:

A **prime number** is a natural number greater than 1 that has no positive divisors other than 1 and itself. To determine whether a number is prime, we can check if it is divisible by any integer from 2 to the square root of the number.

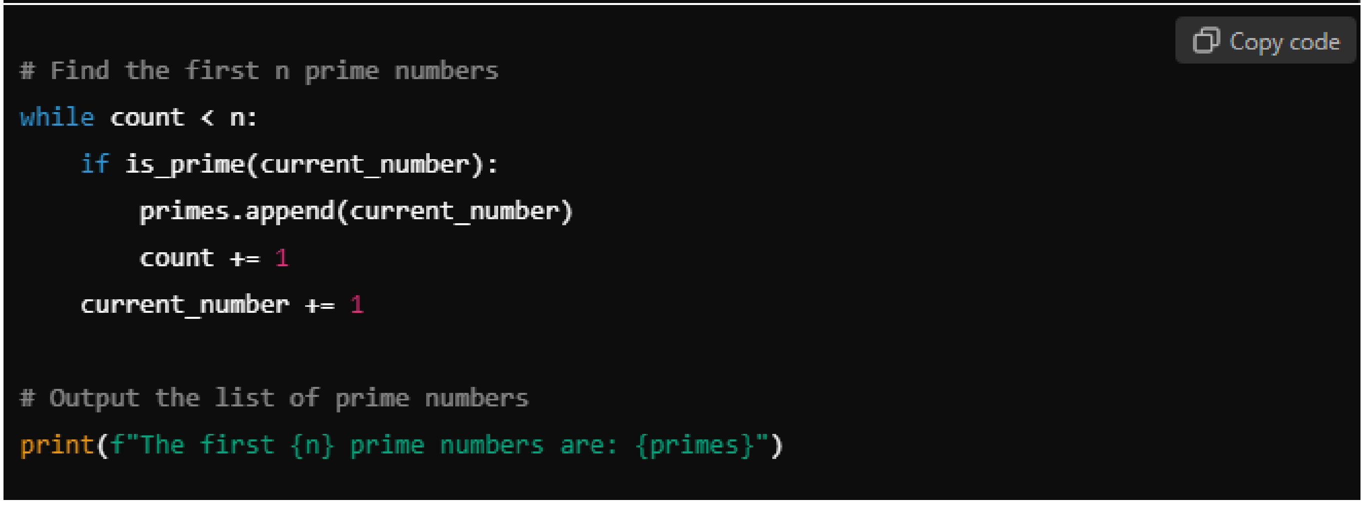
Procedure:

1. **Step 1:** Accept the value of nnn from the user (the number of prime numbers to find).
2. **Step 2:** Implement a function to check if a number is prime.
3. **Step 3:** Generate the first nnn prime numbers.
4. **Step 4:** Print the list of prime numbers.
5. **Step 5:** Test the code with different values of nnn.

Code:


python Copy code

```python
# Function to check if a number is prime
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

# Input the number of primes to find
n = int(input("Enter the number of prime numbers to generate: "))

# List to store the first n prime numbers
primes = []
count = 0
current_number = 2  # Starting from the first prime number
```


Copy code

```python
# Find the first n prime numbers
while count < n:
    if is_prime(current_number):
        primes.append(current_number)
        count += 1
    current_number += 1

# Output the list of prime numbers
print(f"The first {n} prime numbers are: {primes}")
```

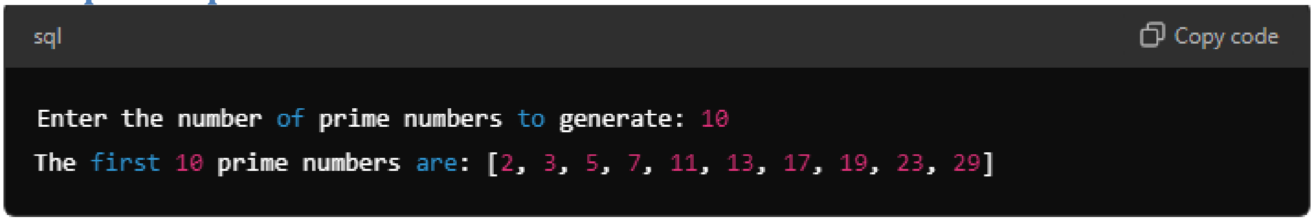
Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Procedure Explanation:

1. **Step 1:** Define the is_prime(num) function to check if a number is prime:
   - If the number is less than or equal to 1, it returns False.
   - The function iterates from 2 to the square root of the number. If the number is divisible by any of these, it returns False. Otherwise, it returns True.
2. **Step 2:** Define the first_n_primes(n) function to generate the first nnn prime numbers:
   - It initializes an empty list primes and starts checking from the number 2.
   - As long as the length of the primes list is less than nnn, it checks if the candidate number is prime using the is_prime function.
   - If it is prime, it appends it to the primes list and increments the candidate number.
3. **Step 3:** Accept user input for the number of primes to find.

4.Step 4: Call the function to get the list of prime numbers and print the result.

Example Output:



```
Enter the number of prime numbers to generate: 10
The first 10 prime numbers are: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

The implemented algorithm efficiently finds the first n prime numbers using a basic prime-checking function. This exercise demonstrates how to use loops and conditional statements in Python to solve a mathematical problem.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

9. Python Lab Exercise: Matrix Multiplication

Objective:

To perform multiplication of two matrices using Python.

Theory:

Matrix multiplication is a binary operation that produces a matrix from two matrices. The number of columns in the first matrix must equal the number of rows in the second matrix. The element at position (i,j)(i, j)(i,j) in the resulting matrix is calculated by taking the dot product of the iiith row of the first matrix and the jjjth column of the second matrix.

- **Matrix A:** of dimensions m×nm \times nm×n

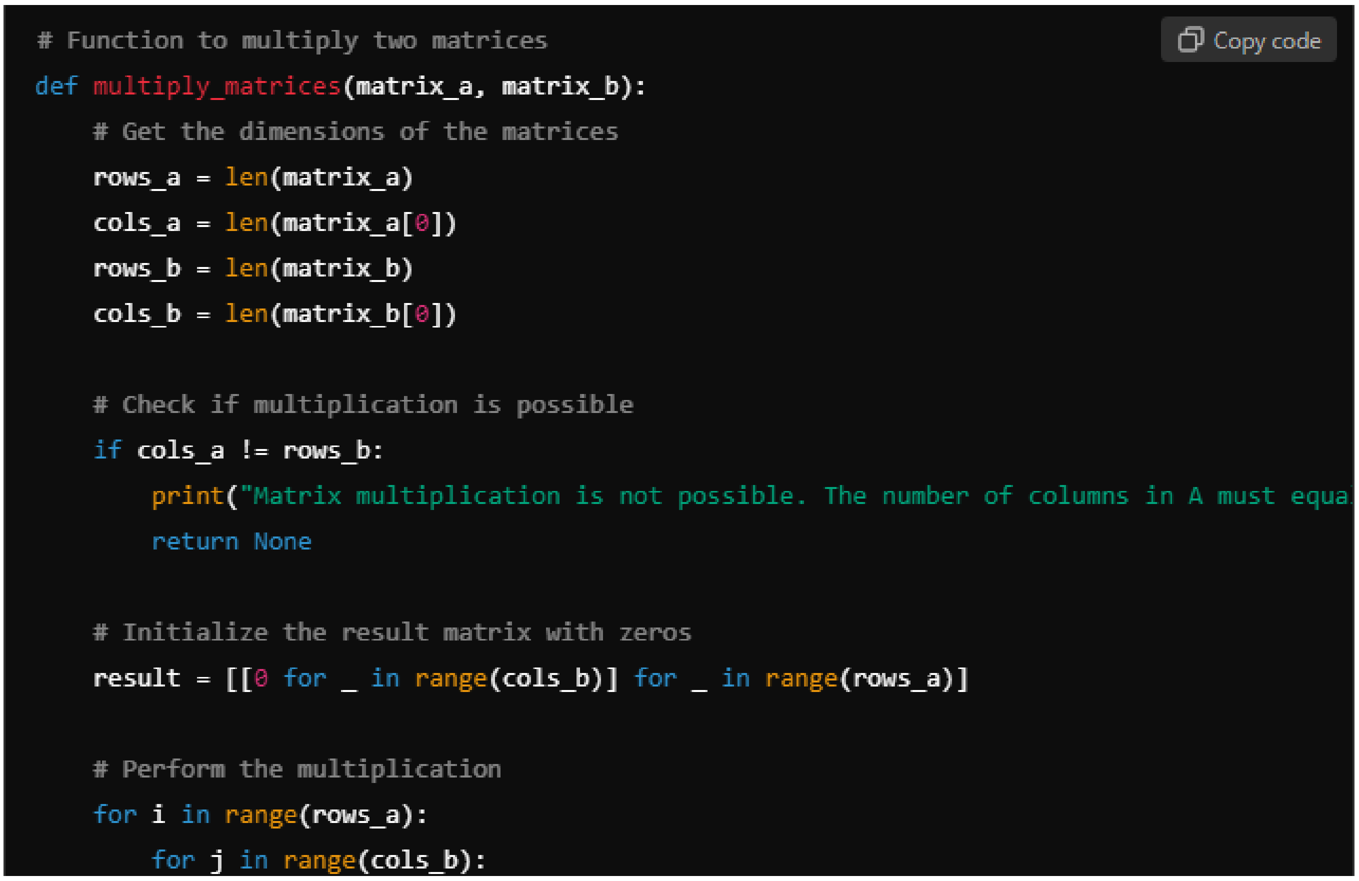
- **Matrix B:** of dimensions n×pn \times pn×p
- **Resulting Matrix C:** of dimensions m×pm \times pm×p

Procedure:

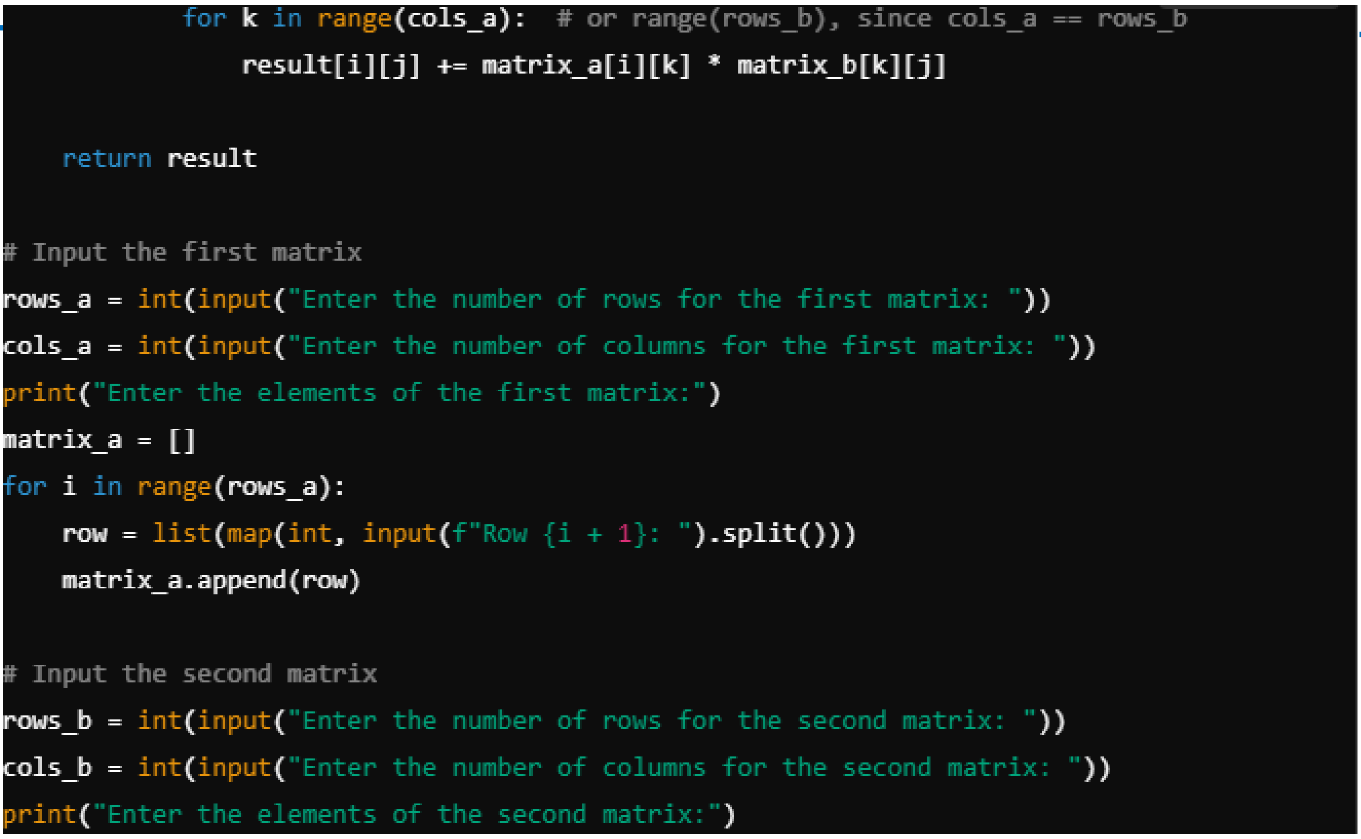
1. **Step 1:** Accept the dimensions of the matrices from the user.
2. **Step 2:** Input the elements of both matrices.
3. **Step 3:** Implement a function to multiply the two matrices.
4. **Step 4:** Print the resulting matrix.
5. **Step 5:** Test the code with different matrices.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

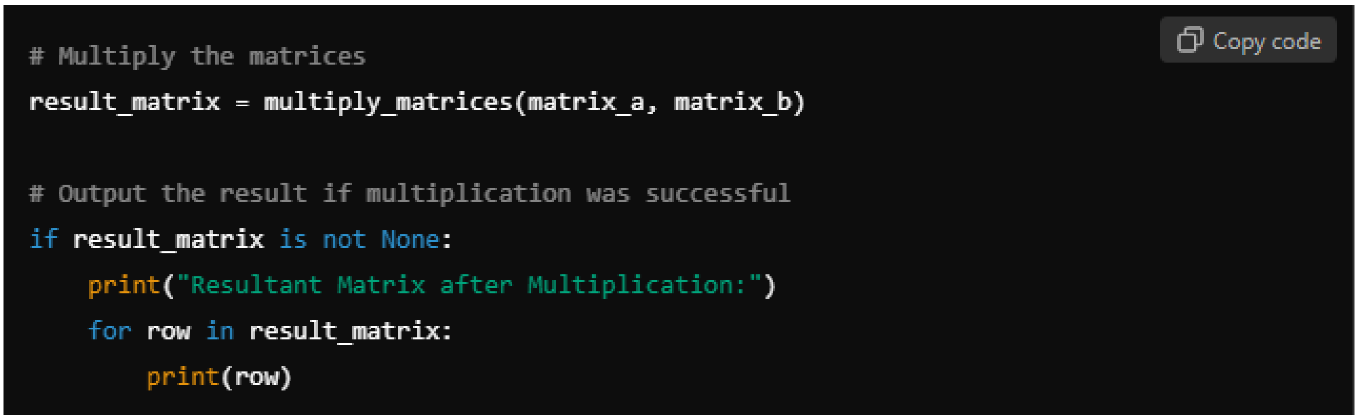
Code:



```
# Function to multiply two matrices
def multiply_matrices(matrix_a, matrix_b):
    # Get the dimensions of the matrices
    rows_a = len(matrix_a)
    cols_a = len(matrix_a[0])
    rows_b = len(matrix_b)
    cols_b = len(matrix_b[0])

    # Check if multiplication is possible
    if cols_a != rows_b:
        print("Matrix multiplication is not possible. The number of columns in A must equa
        return None

    # Initialize the result matrix with zeros
    result = [[0 for _ in range(cols_b)] for _ in range(rows_a)]

    # Perform the multiplication
    for i in range(rows_a):
        for j in range(cols_b):
```



```
        for k in range(cols_a):  # or range(rows_b), since cols_a == rows_b
            result[i][j] += matrix_a[i][k] * matrix_b[k][j]

    return result

# Input the first matrix
rows_a = int(input("Enter the number of rows for the first matrix: "))
cols_a = int(input("Enter the number of columns for the first matrix: "))
print("Enter the elements of the first matrix:")
matrix_a = []
for i in range(rows_a):
    row = list(map(int, input(f"Row {i + 1}: ").split()))
    matrix_a.append(row)

# Input the second matrix
rows_b = int(input("Enter the number of rows for the second matrix: "))
cols_b = int(input("Enter the number of columns for the second matrix: "))
print("Enter the elements of the second matrix:")
```



```
# Multiply the matrices
result_matrix = multiply_matrices(matrix_a, matrix_b)

# Output the result if multiplication was successful
if result_matrix is not None:
    print("Resultant Matrix after Multiplication:")
    for row in result_matrix:
        print(row)
```

Procedure Explanation:

1. **Step 1:** Define the input_matrix(rows, cols) function to read matrix elements from the user.

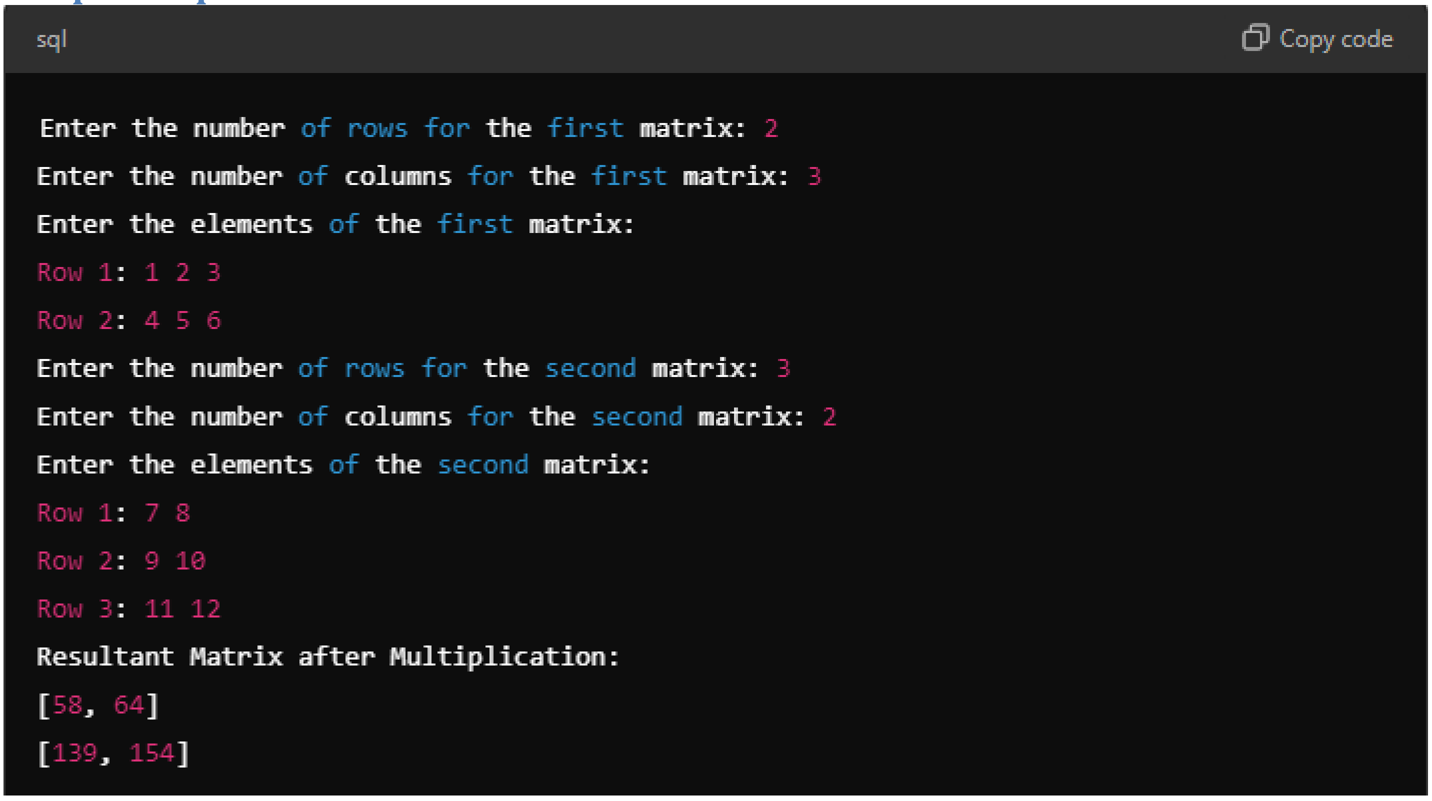
  - The function iterates over the number of rows, prompting the user for each row's elements.

2. **Step 2:** Define the multiply_matrices(A, B) function to multiply two matrices:

  - It initializes the resulting matrix CCC with zeros, using list comprehension.
  - The function contains three nested loops to calculate each element of the resulting matrix by summing the products of corresponding elements from matrix AAA and matrix BBB.

3. **Step 4:** Accept dimensions and input elements for both matrices.
4. **Step 5:** Check if the matrices are compatible for multiplication (i.e., columns of AAA must equal rows of BBB).
5. **Step 6:** If compatible, the function is called to multiply the matrices.
6. **Step 7:** Print the resulting matrix.

Example Output:



```
Enter the number of rows for the first matrix: 2
Enter the number of columns for the first matrix: 3
Enter the elements of the first matrix:
Row 1: 1 2 3
Row 2: 4 5 6
Enter the number of rows for the second matrix: 3
Enter the number of columns for the second matrix: 2
Enter the elements of the second matrix:
Row 1: 7 8
Row 2: 9 10
Row 3: 11 12
Resultant Matrix after Multiplication:
[58, 64]
[139, 154]
```

Conclusion:

The implemented code successfully multiplies two matrices by checking their compatibility and applying the matrix multiplication algorithm. This exercise illustrates the use of nested loops and user input handling in Python.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

10.Python Lab Exercise: Word Count with Command Line Arguments

Objective:

To create a Python program that takes a string input as a command line argument and counts the number of words in that string.

Theory:

Command line arguments allow users to pass inputs to a program when executing it from the terminal or command prompt. In Python, the sys module provides access to command line arguments through sys.argv, which is a list where the first element is the name of the script and subsequent elements are the arguments passed.

Procedure:

1. **Step 1:** Import the sys module to access command line arguments.
2. **Step 2:** Check if the correct number of arguments is provided.
3. **Step 3:** Count the words in the provided string.
4. **Step 4:** Print the word count.
5. **Step 5:** Run the program from the command line with arguments.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

SriChandrasekharendraSaraswathiViswaMahavidyalaya

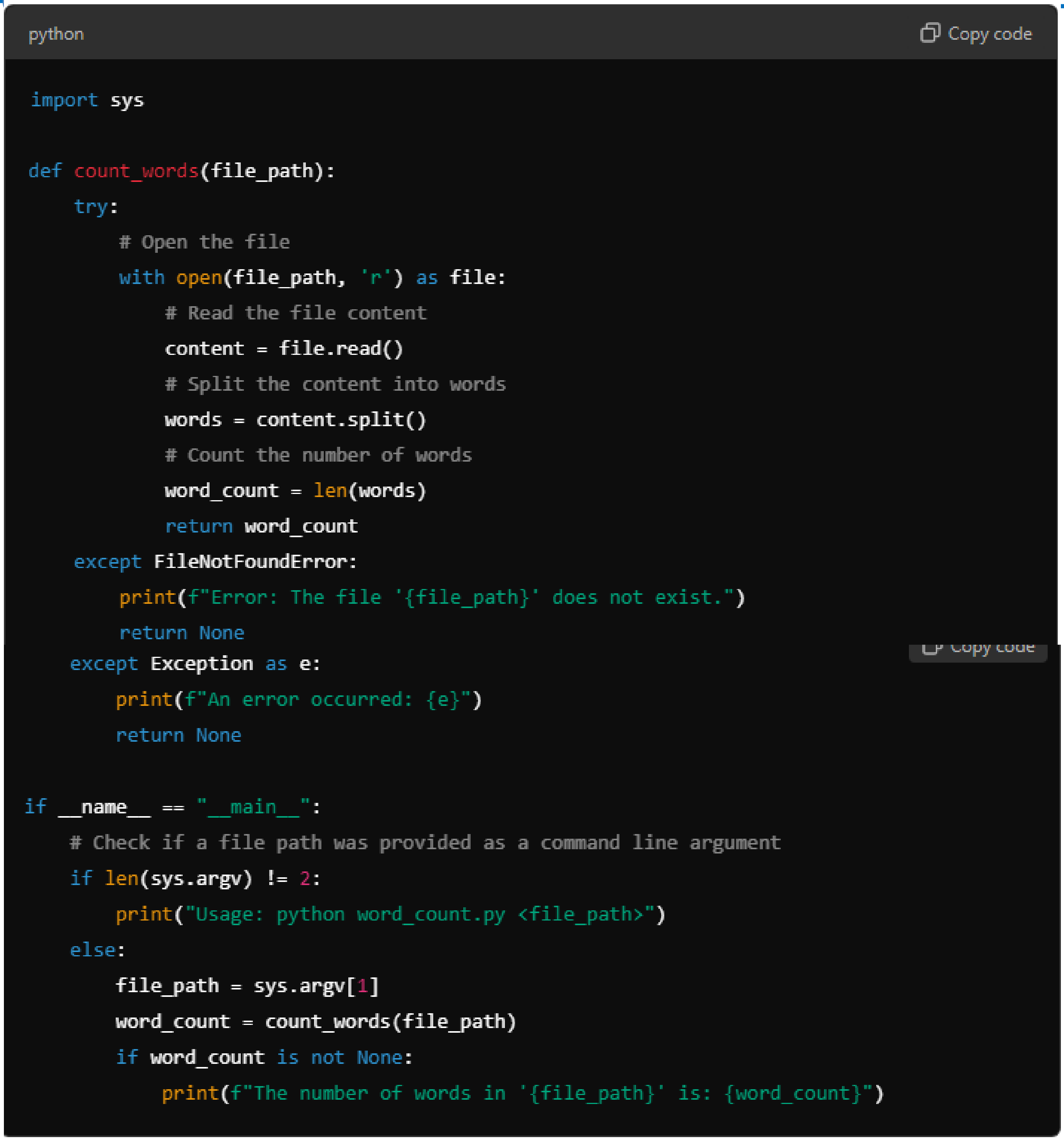
[SCSVMV**]**

[Deemed to be University U/S 3 of the UGC Act 1956] Accredited with
'A' Grade by NAAC

Enathur,Kanchipuram–631561

Department of Computer Science and Engineering

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu



```python
import sys

def count_words(file_path):
    try:
        # Open the file
        with open(file_path, 'r') as file:
            # Read the file content
            content = file.read()
            # Split the content into words
            words = content.split()
            # Count the number of words
            word_count = len(words)
            return word_count
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' does not exist.")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

if __name__ == "__main__":
    # Check if a file path was provided as a command line argument
    if len(sys.argv) != 2:
        print("Usage: python word_count.py <file_path>")
    else:
        file_path = sys.argv[1]
        word_count = count_words(file_path)
        if word_count is not None:
            print(f"The number of words in '{file_path}' is: {word_count}")
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Procedure Explanation:

1. **Step 1:** Import the sys module to handle command line arguments.
2. **Step 2:** Define the count_words(input_string) function that splits the input string into words using the split() method and returns the count of words.
3. **Step 3:** The script checks if the user has provided at least one argument (the input string). If not, it prints usage instructions and exits.
4. **Step 4:** It combines all command line arguments (excluding the script name) into a single string.
5. **Step 5:** Calls the count_words function to get the word count.
6. **Step 6:** Prints the result.

