



Department of Computer Science and Engineering

**II Year II Semester
Python Programming Lab
(R23CC21L3)**

Lab Manual

(R23 Regulations)



Narasaraopeta Engineering College

Kotappakonda Road, Yellamanda (Post), Narasaraopet – 522601, Guntur District, AP
Approved by AICTE, New Delhi & Permanently affiliated to JNTUK, Kakinada, Code: 47,
Accredited by NBA & NAAC, RTA Approved Pollution test Centre, ISO 9001: 2008 Certified Institution
Phone: 08647-239905 Website: www.nrtec.in

Institute's Vision, Mission & Values

Vision:

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community.

Mission:

M1: Provide the best class infra-structure to explore the field of engineering and research.

M2: Build a passionate and a determined team of faculty with student centric teaching, imbining experiential, innovative skills.

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems.

Values:

- **Student-centric education:** Meeting the community's and student's needs by developing a world-class educational environment with cultural values.
- **Excellence:** Giving special attention towards the standards of integrity and performance to help the institute in leading academic achievements and professional goals.
- **Collaboration:** Seeking the latest input and working closely with all the industrial sectors and Society for the continuous upgradation of the quality of education.
- **Diversity:** Creating a favourable on-campus environment in which the goals and learning styles of all students are recognised and nurtured.
- **Continuous Development:** Encouraging enthusiastic, innovative thinkers and learners to strive for personal growth in the world of inventions and start-ups.
- **Technological Advancement:** Keeping pace with evolving technology and professional trends to prepare all its students to achieve success in the workplace.



Department of Computer Science and Engineering

Vision:

To become a centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

Mission:

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertise in modern software tools and technologies to cater to the real time requirements of the industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

Program Specific Outcomes:

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

Program Educational Objectives:

The graduates of the Programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

PROGRAM OUTCOMES (POs)

P01:	Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
P02:	Problem Analysis: Identify, formulate, review research literature and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
P03:	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations.
P04:	Conduct investigations of complex problems: Use research based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
P05:	Model tool usage: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
P06:	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
P07:	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
P08:	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
P09:	Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.
P010:	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
P011:	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
P012:	Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broad test context of technological change.

PROGRAM : B.Tech.(CSE)
YEAR/SEM : II/II
COURSE : PYTHON LAB

ACADEMIC YEAR :2024-2025
REGULATION :R23

COURSE OBJECTIVES:

The main objectives of the course are to

- Introduce core programming concept of Python programming language.
- Demonstrate about Python data structures like Lists, Tuples, Sets and dictionaries.
- Implements Functions, Modules and Regular Expressions in Python Programming and to crate practical and contemporary applications using these

COURSE OUTCOMES:

At the end of the course students will be able to

CO 1: Make use of control flow statements and functions to develop python programs. [K3].

CO 2: Develop Python programs using strings, Lists, dictionaries, tuples and sets. [K3].

CO 3: Develop Python programs on object oriented programming and regular expressions. [K3].

CO 4: Develop Python programs using NumPy and Pandas. [K3].

COURSE OUTCOMES MAPPING WITH PROGRAM OUTCOME

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	2	1	2	-	-	-	-	-	-	-	-
CO2	3	2	-	2	-	-	-	-	-	-	-	-
CO3	2	2	-	2	-	-	-	-	-	-	-	-
CO4	2	2	1	2	-	-	-	-	-	-	-	-

LIST OF EXPERIMENTS

Week No.	NAME OF THE EXPERIMENT	PAGE No.
1	1. Write a program to find the largest element among three Numbers. 2. Write a Program to display all prime numbers within an interval 3. Write a program to swap two numbers without using a temporary variable.	
2	4. Demonstrate the following Operators in Python with suitable examples. i) Arithmetic Operators ii) Relational Operators iii) Assignment Operators iv) Logical Operators v) Bit wise Operators vi) Ternary Operator vii) Membership Operators viii) Identity Operators 5. Write a program to add and multiply complex numbers 6. Write a program to print multiplication table of a given number.	
3	1. Write a program to define a function with multiple return values. 2. Write a program to define a function using default arguments. 3. Write a program to find the length of the string without using any library functions.	
4	4. Write a program to check if the substring is present in a given string or not. 5. Write a program to perform the given operations on a list: i. addition ii. Insertion iii. slicing 6. Write a program to perform any 5 built-in functions by taking any list.	
5.	1. Write a program to create tuples (name, age, address, college) for at least two members and concatenate the tuples and print the concatenated tuples. 2. Write a program to count the number of vowels in a string (No control flow allowed). 3. Write a program to check if a given key exists in a dictionary or not.	
6.	4. Write a program to add a new key-value pair to an existing dictionary. 5. Write a program to sum all the items in a given dictionary.	
7.	1. Write a program to sort words in a file and put them in another file. The output file should have only lower-case words, so any upper-case words from source must be lowered. 2. Python program to print each line of a file in reverse order.	

8.	<p>3. Python program to compute the number of characters, words and lines in a file.</p> <p>4. Write a program to create, display, append, insert and reverse the order of the items in the array.</p>	
9.	<p>5. Write a program to add, transpose and multiply two matrices.</p> <p>6. Write a Python program to create a class that represents a shape. Include methods to calculate its area and perimeter. Implement subclasses for different shapes like circle, triangle, and square.</p>	
10.	<p>1. Python program to check whether a JSON string contains complex object or not.</p> <p>2. Python Program to demonstrate NumPy arrays creation using array () function.</p> <p>3. Python program to demonstrate use of ndim, shape, size, dtype.</p>	
11.	<p>4. Python program to demonstrate basic slicing, integer and Boolean indexing.</p> <p>5. Python program to find min, max, sum, cumulative sum of array.</p>	
12.	<p>6. Create a dictionary with at least five keys and each key represent value as a list where this list contains at least ten values and convert this dictionary as a pandas data frame and explore the data through the data frame as follows:</p> <p>a) Apply head () function to the pandas data frame</p> <p>b) Perform various data selection operations on Data Frame</p> <p>7. Select any two columns from the above data frame, and observe the change in one attribute with respect to other attribute with scatter and plot operations in Matplotlib.</p>	

REFERENCE BOOKS:

1. Gowrishankar S, Veena A., Introduction to Python Programming, CRC Press
2. Python Programming, S Sridhar, J Indumathi, V M Hariharan, 2 ndEdition, Pearson, 2024
3. Introduction to Programming Using Python, Y. Daniel Liang, Pearson

ONLINE LEARNING RESOURCES / VIRTUAL LABS:

1. <https://www.coursera.org/learn/python-for-applied-data-science-ai>
2. <https://www.coursera.org/learn/python?specialization=python#syllabus>

Experiment-1

Date: _____

PROGRAM-1

Write a program to find the largest element among three Numbers.

AIM

The aim of this program is to find the largest element among three given numbers.

ALGORITHM

Step-1: Read three numbers from the user.

Step-2: Compare the first number with the second and third numbers.

Step-3: If the first number is greater than or equal to both the second and third numbers, it is the largest.

Step-4: If not, compare the second and third numbers to determine the largest among them.

Step-5: Output the largest number.

PROGRAM

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))
if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >= num3):
    largest = num2
else:
    largest = num3
print("The largest number is", largest)
```


OUTPUT

Python Programming

```
Enter first number: 10
Enter second number: 25
Enter third number: 15
The largest number is: 25
```

VIVA QUESTIONS

1. How do you compare three numbers to find the largest one?
2. What is the time complexity of your program?
3. Can you write the logic for finding the largest number using Python's built-in functions?
4. How would you modify the program to find the largest number among more than three numbers?
5. What are the different ways to take input from the user in python?

Experiment-2

Date: _____

PROGRAM-2

Write a Program to display all prime numbers within an interval.

AIM

To display all prime numbers within a given interval.

ALGORITHM

Step-1: Read the lower and upper bounds of the interval from the user.

Step-2: Iterate through each number within the interval.

Step-3: For each number, check if it is prime.

Step-4: To check if a number is prime, iterate from 2 to the square root of the number.

Step-5: If the number is divisible by any number within this range, it is not prime. Otherwise, it is prime.

Step-6: Output all prime numbers within the interval.

PROGRAM