How to Run the Program:

1. Save the code in a file named word_count.py.
2. Open your command line or terminal.
3. Navigate to the directory where the script is saved.
4. Run the script with a string argument enclosed in quotes. For example:

```bash
python word_count.py sample.txt
```

Example Output:

```csharp
The number of words in 'sample.txt' is: 42
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

This program demonstrates how to handle command line arguments in Python and perform basic string manipulation. It efficiently counts the number of words in a string provided as an input when the script is executed, showcasing practical usage of the sys module.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

11.Python Lab Exercise: Finding Most Frequent Words in a Text File

Objective:

To read a text file and find the most frequent words within that file using Python.

Theory:

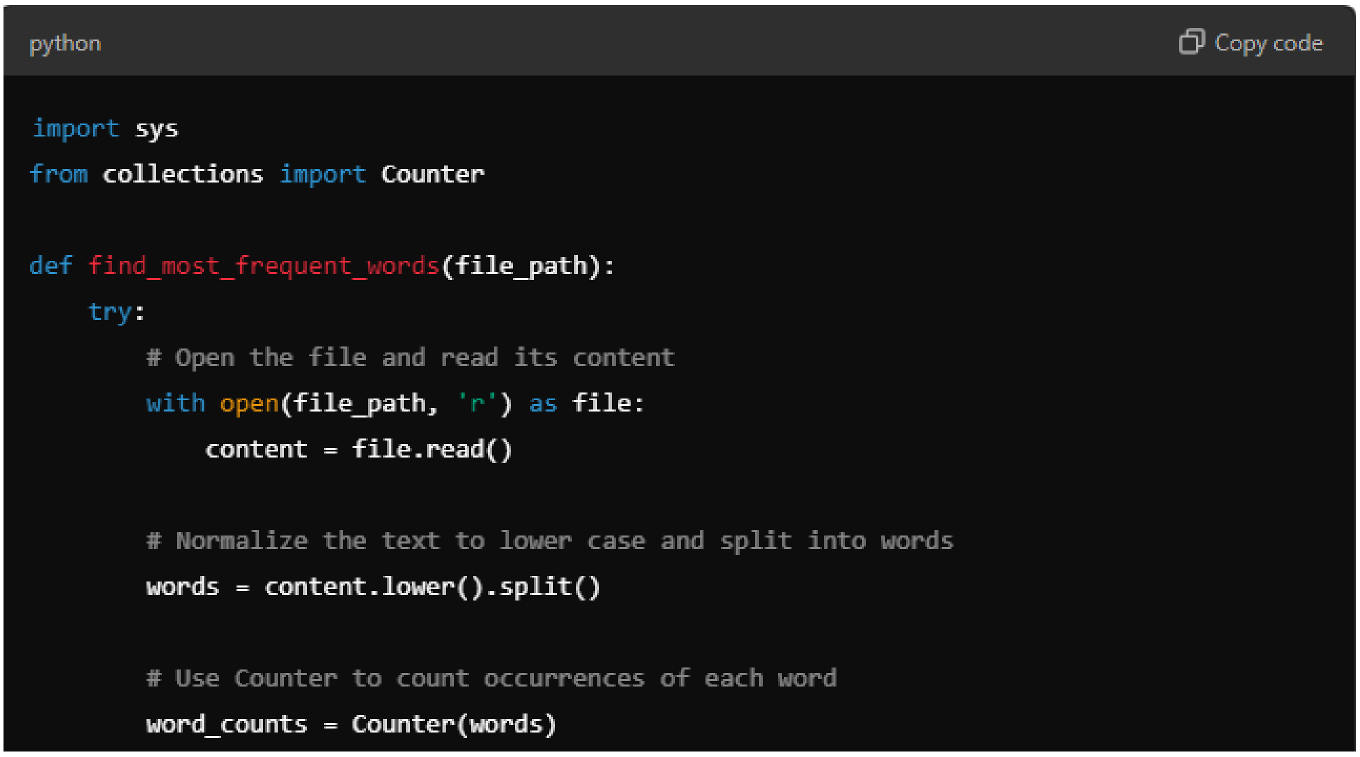
The frequency of words in a text can be determined by reading the contents of the file, splitting the text into individual words, and counting occurrences. The collections module in Python provides a convenient Counter class that can help with counting hashable objects like words.

Procedure:

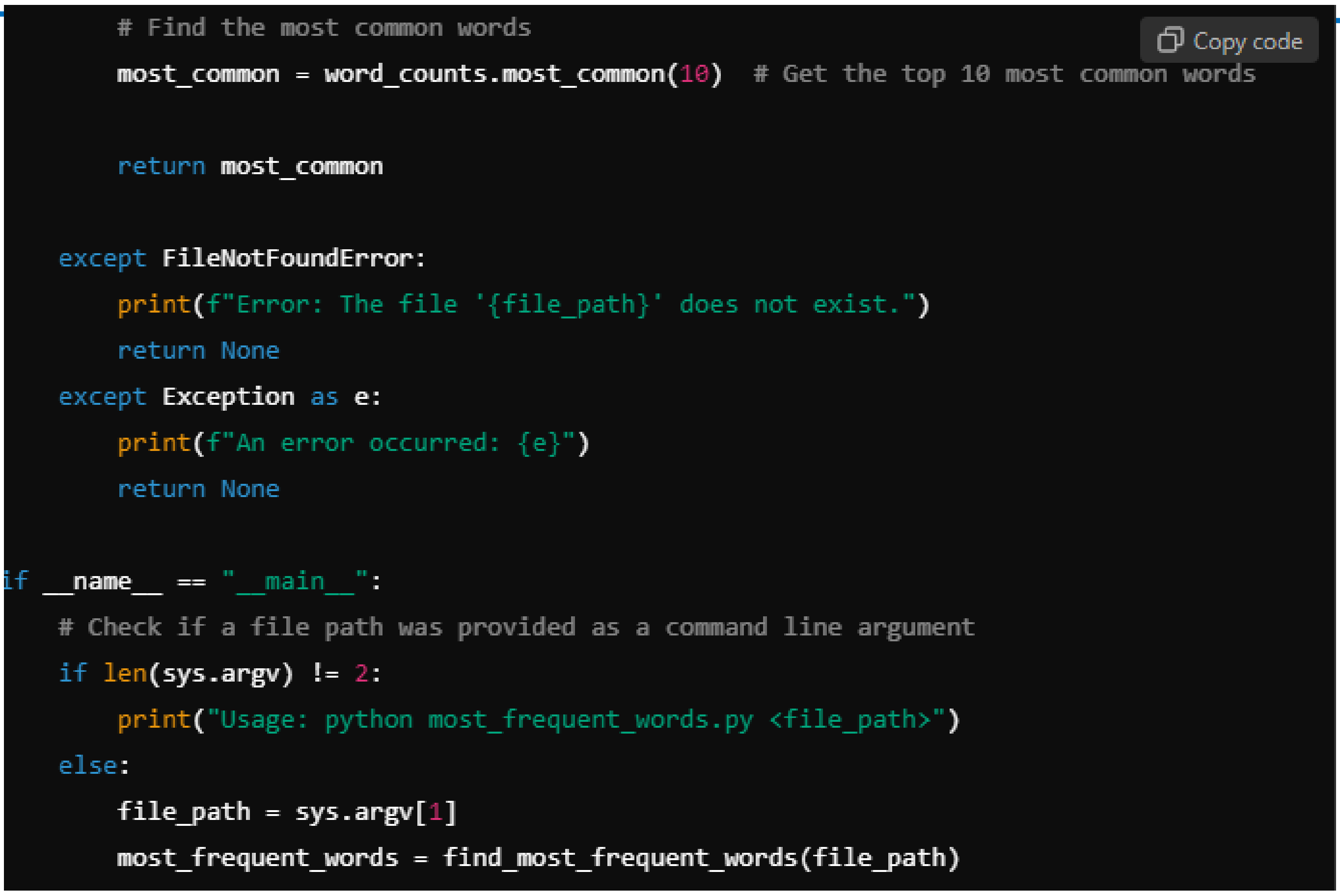
1. **Step 1:** Read the contents of a text file.
2. **Step 2:** Normalize the text by converting it to lowercase and removing punctuation.
3. **Step 3:** Split the text into words and count their occurrences.
4. **Step 4:** Identify the most frequent words.
5. **Step 5:** Print the results.
6. **Step 6:** Test the code with different text files.

Code:

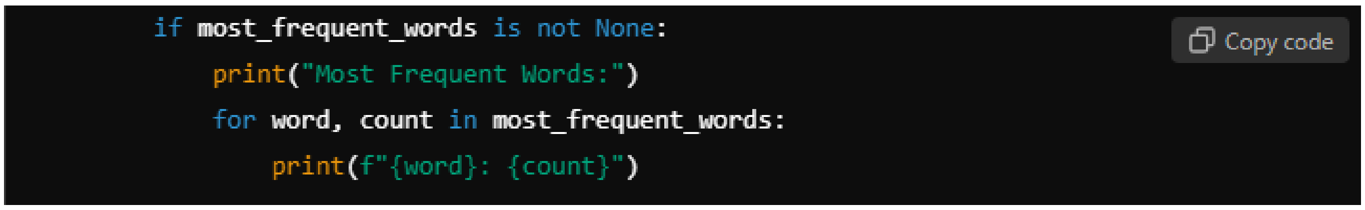
Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu


python
Copy code

```python
import sys
from collections import Counter

def find_most_frequent_words(file_path):
    try:
        # Open the file and read its content
        with open(file_path, 'r') as file:
            content = file.read()

        # Normalize the text to lower case and split into words
        words = content.lower().split()

        # Use Counter to count occurrences of each word
        word_counts = Counter(words)
```



```
        # Find the most common words
        most_common = word_counts.most_common(10)  # Get the top 10 most common words

        return most_common

    except FileNotFoundError:
        print(f"Error: The file '{file_path}' does not exist.")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

if __name__ == "__main__":
    # Check if a file path was provided as a command line argument
    if len(sys.argv) != 2:
        print("Usage: python most_frequent_words.py <file_path>")
    else:
        file_path = sys.argv[1]
        most_frequent_words = find_most_frequent_words(file_path)
```



```
        if most_frequent_words is not None:
            print("Most Frequent Words:")
            for word, count in most_frequent_words:
                print(f"{word}: {count}")
```

Procedure Explanation:

1. **Step 1:** The read_file(file_path) function opens and reads the file contents, returning the text.
2. **Step 2:** The count_words(text) function normalizes the text by converting it to lowercase and using a regular expression to extract words. The Counter class counts occurrences of each word.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

3. **Step 3:** The most_frequent_words(counter, n=10) function retrieves the nnn most common words from the counter.
4. **Step 4:** The main execution block prompts the user for the file path, reads the file, counts the words, and prints the most frequent words. It includes error handling for file not found and other exceptions.

How to Run the Program:

1. Save the code in a file named most_frequent_words.py.
2. Create a text file (e.g., sample.txt) with some content.
3. Open your command line or terminal.
4. Navigate to the directory where the script is saved.
5. Run the script:

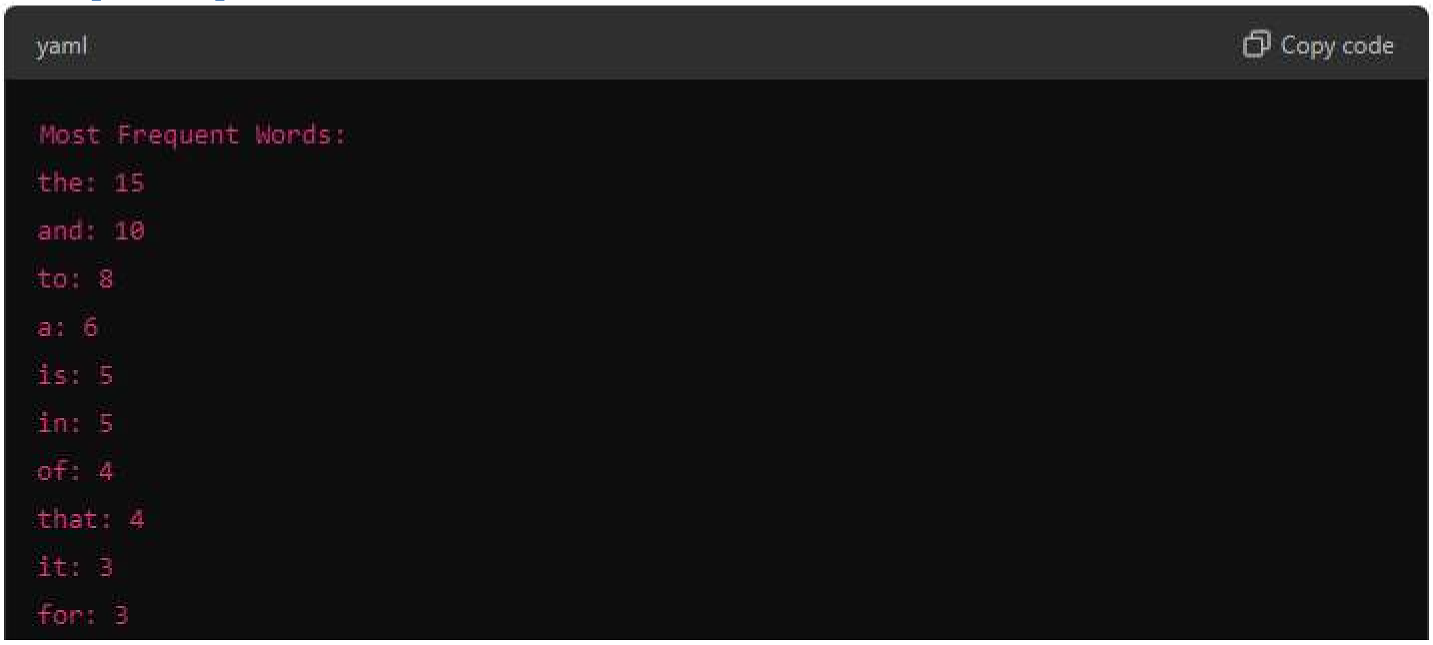
```bash
python most_frequent_words.py sample.txt
```

6. Enter the path to your text file when prompted:

   Enter the path to the text file: sample.txt

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Example Output:



yaml Copy code

```yaml
Most Frequent Words:
the: 15
and: 10
to: 8
a: 6
is: 5
in: 5
of: 4
that: 4
it: 3
for: 3
```

Conclusion:

This program effectively demonstrates how to read a text file, process its content, and count word occurrences using Python. It showcases the utility of the Counter class for frequency analysis and illustrates basic file handling and string manipulation techniques.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

12.Python Lab Exercise: Simulating Elliptical Orbits in Pygame

Objective:

To create a simple simulation of elliptical orbits using Pygame, demonstrating the principles of orbital mechanics.

Theory:

In celestial mechanics, an elliptical orbit is defined by two foci. The center of the ellipse is at the midpoint between the two foci. The distance from any point on the ellipse to one focus plus the distance to the other focus is constant.

To simulate an elliptical orbit, we can use the parametric equations of an ellipse:

- x(t)=a·cos(t)x(t) = a \cdot \cos(t)x(t)=a·cos(t)
- y(t)=b·sin(t)y(t) = b \cdot \sin(t)y(t)=b·sin(t)

Where:

- aaa is the semi-major axis (half the length of the longest diameter).
- bbb is the semi-minor axis (half the length of the shortest diameter).
- ttt is the parameter that varies over time.

Procedure:

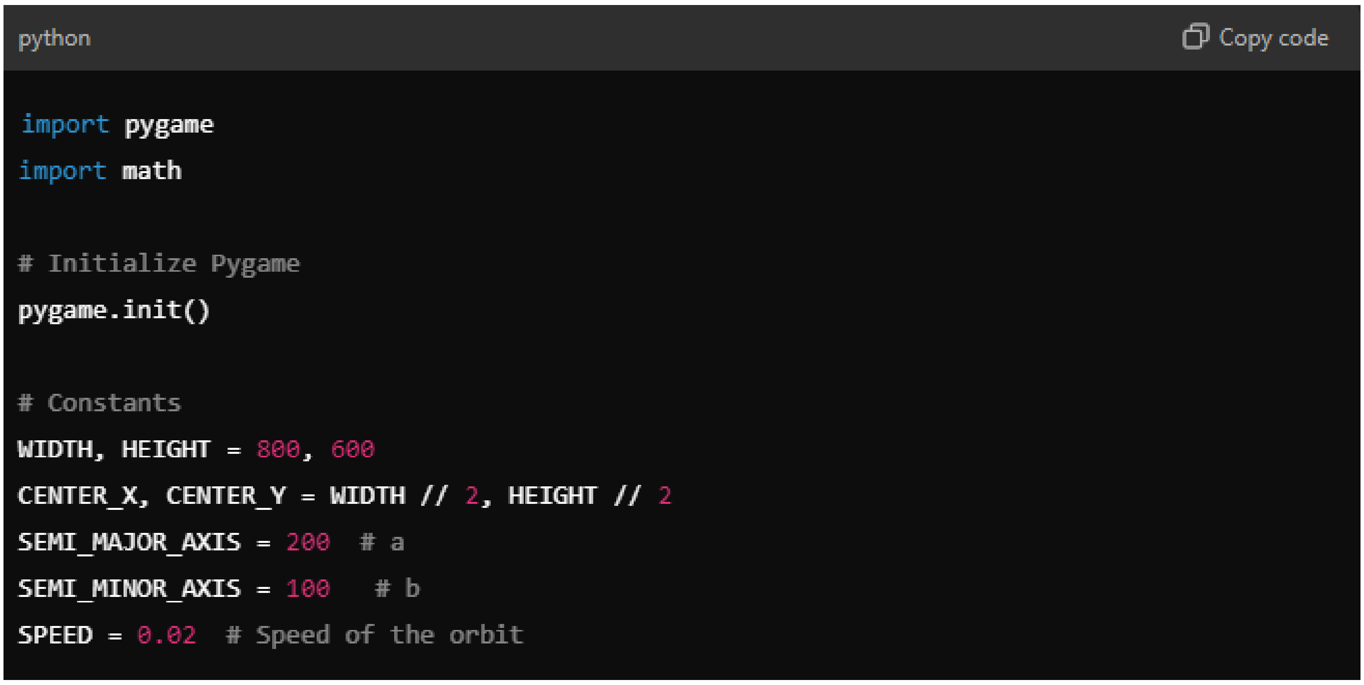
1. **Step 1:** Install Pygame.
2. **Step 2:** Set up the Pygame environment.
3. **Step 3:** Define the parameters of the ellipse (semi-major and semi-minor axes).

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

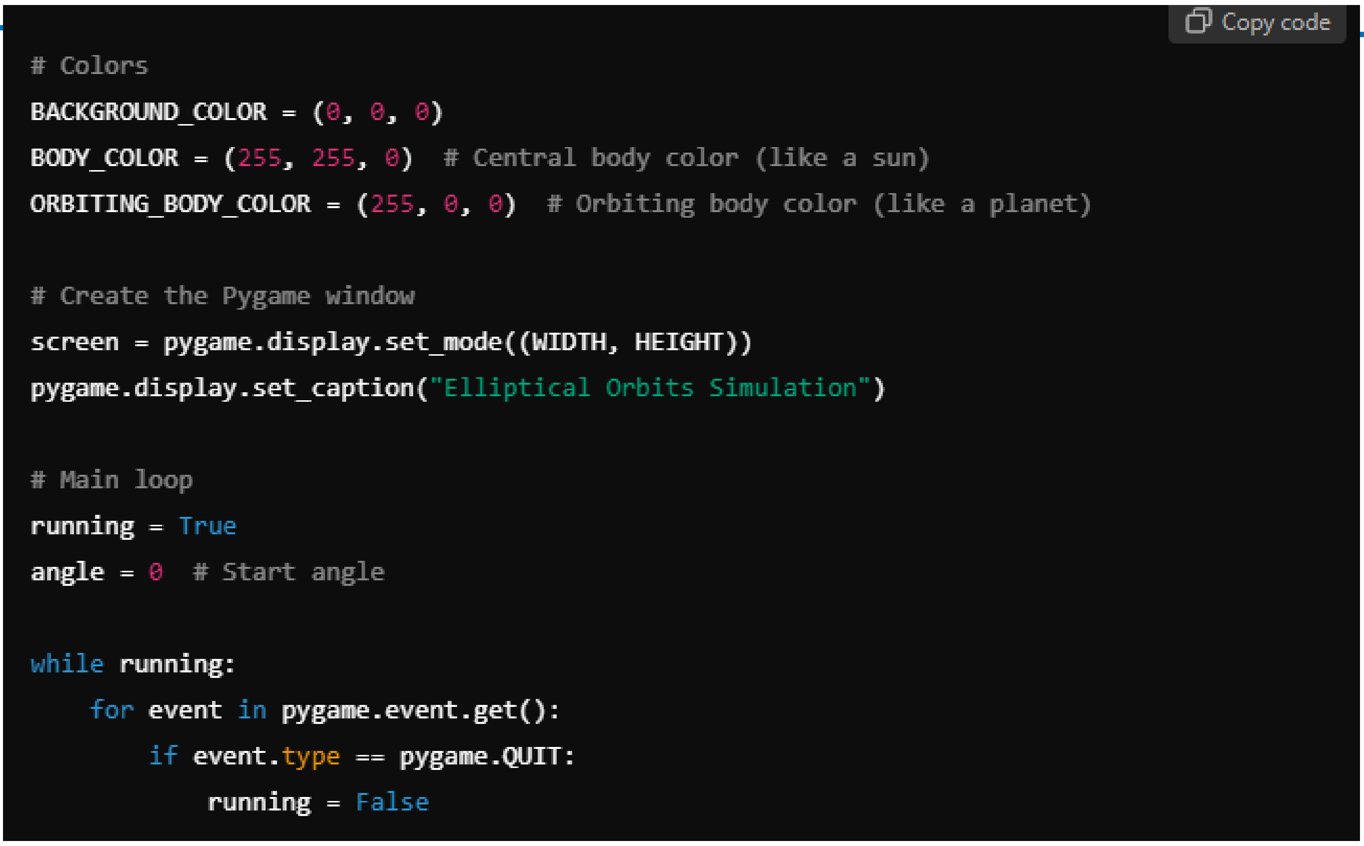
Step 4: Calculate the position of the orbiting object using the parametric equations.

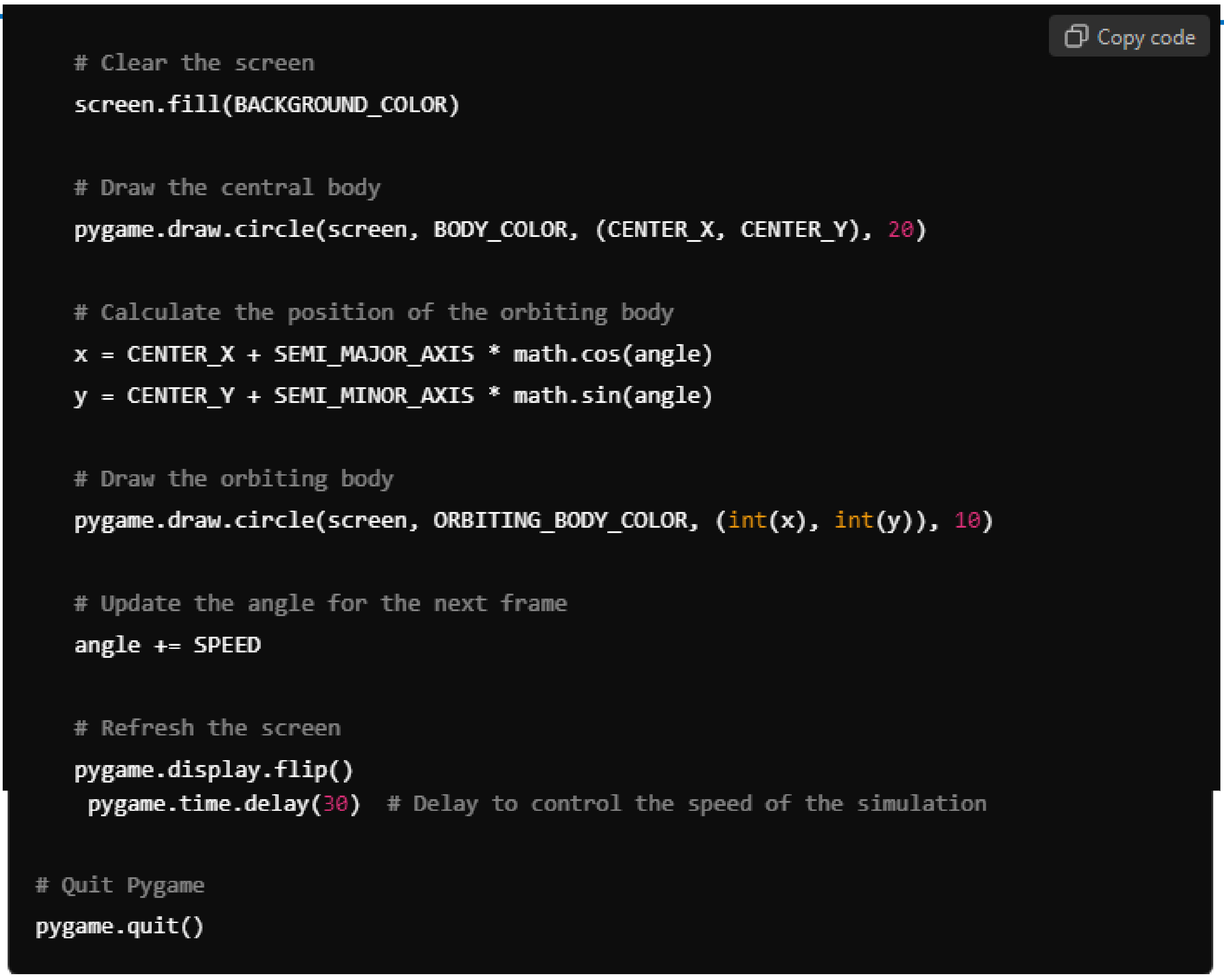
4. **Step 5:** Draw the ellipse and the orbiting object.
5. **Step 6:** Implement a game loop to update the position continuously.
6. **Step 7:** Run the simulation.

Code:



```python
import pygame
import math

# Initialize Pygame
pygame.init()

# Constants
WIDTH, HEIGHT = 800, 600
CENTER_X, CENTER_Y = WIDTH // 2, HEIGHT // 2
SEMI_MAJOR_AXIS = 200  # a
SEMI_MINOR_AXIS = 100   # b
SPEED = 0.02  # Speed of the orbit
```


```
# Colors
BACKGROUND_COLOR = (0, 0, 0)
BODY_COLOR = (255, 255, 0)  # Central body color (like a sun)
ORBITING_BODY_COLOR = (255, 0, 0)  # Orbiting body color (like a planet)

# Create the Pygame window
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Elliptical Orbits Simulation")

# Main loop
running = True
angle = 0  # Start angle

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
```



Copy code

```
    # Clear the screen
    screen.fill(BACKGROUND_COLOR)

    # Draw the central body
    pygame.draw.circle(screen, BODY_COLOR, (CENTER_X, CENTER_Y), 20)

    # Calculate the position of the orbiting body
    x = CENTER_X + SEMI_MAJOR_AXIS * math.cos(angle)
    y = CENTER_Y + SEMI_MINOR_AXIS * math.sin(angle)

    # Draw the orbiting body
    pygame.draw.circle(screen, ORBITING_BODY_COLOR, (int(x), int(y)), 10)

    # Update the angle for the next frame
    angle += SPEED

    # Refresh the screen
    pygame.display.flip()
    pygame.time.delay(30)  # Delay to control the speed of the simulation

# Quit Pygame
pygame.quit()
```

Procedure Explanation:

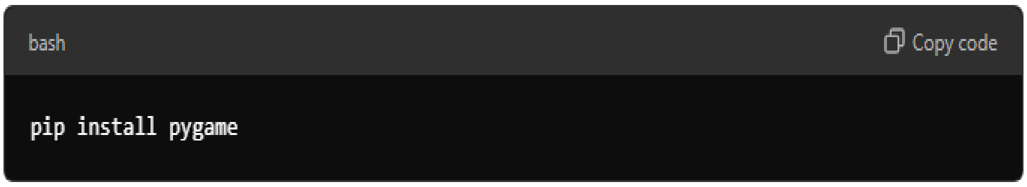
1. **Step 1:** Import the Pygame library and initialize it.
2. **Step 2:** Set up the display dimensions and create the Pygame window.
3. **Step 3:** Define the semi-major and semi-minor axes of the ellipse and set the center of the ellipse.
4. **Step 4:** Initialize a variable t to represent time and set a speed for the orbiting object.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

5. **Step 5:** Enter the main loop where the program runs until the user quits.
6. **Step 6:** Calculate the current position of the orbiting object using the parametric equations.
7. **Step 7:** Clear the screen for redrawing.
8. **Step 8:** Draw the ellipse representing the orbit.
9. **Step 9:** Draw the orbiting object as a circle.
10. **Step 10:** Update the display to show the changes.
11. **Step 11:** Increment the time variable to update the position for the next frame.
12. **Step 12:** Control the frame rate to ensure smooth animation.
13. **Step 13:** Quit Pygame cleanly when the loop ends.

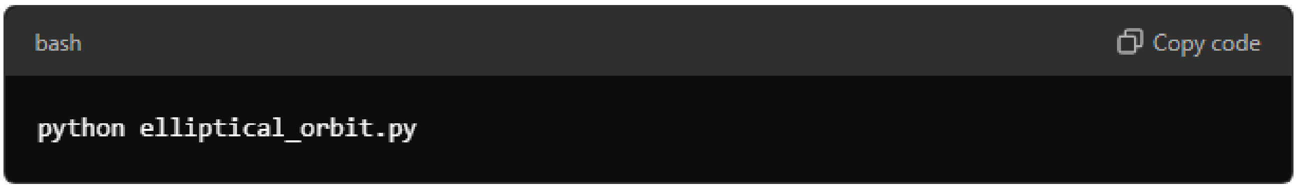
How to Run the Program:

1. Ensure you have Python and Pygame installed. You can install Pygame using pip:



```bash
pip install pygame
```

2. Save the code in a file named elliptical_orbit.py.
3. Open your command line or terminal.
4. Navigate to the directory where the script is saved.
5. Run the script:



```bash
python elliptical_orbit.py
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

This simulation demonstrates the behavior of elliptical orbits using Pygame. It illustrates the use of parametric equations and provides a visual representation of celestial mechanics principles. You can modify the parameters (such as the axes) to see how they affect the orbit shape and behavior.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

13.Python Lab Exercise: Simulating a Bouncing Ball in Pygame

Objective:

To create a simulation of a bouncing ball using Pygame, demonstrating basic physics principles such as gravity and collision detection.