```
import math

lower = int(input("Enter the lower bound of the interval: "))

upper = int(input("Enter the upper bound of the interval: "))

print("Prime numbers within the interval [{}, {}]:".format(lower, upper))

for num in range(lower, upper + 1):

    if num > 1:

        is_prime = True

        for i in range(2, int(math.sqrt(num)) + 1):
```

```
if num % i == 0:
```

```
    is_prime = False
```

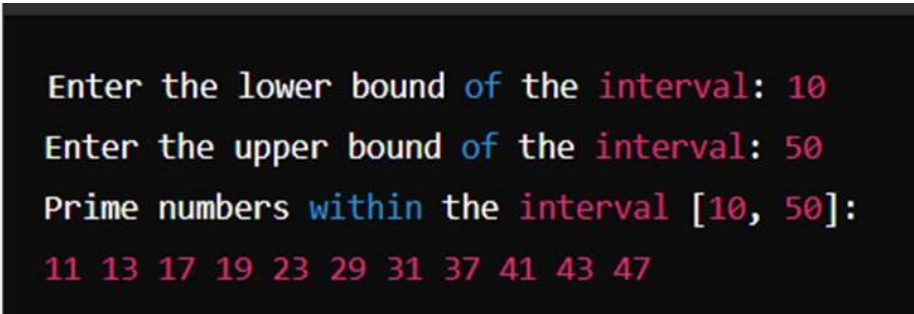
```
    break
```

```
if is_prime:
```

```
    print(num, end=" ")
```

Lab Record

OUTPUT



```
Enter the lower bound of the interval: 10
Enter the upper bound of the interval: 50
Prime numbers within the interval [10, 50]:
11 13 17 19 23 29 31 37 41 43 47
```

VIVA QUESTIONS

1. What is a prime number?
2. How does your program check if a number is prime?
3. What is the time complexity of checking if a single number is prime?
4. Can you optimize your program to improve its efficiency?
5. How do you handle the edge cases in your program (e.g., intervals with negative num

Experiment-3

Date: _____

PROGRAM-3

Write a program to swap two numbers without using a temporary variable.

AIM

The Aim of this Program to swap two numbers without using a temporary variable.

ALGORITHM

Step-1: Read input for two numbers, num1 and num2.

Step-2: Print the original values of num1 and num2.

Step-3: Swap the values of num1 and num2 without using a temporary variable.

Step-4 : Print the swapped values of num1 and num2.

PROGRAM

```
num1 = int(input("Enter the first number: "))  
num2 = int(input("Enter the second number: "))  
print("Before swapping: num1 =", num1, ", num2 =", num2)  
num1 = num1 + num2  
num2 = num1 - num2  
num1 = num1 - num2  
print("After swapping: num1 =", num1, ", num2 =", num2)
```

OUTPUT

```
Enter the first number: 5
Enter the second number: 10
Before swapping: num1 = 5 , num2 = 10
After swapping: num1 = 10 , num2 = 5
```

VIVA QUESTIONS

1. What are the different ways to swap two variables in Python?
2. Why might you prefer not to use a temporary variable?
3. Can you explain the concept of tuple unpacking in Python?
4. What are the potential pitfalls of swapping using arithmetic operations?

Experiment-4

Date: _____

PROGRAM-4

Demonstrate the following Operators in Python with suitable examples .i) Arithmetic Operators
ii) Relational Operators iii) Assignment Operators iv) Logical Operators v) Bit wise Operators
vi) Ternary Operator vii) Membership Operators viii) Identity Operators

i) Arithmetic Operators

AIM

The aim is to perform basic arithmetic operations such as addition, subtraction, multiplication, division, etc., on two numbers.

ALGORITHM

Step-1: Read two numbers from the user.

Step-2: Perform various arithmetic operations on the numbers.

Step-3: Display the results.

PROGRAM

```
num1 = int(input("Enter first number: "))

num2 = int(input("Enter second number: "))