Theory:

In this simulation, a ball will fall due to gravity and bounce off the edges of the screen. When the ball collides with the bottom of the window, it will reverse its vertical velocity, simulating a bounce. We will apply simple physics equations to update the position of the ball.

- **Gravity:** This will be a constant acceleration that affects the ball's vertical velocity.
- **Collision:** When the ball's position hits the bottom of the screen, we reverse its vertical velocity.

Procedure:

1. **Step 1:** Install Pygame.
2. **Step 2:** Set up the Pygame environment.
3. **Step 3:** Define the ball properties (position, velocity, and radius).
4. **Step 4:** Implement the game loop to update the ball's position based on gravity and handle collisions.
5. **Step 5:** Draw the ball on the screen.
6. **Step 6:** Run the simulation.

SriChandrasekharendraSaraswathiViswaMahavidyalaya

[SCSVMV**]**

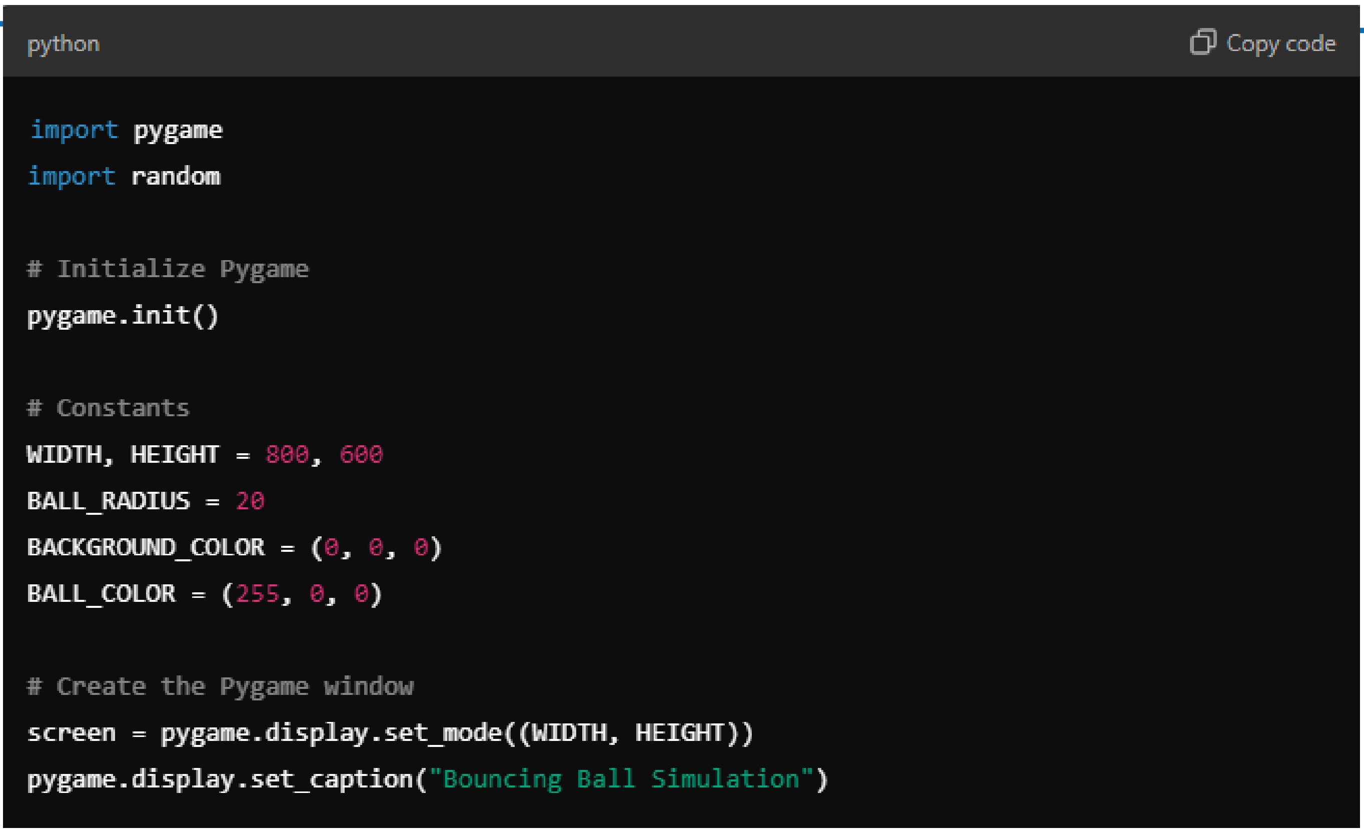
[Deemed to be University U/S 3 of the UGC Act 1956] Accredited with 'A' Grade by NAAC

Enathur,Kanchipuram–631561

Department of Computer Science and Engineering

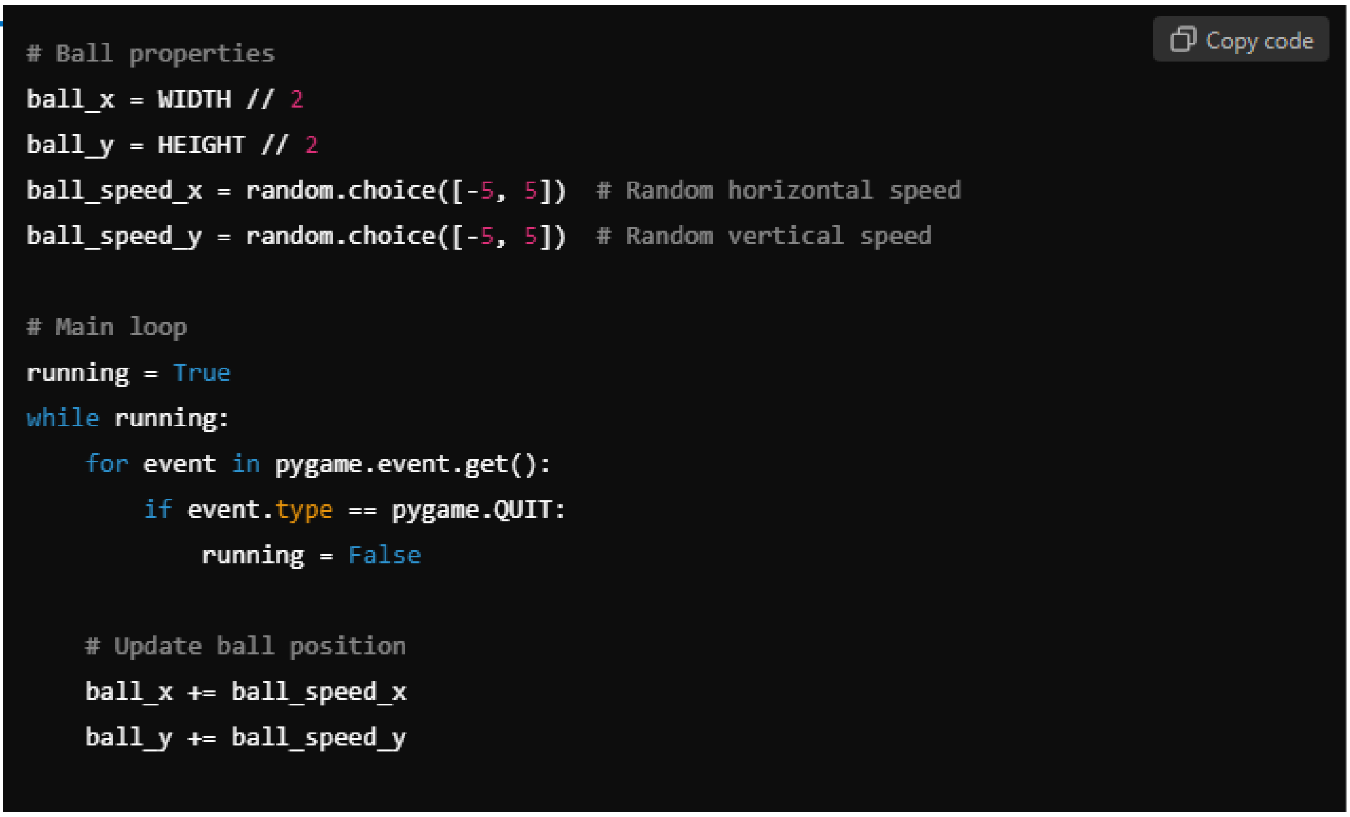
Code:

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

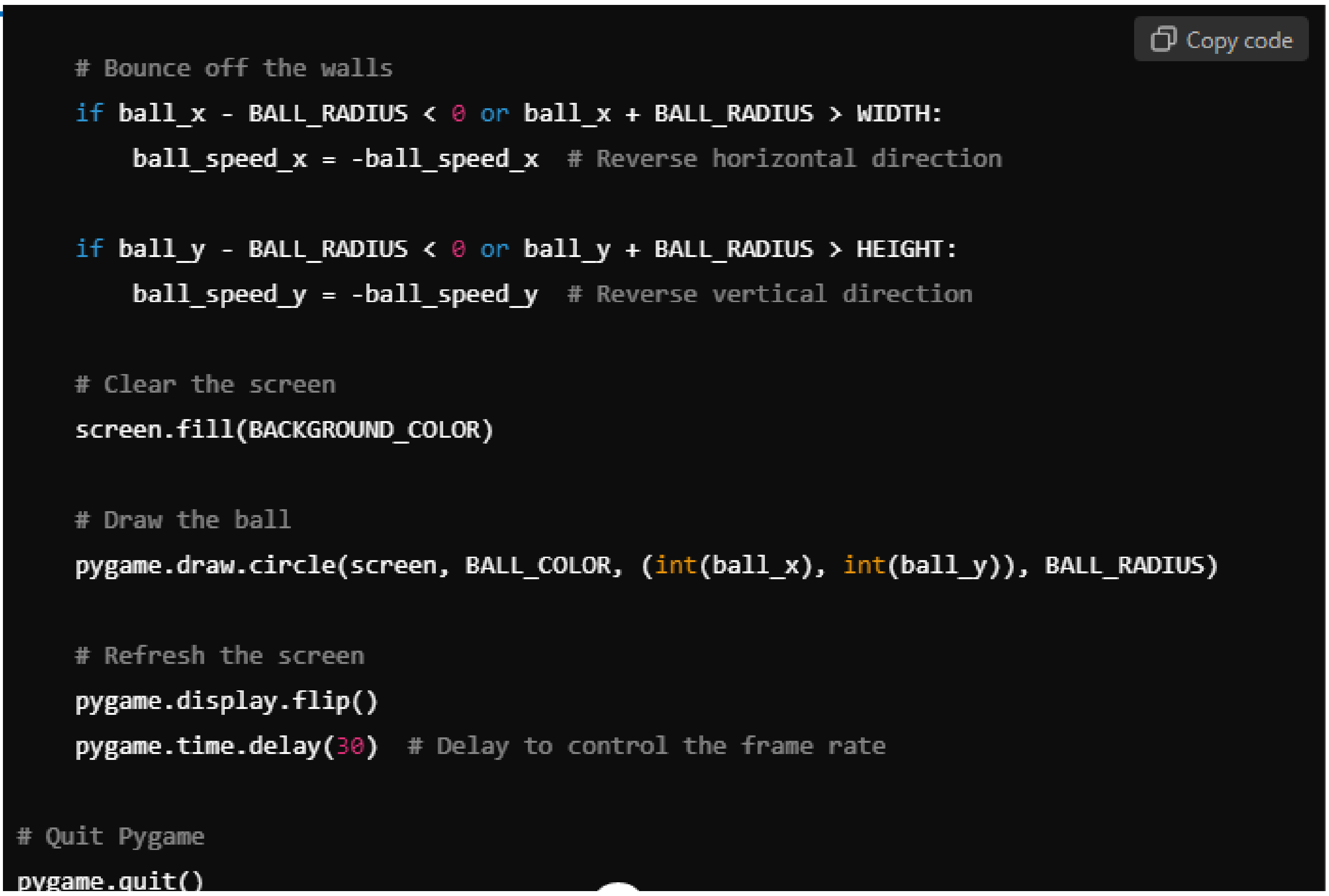

python Copy code

```python
import pygame
import random

# Initialize Pygame
pygame.init()

# Constants
WIDTH, HEIGHT = 800, 600
BALL_RADIUS = 20
BACKGROUND_COLOR = (0, 0, 0)
BALL_COLOR = (255, 0, 0)

# Create the Pygame window
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Bouncing Ball Simulation")
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu


Copy code

```
# Ball properties
ball_x = WIDTH // 2
ball_y = HEIGHT // 2
ball_speed_x = random.choice([-5, 5])  # Random horizontal speed
ball_speed_y = random.choice([-5, 5])  # Random vertical speed

# Main loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update ball position
    ball_x += ball_speed_x
    ball_y += ball_speed_y
```


```
    # Bounce off the walls
    if ball_x - BALL_RADIUS < 0 or ball_x + BALL_RADIUS > WIDTH:
        ball_speed_x = -ball_speed_x  # Reverse horizontal direction

    if ball_y - BALL_RADIUS < 0 or ball_y + BALL_RADIUS > HEIGHT:
        ball_speed_y = -ball_speed_y  # Reverse vertical direction

    # Clear the screen
    screen.fill(BACKGROUND_COLOR)

    # Draw the ball
    pygame.draw.circle(screen, BALL_COLOR, (int(ball_x), int(ball_y)), BALL_RADIUS)

    # Refresh the screen
    pygame.display.flip()
    pygame.time.delay(30)  # Delay to control the frame rate

# Quit Pygame
pygame.quit()
```

Procedure Explanation:

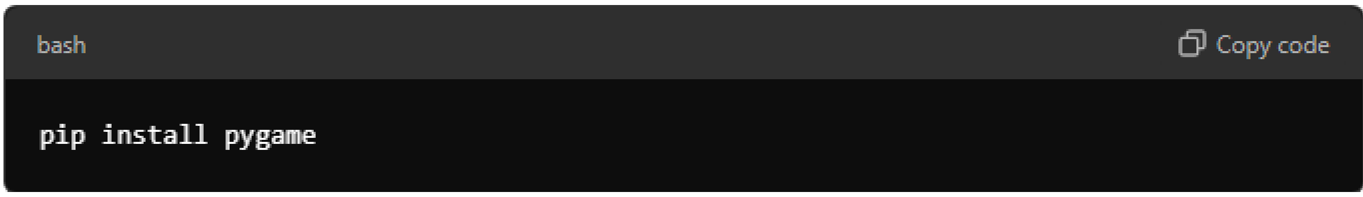
1. **Step 1:** Import the Pygame library and initialize it.
2. **Step 2:** Set the dimensions of the display and create a Pygame window.
3. **Step 3:** Define the ball's properties, including its radius, color, initial position, initial vertical velocity, and gravity constant.
4. **Step 4:** Enter the main loop where the program runs until the user quits.
5. **Step 5:** Update the ball's vertical velocity by adding gravity and then update its position accordingly.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

6. **Step 6:** Check for a collision with the bottom edge of the screen. If the ball hits the bottom, adjust its position and reverse its velocity to simulate a bounce, applying a damping factor to simulate energy loss.
7. **Step 7:** Clear the screen for redrawing.
8. **Step 8:** Draw the ball on the screen at its updated position.
9. **Step 9:** Update the display to show the changes.
10. **Step 10:** Control the frame rate for smooth animation.
11. **Step 11:** Quit Pygame cleanly when the loop ends.

How to Run the Program:

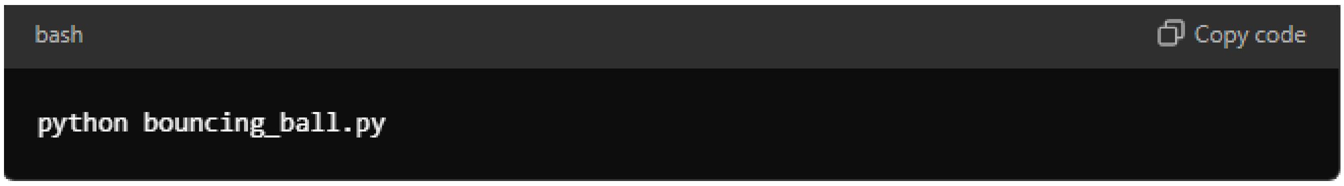
1. Ensure you have Python and Pygame installed. You can install Pygame using



```bash
pip install pygame
```

Save the code in a file named bouncing_ball.py.

2. Open your command line or terminal.
3. Navigate to the directory where the script is saved.
4. Run the script:



```bash
python bouncing_ball.py
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

This simulation effectively demonstrates the principles of motion and collision detection using Pygame. It showcases basic physics concepts like gravity and energy loss during bounces. You can experiment with different gravity values, damping factors, and ball properties to see how they affect the simulation.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

SriChandrasekharendraSaraswathiViswaMahavidyalaya

[SCSVMV**]**

[Deemed to be University U/S 3 of the UGC Act 1956] Accredited with
‘A’ Grade by NAAC

Enathur,Kanchipuram–631561

Department of Computer Science and Engineering

ADDITIONAL EXERCISES

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Aim:

To write and execute a Python program that prints "Hello, World!" on the screen.

Theory:

The **"Hello, World!"** program is traditionally the first program written when learning a new programming language. It is a simple program that helps in understanding how output is handled in the language. In Python, we use the built-in `print()` function to display text on the screen.

- The `print()` function is used to output data to the standard output device (usually the console).
- The string that we want to display is passed to the `print()` function enclosed in either single quotes (`'`) or double quotes (`"`).

Algorithm:

1. Start.
2. Open a Python environment (IDE or terminal).
3. Write the Python code using the `print()` function to display "Hello, World!".
4. Save the program with a `.py` extension.
5. Run the Python program.
6. The message "Hello, World!" will be displayed on the screen.
7. End.

Python Code:

```
# Python program to print "Hello, World!"
print("Hello, World!")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or an online Python interpreter.
2. **Write the Code**:
   - In the editor, type the code `print("Hello, World!")`.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `hello_world.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where your file is saved and run the following command:

```
 hello_world.py
```

   - If using an IDE, use the run command provided by the IDE.
5. **View the Output**:
   - The console should display the message:

```
Hello, World!
```

Output:

When the Python program is executed, the output displayed on the screen will be:

```
Hello, World!
```

Conclusion:

The program successfully prints "Hello, World!" on the screen using the `print()` function in Python. This basic exercise introduces the concept of output in Python and helps to familiarize oneself with the structure of a Python program.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Printing Address in Python:

Aim:

To write and execute a Python program that prints an address on the screen.

Theory:

In Python, we use the `print()` function to output text to the screen. The function can handle multiple lines of text using either multiple `print()` statements or by including newline characters (`\n`) within a single `print()` statement.

The address can be printed in a formatted way by using string literals that include details like the name, street, city, state, and postal code.

Algorithm:

1. Start.
2. Open a Python environment (IDE or terminal).
3. Write the Python code using the `print()` function to display the address.
4. Save the program with a `.py` extension.
5. Run the Python program.
6. The address will be displayed on the screen.
7. End.

Python Code:

```
# Python program to print an address

# Using multiple print statements to print each line of the address
print("John Doe")
print("1234 Elm Street")
print("Cityville, ST 56789")
print("USA")
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Or, you can use a single `print()` statement with newline characters:

```
# Python program to print an address in a single statement
print("John Doe\n1234 Elm Street\nCityville, ST 56789\nUSA")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or an online Python interpreter.
2. **Write the Code**:
   - In the editor, type the code using the `print()` function to display the address.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `print_address.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where your file is saved and run the following command:

     ```
     python print_address.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **View the Output**:
   - The console should display the address in a formatted manner.

Output:

When the Python program is executed, the output displayed on the screen will be:

```
yaml
Copy code
John Doe
1234 Elm Street
Cityville, ST 56789
USA
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Conclusion:

The program successfully prints the address on the screen using the `print()` function in Python. This exercise demonstrates how to output formatted text using multiple lines, an essential concept in Python programming.

Arithmetic Operations in Python:

Aim:

To write and execute a Python program to perform basic arithmetic operations: addition, subtraction, multiplication, division, modulus, and exponentiation.

Theory:

Arithmetic operations are fundamental to all programming languages. In Python, the following arithmetic operators are used to perform different operations:

1. **Addition (`+`)**: Adds two numbers.
2. **Subtraction (`-`)**: Subtracts the second number from the first.
3. **Multiplication (`*`)**: Multiplies two numbers.
4. **Division (`/`)**: Divides the first number by the second and returns a float result.
5. **Modulus (`%`)**: Returns the remainder when the first number is divided by the second.
6. **Exponentiation (`**`)**: Raises the first number to the power of the second.
7. **Floor Division (`//`)**: Divides the first number by the second and returns an integer result by removing the fractional part.

Algorithm:

1. Start.
2. Define two numbers (e.g., `num1` and `num2`).
3. Perform and display the results of the following operations:
   - Addition
   - Subtraction
   - Multiplication
   - Division

   - Modulus
   - Exponentiation
4. Display the results of each operation using the `print()` function.
5. End.

Python Code:

```
# Python program to perform arithmetic operations

# Define two numbers
num1 = 15
num2 = 4

# Perform arithmetic operations
addition = num1 + num2
subtraction = num1 - num2
multiplication = num1 * num2
division = num1 / num2
modulus = num1 % num2
exponentiation = num1 ** num2
floor_division = num1 // num2

# Display the results
print("Addition of", num1, "and", num2, "is:", addition)
print("Subtraction of", num1, "and", num2, "is:", subtraction)
print("Multiplication of", num1, "and", num2, "is:", multiplication)
print("Division of", num1, "by", num2, "is:", division)
print("Modulus of", num1, "and", num2, "is:", modulus)
print("Exponentiation of", num1, "to the power of", num2, "is:", exponentiation)
print("Floor division of", num1, "by", num2, "is:", floor_division)
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or an online Python interpreter.
2. **Write the Code**:
   - In the editor, type the Python code that performs basic arithmetic operations on two numbers.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `arithmetic_operations.py`.
4. **Run the Program**:

- If using a terminal, navigate to the directory where your file is saved and run the following command:

  ```
  python arithmetic_operations.py
  ```

- If using an IDE, use the run command provided by the IDE.

5. **View the Output**:
   - The console will display the results of the arithmetic operations performed.

Conversion from Pounds to Kilograms in Python

Aim:

To write and execute a Python program that converts weight in pounds to kilograms.

Theory:

- Weight can be measured in different units such as pounds and kilograms.
- **Pounds (lbs)** and **Kilograms (kg)** are common units used to measure weight.
- The conversion formula from pounds to kilograms is: Kilograms=Pounds×0.453592\text{Kilograms} = \text{Pounds} \times 0.453592Kilograms=Pounds×0.453592 This conversion factor is derived from the fact that 1 pound is equal to approximately 0.453592 kilograms.

Algorithm:

1. Start.
2. Get the input from the user for weight in pounds.
3. Multiply the value in pounds by the conversion factor `0.453592` to convert it into kilograms.
4. Print the result in kilograms.
5. End.

Python Code:

```
# Python program to convert pounds to kilograms

# Step 1: Take input from the user for weight in pounds
pounds = float(input("Enter the weight in pounds: "))
```

```
# Step 2: Conversion factor
conversion_factor = 0.453592

# Step 3: Convert pounds to kilograms
kilograms = pounds * conversion_factor

# Step 4: Display the result
print(f"{pounds} pounds is equal to {kilograms:.2f} kilograms")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to convert pounds to kilograms using the formula mentioned in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `pounds_to_kg.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python pounds_to_kg.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter a value for the weight in pounds when prompted.
6. **View the Output**:
   - The converted weight in kilograms will be displayed on the screen.

Sample Input/Output:

Input:

```
mathematica
Enter the weight in pounds: 150
```

Output:

```
vbnet
```

```
150 pounds is equal to 68.04 kilograms
```

Conclusion:

The program successfully converts a given weight in pounds to its equivalent weight in kilograms using the `print()` function in Python. This exercise demonstrates how to perform arithmetic operations and output formatted results using Python.

Area of a Circle in Python:

Aim:

To write and execute a Python program to calculate the area of a circle when the radius is given.

Theory:

- The area of a circle is the amount of space enclosed by the circle. It is calculated using the following formula:

  Area=π×r2\text{Area} = \pi \times r^2Area=π×r2

  Where:

  - π\piπ is a mathematical constant approximately equal to 3.14159.
  - rrr is the radius of the circle.
- Python provides the `math` library, which includes a constant `math.pi` that represents the value of π\piπ.

Algorithm:

1. Start.
2. Input the radius of the circle from the user.
3. Use the formula Area=π×r2\text{Area} = \pi \times r^2Area=π×r2 to calculate the area.

4. Display the calculated area.
5. End.

Python Code:

```
# Python program to calculate the area of a circle

# Step 1: Import the math module to use math.pi
import math

# Step 2: Take input from the user for the radius
radius = float(input("Enter the radius of the circle: "))

# Step 3: Calculate the area using the formula
area = math.pi * radius ** 2

# Step 4: Display the area
print(f"The area of the circle with radius {radius} is {area:.2f} square units.")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to calculate the area of a circle using the formula π×r2\pi \times r^2π×r2.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `area_of_circle.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python area_of_circle.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter a value for the radius when prompted.
6. **View the Output**:
   - The area of the circle will be displayed on the screen.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Sample Input/Output:

Input:

```
arduino
Copy code
Enter the radius of the circle: 5
```

Output:

```
csharp
Copy code
The area of the circle with radius 5.0 is 78.54 square units.
```

Conclusion:

The program successfully calculates the area of a circle when the radius is given by the user. This exercise demonstrates how to use mathematical operations and constants in Python, as well as how to format the output using the `print()` function.

Calculating the Area of a Triangle

Objective

To understand and implement a Python program that calculates the area of a triangle using the formula:

Area=12×base×height\text{Area} = \frac{1}{2} \times \text{base} \times \text{height}Area=21×base×height

Materials

- Computer with Python installed
- Integrated Development Environment (IDE) or text editor (e.g., PyCharm, VS Code, Jupyter Notebook)
- Basic understanding of Python syntax

Procedure

1. **Introduction to the Formula**: Explain the formula for calculating the area of a triangle and the meanings of the base and height.
2. **Write the Python Program**: a. Open your IDE or text editor. b. Create a new Python file (e.g., `triangle_area.py`). c. Write the following code:

```
Python CODE
# Program to calculate the area of a triangle

# Function to calculate area
def calculate_area(base, height):
    return 0.5 * base * height
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

```
# Input: Get base and height from user
try:
    base = float(input("Enter the base of the triangle: "))
    height = float(input("Enter the height of the triangle: "))

    # Output: Calculate and display the area
    area = calculate_area(base, height)
    print(f"The area of the triangle with base {base} and height {height} is:
{area}")

except ValueError:
    print("Please enter valid numeric values for base and height.")
```

3. **Run the Program**: a. Save the file. b. Execute the program in your terminal or IDE. c. Input values for base and height when prompted.

4. **Record the Output**: a. Note down the area calculated by the program for different sets of base and height values.

Results

- **Sample Inputs and Outputs**:
  - Input: Base = 5, Height = 10 → Output: Area = 25.0
  - Input: Base = 3, Height = 6 → Output: Area = 9.0
  - Input: Base = 7.5, Height = 4 → Output: Area = 15.0

Conclusion

- Discuss the results and any patterns observed.
- Reflect on the importance of the area calculation in geometry and its applications in real-world scenarios.
- Consider potential improvements or additional features for the program, such as handling different input formats or calculating the area using different methods (e.g., Heron's formula).

Swapping Two Numbers

Objective

To understand and implement a Python program that swaps the values of two variables without using a third variable.

Materials

- Computer with Python installed
- Integrated Development Environment (IDE) or text editor (e.g., PyCharm, VS Code, Jupyter Notebook)
- Basic understanding of Python syntax

Procedure

1. **Introduction to the Concept**: Explain the concept of swapping variables and the various methods to do so, including the method that does not use a third variable.
2. **Write the Python Program**: a. Open your IDE or text editor. b. Create a new Python file (e.g., `swap_numbers.py`). c. Write the following code:

```
python
# Program to swap two numbers

# Input: Get two numbers from user
try:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    # Display numbers before swapping
    print(f"Before swapping: num1 = {num1}, num2 = {num2}")

    # Swap the numbers
    num1, num2 = num2, num1

    # Output: Display numbers after swapping
    print(f"After swapping: num1 = {num1}, num2 = {num2}")

except ValueError:
    print("Please enter valid numeric values.")
```

3. **Run the Program**: a. Save the file. b. Execute the program in your terminal or IDE. c. Input values for the two numbers when prompted.
4. **Record the Output**: a. Note down the values of the numbers before and after swapping for different sets of inputs.

Results

- **Sample Inputs and Outputs**:
  - Input: First Number = 5, Second Number = 10
    - Output: Before swapping: num1 = 5.0, num2 = 10.0
    - Output: After swapping: num1 = 10.0, num2 = 5.0
  - Input: First Number = 3.5, Second Number = 7.2

- Output: Before swapping: num1 = 3.5, num2 = 7.2
- Output: After swapping: num1 = 7.2, num2 = 3.5

Conclusion

- Discuss the results and the success of the swapping operation.
- Reflect on the importance of variable manipulation in programming.
- Consider potential enhancements to the program, such as adding functionality to swap multiple numbers or using different methods (e.g., arithmetic operations) for swapping.

Displaying the Type of a Variable in Python:

Aim:

To write and execute a Python program that displays the type of a variable.

Theory:

In Python, every value has a type, which indicates what kind of data it is. The `type()` function in Python is used to determine the type of a variable or value. Common data types in Python include:

- `int`: Integer numbers
- `float`: Floating-point numbers (decimal numbers)
- `str`: Strings (text)
- `bool`: Boolean values (`True` or `False`)
- `list`: A collection of items
- `tuple`: An immutable collection of items
- `dict`: A dictionary (key-value pairs)

Using the `type()` function, you can easily find out the data type of any variable.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Algorithm:

1. Start.
2. Define variables of different types (e.g., integer, float, string, etc.).
3. Use the `type()` function to get the type of each variable.
4. Print the variable along with its type.
5. End.

Python Code:

```
# Python program to display the type of various variables

# Step 1: Define variables of different types

integer_variable = 10
float_variable = 10.5
string_variable = "Hello, World!"
boolean_variable = True
list_variable = [1, 2, 3]
tuple_variable = (1, 2, 3)
dict_variable = {"key": "value"}

# Step 2: Display the type of each variable
print(f"The type of integer_variable ({integer_variable}) is:
{type(integer_variable)}")
print(f"The type of float_variable ({float_variable}) is: {type(float_variable)}")
print(f"The type of string_variable ({string_variable}) is:
{type(string_variable)}")
print(f"The type of boolean_variable ({boolean_variable}) is:
{type(boolean_variable)}")
print(f"The type of list_variable ({list_variable}) is: {type(list_variable)}")
print(f"The type of tuple_variable ({tuple_variable}) is: {type(tuple_variable)}")
print(f"The type of dict_variable ({dict_variable}) is: {type(dict_variable)}")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to define variables and display their types using the `type()` function.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `variable_type.py`.
4. **Run the Program**:

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

- If using a terminal, navigate to the directory where the file is saved and run the following command:

  ```
  Copy code
  python variable_type.py
  ```

- If using an IDE, use the run command provided by the IDE.

5. **View the Output**:
   - The program will display the type of each variable in the console.

Sample Output:

When the program is executed, the output will display the types of the defined variables:

```
The type of integer_variable (10) is: <class 'int'>
The type of float_variable (10.5) is: <class 'float'>
The type of string_variable (Hello, World!) is: <class 'str'>
The type of boolean_variable (True) is: <class 'bool'>
The type of list_variable ([1, 2, 3]) is: <class 'list'>
The type of tuple_variable ((1, 2, 3)) is: <class 'tuple'>
The type of dict_variable ({'key': 'value'}) is: <class 'dict'>
```

Conclusion:

The program successfully displays the type of various variables in Python. This exercise helps in understanding Python's dynamic typing and provides insight into the different data types available in the language.

Converting Celsius to Fahrenheit in Python: Lab Manual Format

---

Aim:

To write and execute a Python program that converts temperature from Celsius to Fahrenheit.

Theory:

The Celsius and Fahrenheit are two commonly used scales for measuring temperature. The conversion formula from Celsius (°C) to Fahrenheit (°F) is:

°F=(°C×95)+32°F = (°C \times \frac{9}{5}) + 32°F=(°C×59)+32

Where:

- °F°F°F is the temperature in Fahrenheit.
- °C°C°C is the temperature in Celsius.

Using this formula, you can easily convert a temperature given in Celsius to its equivalent in Fahrenheit.

Algorithm:

1. Start.
2. Input the temperature in Celsius from the user.
3. Use the conversion formula to calculate the temperature in Fahrenheit.
4. Display the calculated temperature in Fahrenheit.
5. End.

Python Code:

```
python
Copy code
# Python program to convert Celsius to Fahrenheit

# Step 1: Take input from the user for the temperature in Celsius
celsius = float(input("Enter the temperature in Celsius: "))

# Step 2: Conversion formula
fahrenheit = (celsius * 9/5) + 32

# Step 3: Display the result
print(f"{celsius}°C is equal to {fahrenheit:.2f}°F")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to convert Celsius to Fahrenheit using the formula mentioned in the theory section.
3. **Save the Program**:

- Save the file with a `.py` extension, for example, `celsius_to_fahrenheit.py`.

4. **Run the Program**:
    - If using a terminal, navigate to the directory where the file is saved and run the following command:

      ```
      Copy code
      python celsius_to_fahrenheit.py
      ```

    - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
    - Enter a value for the temperature in Celsius when prompted.
6. **View the Output**:
    - The converted temperature in Fahrenheit will be displayed on the screen.

Sample Input/Output:

Input:

```
mathematica
Enter the temperature in Celsius: 25
```

Output:

```
25.0°C is equal to 77.00°F
```

Conclusion:

The program successfully converts a given temperature in Celsius to Fahrenheit. This exercise demonstrates how to perform arithmetic operations in Python and format the output using the `print()` function.

Aim:

To write and execute a Python program that converts distance from kilometers to miles.

Theory:

The kilometer and mile are both units of distance. The conversion formula from kilometers to miles is:

Miles=Kilometers×0.621371\text{Miles} = \text{Kilometers} \times 0.621371Miles=Kilometers×0.621371

Where:

- Miles\text{Miles}Miles is the distance in miles.
- Kilometers\text{Kilometers}Kilometers is the distance in kilometers.

Using this formula, you can easily convert a distance given in kilometers to its equivalent in miles.

Algorithm:

1. Start.
2. Input the distance in kilometers from the user.
3. Use the conversion formula to calculate the distance in miles.
4. Display the calculated distance in miles.
5. End.

Python Code:

```
python
Copy code
# Python program to convert kilometers to miles

# Step 1: Take input from the user for the distance in kilometers
kilometers = float(input("Enter the distance in kilometers: "))

# Step 2: Conversion formula
miles = kilometers * 0.621371

# Step 3: Display the result
print(f"{kilometers} kilometers is equal to {miles:.2f} miles")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to convert kilometers to miles using the formula mentioned in the theory section.
3. **Save the Program**:

   - Save the file with a `.py` extension, for example, `kilometers_to_miles.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python kilometers_to_miles.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter a value for the distance in kilometers when prompted.
6. **View the Output**:
   - The converted distance in miles will be displayed on the screen.

Sample Input/Output:

Input:
```
mathematica
Enter the distance in kilometers: 10
```

Output:
```
10.0 kilometers is equal to 6.21 miles
```

Conclusion:

The program successfully converts a given distance in kilometers to miles. This exercise demonstrates how to perform arithmetic operations in Python and format the output using the `print()` function.

Checking Odd or Even Number in Python: Lab Manual Format

Aim:

To write and execute a Python program that checks whether a given number is odd or even.

Theory:

In mathematics, an integer is considered:

- **Even** if it is divisible by 2 without a remainder (e.g., -4, -2, 0, 2, 4).

- **Odd** if it is not divisible by 2 (e.g., -3, -1, 1, 3).

The modulo operator (`%`) is used in Python to find the remainder of a division. The expression `number % 2` will be:

- **0** for even numbers.
- **1** for odd numbers.

Algorithm:

1. Start.
2. Input a number from the user.
3. Use the modulo operator to check if the number is even or odd.
4. Display the result.
5. End.

Python Code:

```
# Python program to check if a number is odd or even

# Step 1: Take input from the user
number = int(input("Enter a number: "))

# Step 2: Check if the number is even or odd
if number % 2 == 0:
    result = "even"
else:
    result = "odd"

# Step 3: Display the result
print(f"The number {number} is {result}.")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to check whether a number is odd or even using the logic described in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `odd_or_even.py`.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python odd_or_even.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter an integer when prompted.
6. **View the Output**:
   - The program will display whether the entered number is odd or even.

Sample Input/Output:

Input:

```
css
Copy code
Enter a number: 7
```

Output:

```
csharp
Copy code
The number 7 is odd.
```

Input:

```
css
Copy code
Enter a number: 10
```

Output:

```
csharp
Copy code
The number 10 is even.
```

Conclusion:

The program successfully checks if a given number is odd or even based on user input. This exercise demonstrates the use of conditional statements and the modulo operator in Python.

Checking Leap Year in Python

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Aim:

To write and execute a Python program that determines whether a given year is a leap year or not.

Theory:

A **leap year** is a year that is divisible by 4, but not every year that is divisible by 4 is a leap year. The rules for determining a leap year are as follows:

1. A year is a leap year if it is divisible by 4.
2. However, if the year is divisible by 100, it is **not** a leap year, unless:
3. The year is also divisible by 400, in which case it **is** a leap year.

Using these rules, we can determine if a given year is a leap year.

Algorithm:

```
1. Start.
2. Input the year from the user.
3. Check if the year is divisible by 4.
     o If yes, check if it is divisible by 100.
          ▪ If yes, check if it is divisible by 400.
               ▪ If yes, it is a leap year.
               ▪ If no, it is not a leap year.
          ▪ If no, it is a leap year.
     o If no, it is not a leap year.
4. Display the result.
5. End.
```

Python Code:

```
python
Copy code
# Python program to check if a year is a leap year

# Step 1: Take input from the user
year = int(input("Enter a year: "))

# Step 2: Check for leap year
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

```
    result = "a leap year"
else:
    result = "not a leap year"

# Step 3: Display the result
print(f"The year {year} is {result}.")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to check whether a year is a leap year using the rules described in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `leap_year.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python leap_year.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter a year when prompted.
6. **View the Output**:
   - The program will display whether the entered year is a leap year or not.

Sample Input/Output:

Input:

```
yaml
Copy code
Enter a year: 2020
```

Output:

```
csharp
Copy code
The year 2020 is a leap year.
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Input:

```
yaml
Copy code
Enter a year: 1900
```

Output:

```
csharp
Copy code
The year 1900 is not a leap year.
```

Conclusion:

The program successfully determines whether a given year is a leap year based on user input. This exercise demonstrates the use of conditional statements to implement logical checks in Python.

Checking Positive or Negative Number in Python:

Aim:

To write and execute a Python program that checks whether a given number is positive, negative, or zero.

Theory:

In mathematics:

- A **positive number** is greater than zero (e.g., 1, 2, 3).

- A **negative number** is less than zero (e.g., -1, -2, -3).
- **Zero** is neither positive nor negative.

Using conditional statements, we can check the sign of a number and classify it accordingly.

Algorithm:

1. Start.
2. Input a number from the user.
3. Check if the number is greater than, less than, or equal to zero.

- If greater than zero, classify it as positive.
- If less than zero, classify it as negative.
- If equal to zero, classify it as zero.

4. Display the result.
5. End.

Python Code:

```
python
Copy code
# Python program to check if a number is positive, negative, or zero

# Step 1: Take input from the user
number = float(input("Enter a number: "))

# Step 2: Check if the number is positive, negative, or zero
if number > 0:
    result = "positive"
elif number < 0:
    result = "negative"
else:
    result = "zero"

# Step 3: Display the result
print(f"The number {number} is {result}.")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to check whether a number is positive, negative, or zero.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `positive_negative.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python positive_negative.py
     ```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

- If using an IDE, use the run command provided by the IDE.

5. **Provide Input**:
    - Enter a number when prompted.
6. **View the Output**:
    - The program will display whether the entered number is positive, negative, or zero.

|

Sample Input/Output:

Input:
```
Enter a number: 5
```

Output:
```
The number 5.0 is positive.
```

Input:
```
Enter a number: -3
```

Output:
```
The number -3.0 is negative.
```

Input:
```
Enter a number: 0
```

Output:
```
The number 0.0 is zero.
```

Conclusion:

The program successfully determines whether a given number is positive, negative, or zero based on user input. This exercise demonstrates the use of conditional statements to implement logical checks in Python.

Calculating Grade Based on Marks in Python:

Aim:

To write and execute a Python program that calculates the grade of a student based on their marks.

Theory:

In educational grading systems, grades are typically assigned based on the following scale (this can vary by institution):

- **A**: 90 - 100
- **B**: 80 - 89
- **C**: 70 - 79
- **D**: 60 - 69
- **F**: Below 60

This program will take a student's marks as input and determine the corresponding grade based on these ranges.

Algorithm:

```
1. Start.
2. Input the marks from the user.
3. Check the marks and determine the grade:
     o If marks are 90 or above, assign grade A.
     o If marks are between 80 and 89, assign grade B.
     o If marks are between 70 and 79, assign grade C.
     o If marks are between 60 and 69, assign grade D.
     o If marks are below 60, assign grade F.
4. Display the result.
5. End.
```

Python Code:

```python
# Python program to calculate grade based on marks

# Step 1: Take input from the user for the marks
marks = float(input("Enter your marks: "))

# Step 2: Determine the grade based on the marks
if marks >= 90:
    grade = 'A'
elif marks >= 80:
    grade = 'B'
elif marks >= 70:
    grade = 'C'
```

```
elif marks >= 60:
    grade = 'D'
else:
    grade = 'F'

# Step 3: Display the result
print(f"Your grade is: {grade}")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to calculate the grade based on the marks as described in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `calculate_grade.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python calculate_grade.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter the marks when prompted.
6. **View the Output**:
   - The program will display the calculated grade based on the entered marks.

Sample Input/Output:

Input:
```
Enter your marks: 85
```

Output:
```
Your grade is: B
```

Input:
```
Enter your marks: 72
```

Output:
```
Your grade is: C
```

Input:
```
Enter your marks: 45
```

Output:
```
Your grade is: F
```

Conclusion:

The program successfully calculates the grade based on the marks entered by the user. This exercise demonstrates the use of conditional statements in Python to classify data based on specific criteria.

Exercise 1: Largest of Three Numbers

Problem Statement:

Write a Python program to find the largest of three numbers using if-else statements.

Procedure:

1. Start by importing the necessary libraries (if needed).
2. Define a function that takes three numbers as input.
3. Use the if-else construct to compare the three numbers and find the largest.
4. Return and print the largest number.
5. Test the function with user inputs.

Code:

```
# Step 1: Define the function
deffind_largest(a, b, c):
if (a >= b) and (a >= c):
largest = a
elif (b >= a) and (b >= c):
largest = b
else:
largest = c
return largest

# Step 2: Input from the user
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))

# Step 3: Print the largest number
print("The largest number is:", find_largest(a, b, c))
```

Exercise 2: Check Even or Odd

Problem Statement:

Write a Python program to check whether a number is even or odd using conditional statements.

Procedure:

1. Start by defining a function that takes a number as input.
2. Use the if-else statement to check if the number is divisible by 2.
3. If the number is divisible by 2, return "Even"; otherwise, return "Odd".
4. Test the function by accepting input from the user.

Code:

```
# Step 1: Define the function to check even or odd
```

```
defcheck_even_odd(num):
ifnum % 2 == 0:
return"Even"
else:
return"Odd"

# Step 2: Input from the user
num = int(input("Enter a number: "))

# Step 3: Print the result
print(f"{num} is", check_even_odd(num))
```

Exercise 3: Fibonacci Sequence

Problem Statement:

Write a Python program to print the Fibonacci sequence up to n terms using a while loop.

Procedure:

1. Define a function to print the Fibonacci sequence.
2. Initialize two variables, a andb, to represent the first two numbers of the Fibonacci sequence.
3. Use a while loop to generate the next numbers in the sequence until n terms are reached.
4. Print the numbers.

Code:

```
# Step 1: Define the function to generate Fibonacci sequence
deffibonacci(n):
a, b = 0, 1
count = 0
```

```
while count < n:
print(a, end=" ")
a, b = b, a + b
count += 1

# Step 2: Input from the user
n = int(input("Enter number of terms: "))

# Step 3: Print Fibonacci sequence
print("Fibonacci sequence:")
fibonacci(n)
```

Exercise 4: Prime Number Check

Problem Statement:

Write a Python program to check whether a number is prime or not using a for loop.

Procedure:

1. Define a function that takes a number as input.
2. Use a for loop to check if the number is divisible by any number from 2 to the square root of the input number.
3. If the number is divisible by any number, it's not prime; otherwise, it is prime.
4. Return and print the result.

Code:

```
# Step 1: Define the function to check prime numbers
```

```
defis_prime(num):
ifnum<2:
returnFalse
foriinrange(2, int(num ** 0.5) + 1):
ifnum % i == 0:
returnFalse
returnTrue

# Step 2: Input from the user
num = int(input("Enter a number: "))

# Step 3: Print whether the number is prime
ifis_prime(num):
print(f"{num} is a prime number.")
else:
print(f"{num} is not a prime number.")
```

Exercise 5: Factorial Using Recursion

Problem Statement:

Write a Python program to find the factorial of a number using recursion.

Procedure:

1. Define a recursive function that calculates the factorial.
2. The base case is when n == 1, and the recursive case is n * factorial(n-1).
3. Call the function with user input.
4. Print the factorial of the number.

Code:

```
# Step 1: Define the recursive function to find factorial
deffactorial(n):
```

```
if n == 1:
return1
else:
return n * factorial(n - 1)

# Step 2: Input from the user
num = int(input("Enter a number: "))

# Step 3: Print the factorial
print(f"Factorial of {num} is", factorial(num))
```

Exercise 6: Reverse a String

Problem Statement:

Write a Python function to reverse a string.

Procedure:

1. Define a function that takes a string as input.
2. Use slicing [::-1] to reverse the string.
3. Return and print the reversed string.
4. Test the function with user input.

Code:

```
# Step 1: Define the function to reverse a string
defreverse_string(s):
return s[::-1]

# Step 2: Input from the user
```

```
s = input("Enter a string: ")

# Step 3: Print the reversed string
print("Reversed string:", reverse_string(s))
```

Exercise 7: Sum and Average of a List

Problem Statement:

Write a Python program to find the sum and average of elements in a list.

Procedure:

1. Define a function that takes a list as input.
2. Use the sum() function to find the sum of the list elements.
3. Calculate the average by dividing the sum by the length of the list.

4. Return and print the sum and average.

Code:

```
# Step 1: Define the function to find sum and average
defsum_and_average(lst):
total_sum = sum(lst)
avg = total_sum / len(lst)
returntotal_sum, avg

# Step 2: Input a list of numbers
lst = list(map(int, input("Enter list elements separated by space: ").split()))

# Step 3: Print the sum and average
total, avg = sum_and_average(lst)
print(f"Sum = {total}, Average = {avg}")
```

Exercise 8: Word Count in a String

Problem Statement:

Write a Python program to count the occurrences of each word in a string using a dictionary.

Procedure:

1. Define a function that splits a string into words.
2. Initialize an empty dictionary to store word counts.
3. Use a loop to iterate over the words and update the dictionary with word occurrences.
4. Return and print the dictionary.

Code:

```
# Step 1: Define the function to count word occurrences
defword_count(s):

words = s.split()
count_dict = {}
for word in words:
if word incount_dict:
count_dict[word] += 1
else:
count_dict[word] = 1
returncount_dict

# Step 2: Input a string
s = input("Enter a string: ")

# Step 3: Print the word occurrences
counts = word_count(s)
print("Word occurrences:", counts)
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Calculating BMI in Python:

Aim:

To write and execute a Python program that calculates the Body Mass Index (BMI) based on a person's weight and height.

Theory:

Body Mass Index (BMI) is a measurement that helps to determine whether a person has a healthy body weight for their height. It is calculated using the following formula:

BMI=weight in kg(height in meters)2\text{BMI} = \frac{\text{weight in kg}}{(\text{height in meters})^2}BMI=(height in meters)2weight in kg

BMI Categories:

- **Underweight**: BMI < 18.5
- **Normal weight**: 18.5 ≤ BMI < 24.9
- **Overweight**: 25 ≤ BMI < 29.9
- **Obesity**: BMI ≥ 30

Algorithm:

1. Start.
2. Input the weight (in kilograms) from the user.
3. Input the height (in meters) from the user.
4. Calculate the BMI using the formula.
5. Determine the BMI category based on the calculated value.
6. Display the BMI and its category.
7. End.

Python Code:

```
# Python program to calculate BMI
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

```
# Step 1: Take input from the user for weight and height
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))

# Step 2: Calculate the BMI
bmi = weight / (height ** 2)

# Step 3: Determine the BMI category
if bmi < 18.5:
    category = "Underweight"
elif 18.5 <= bmi < 24.9:
    category = "Normal weight"
elif 25 <= bmi < 29.9:
    category = "Overweight"
else:
    category = "Obesity"

# Step 4: Display the result
print(f"Your BMI is: {bmi:.2f}")
print(f"Category: {category}")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to calculate BMI as described in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `calculate_bmi.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python calculate_bmi.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter the weight in kilograms and height in meters when prompted.
6. **View the Output**:
   - The program will display the calculated BMI and its corresponding category.

Sample Input/Output:

Input:
```
Enter your weight in kilograms: 70
Enter your height in meters: 1.75
```

Output:
```
Your BMI is: 22.86
Category: Normal weight
```

Input:
```
Enter your weight in kilograms: 95
Enter your height in meters: 1.8
```

Output:
```
Your BMI is: 29.32
Category: Overweight
```

Conclusion:

The program successfully calculates the BMI based on user input for weight and height. This exercise demonstrates how to use arithmetic operations and conditional statements in Python to classify data based on specific criteria.

Aim:

To write and execute a Python program that reverses a given number.

Theory:

Reversing a number involves changing the order of its digits. For example, if the input number is 12345, the reversed number will be 54321. This can be achieved by converting the number to a string, reversing it, and then converting it back to an integer (if needed).

Algorithm:

1. Start.

2. Input the number from the user.
3. Convert the number to a string.
4. Reverse the string representation of the number.
5. Convert the reversed string back to an integer (if desired).
6. Display the reversed number.
7. End.

Python Code:

```
python
Copy code
# Python program to reverse a number

# Step 1: Take input from the user
number = input("Enter a number: ")

# Step 2: Reverse the number
reversed_number = number[::-1]

# Step 3: Display the result
print(f"The reversed number is: {reversed_number}")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to reverse a number as described in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `reverse_number.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python reverse_number.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:
   - Enter a number when prompted.
6. **View the Output**:
   - The program will display the reversed number.

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

Sample Input/Output:

Input:

```
css

Enter a number: 12345
```

Output:

```
The reversed number is: 54321
```

Input:

```
Enter a number: 6789
```

Output:

```
The reversed number is: 9876
```

Conclusion:

The program successfully reverses the digits of the number entered by the user. This exercise demonstrates the use of string manipulation in Python to achieve the desired output.

Multiplication Table in Python:

Aim:

To write and execute a Python program that generates the multiplication table for a given number.

Theory:

A multiplication table shows the product of numbers multiplied by a specified number. For example, the multiplication table of 5 is:

```
python
Copy code
1 × 5 = 5
2 × 5 = 10
3 × 5 = 15
...
10 × 5 = 50
```

This program will take an integer input from the user and display the multiplication table for that number from 1 to 10.

Algorithm:

1. Start.
2. Input the number for which the multiplication table is to be generated.
3. Loop through numbers from 1 to 10.
4. For each iteration, calculate the product of the input number and the loop index.
5. Display the result in a formatted manner.
6. End.

Python Code:

```
python
Copy code
# Python program to generate a multiplication table

# Step 1: Take input from the user for the number
number = int(input("Enter a number to generate its multiplication table: "))

# Step 2: Generate and display the multiplication table
print(f"\nMultiplication Table for {number}:\n")
for i in range(1, 11):
    product = number * i
    print(f"{i} × {number} = {product}")
```

Procedure:

1. **Open the Python IDE or Terminal**:
   - Ensure that Python is installed on your system.
   - Open an IDE like PyCharm, VSCode, or a terminal with Python.
2. **Write the Code**:
   - Write the Python code to generate a multiplication table as described in the theory section.
3. **Save the Program**:
   - Save the file with a `.py` extension, for example, `multiplication_table.py`.
4. **Run the Program**:
   - If using a terminal, navigate to the directory where the file is saved and run the following command:

     ```
     Copy code
     python multiplication_table.py
     ```

   - If using an IDE, use the run command provided by the IDE.
5. **Provide Input**:

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

   - Enter the number when prompted.
6. **View the Output**:
   - The program will display the multiplication table for the entered number.

Sample Input/Output:

Input:
```
css
Copy code
Enter a number to generate its multiplication table: 5
```

Output:
```
css
Copy code
Multiplication Table for 5:

1 × 5 = 5
2 × 5 = 10
3 × 5 = 15
4 × 5 = 20
5 × 5 = 25
6 × 5 = 30
7 × 5 = 35
8 × 5 = 40
9 × 5 = 45
10 × 5 = 50
```

Input:
```
css
Copy code
Enter a number to generate its multiplication table: 7
```

Output:
```
css
Copy code
Multiplication Table for 7:

1 × 7 = 7
2 × 7 = 14
3 × 7 = 21
4 × 7 = 28
5 × 7 = 35
6 × 7 = 42
7 × 7 = 49
8 × 7 = 56
9 × 7 = 63
```

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu

```
10 × 7 = 70
```

Conclusion:

The program successfully generates the multiplication table for the specified number. This exercise demonstrates the use of loops and basic arithmetic operations in Python to produce a formatted output.

4o mini

Prepared By
Dr.R.Prema
Assistant Professor
Dept.of CSE
SCSVMV Deemed to be University
Kanchipuram
Tamilnadu