# Performing arithmetic operations

addition = num1 + num2

subtraction = num1 - num2

multiplication = num1 * num2

division = num1 / num2 modulus

= num1 % num2 exponentiation =

num1 ** num2 floor_division =

num1 // num2 # Printing output

print("Addition:", addition)

print("Subtraction:", subtraction)

print("Multiplication:", multiplication)

print("Division:", division)

print("Modulus:", modulus)

print("Exponentiation:", exponentiation)

print("Floor Division:", floor_division)
```

OUTPUT

```
Addition: 10 + 3 = 13
Subtraction: 10 - 3 = 7
Multiplication: 10 * 3 = 30
Division: 10 / 3 = 3.3333333333333335
Modulus: 10 % 3 = 1
Exponentiation: 10 ** 3 = 1000
Floor Division: 10 // 3 = 3
```


ii) Relational Operators

AIM

The aim is to compare two values and determine the relationship between them.

ALGORITHM

Step-1: Read two numbers from the user.

Step-2: Use relational operators to compare the numbers.

Step-3: Display the comparison results.

PROGRAM

```
# Reading input
```

```
num1 = int(input("Enter first number: "))
```

```
num2 = int(input("Enter second number: "))
```

```
# Using relational operators
```

```
print("Equal to:", num1 == num2)
```

```
print("Not equal to:", num1 != num2)
```

```
print("Greater than:", num1 > num2)
```

```
print("Less than:", num1 < num2)
```

```
print("Greater than or equal to:", num1 >= num2)
```

```
print("Less than or equal to:", num1 <= num2)
```

OUTPUT

```
Enter first number: 5
Enter second number: 10
Equal to: False
Not equal to: True
Greater than: False
Less than: True
Greater than or equal to: False
Less than or equal to: True
```

iii) Assignment Operators

AIM

The aim is to assign values to variables with shorthand notation.

ALGORITHM

Step-1: Initialize a variable with a value.

Step-2: Use assignment operators to modify the variable's value.

Step-3: Display the final value of the variable.

PROGRAM

Addition Assignment Operator

x = 5

x += 3 # Equivalent to x = x + 3

print("Addition Assignment Operator:", x) # Output: 8

Subtraction Assignment Operator

y = 10

y -= 2 # Equivalent to y = y - 2

print("Subtraction Assignment Operator:", y) # Output: 8

Multiplication Assignment Operator

z = 4

z *= 5 # Equivalent to z = z * 5

print("Multiplication Assignment Operator:", z) # Output: 20

Division Assignment Operator

w = 16

```
w /= 4 # Equivalent to w = w / 4

print("Division Assignment Operator:", w) # Output: 4.0

# Modulus Assignment Operator

a = 17

a %= 5 # Equivalent to a = a % 5

print("Modulus Assignment Operator:", a) # Output: 2

# Exponentiation Assignment Operator

b = 2

b **= 3 # Equivalent to b = b ** 3

print("Exponentiation Assignment Operator:", b) # Output: 8

# Floor Division Assignment Operator

c = 17

c //= 5 # Equivalent to c = c // 5

print("Floor Division Assignment Operator:", c)
```

OUTPUT

```
Addition Assignment Operator: 8

Subtraction Assignment Operator: 8

Multiplication Assignment Operator: 20

Division Assignment Operator: 4.0

Modulus Assignment Operator: 2

Exponentiation Assignment Operator: 8

Floor Division Assignment Operator: 3
```

iv) Logical Operators

AIM

To demonstrate the use of logical operators in Python.

ALGORITHM

Step-1: Define two boolean variables.

Step-2: Use and, or, and not logical operators.

Step-3: Print the results.

PROGRAM

```
# Logical Operators
```

```
a = True
```

```
b = False
```

```
# AND operator
```

```
and_result = a and b
```

```
# OR operator
```

```
or_result = a or b
```

```
# NOT operator
```

```
not_result = not a
```

```
print(f"a and b: {and_result}")
```

```
print(f"a or b: {or_result}")
```

```
print(f"not a: {not_result}")
```

OUTPUT

a and b: False

a or b: True

not a: False

V) BitwiseOperators

AIM

To demonstrate the use of bitwise operators in Python.

ALGORITHM

Step-1: Define two integer variables.

Step-2: Use &, |, ^, ~, <<, and >> bitwise operators.

Step-3: Print the result.

PROGRAM

```
# Bitwise Operators

a = 10 # 1010 in binary
b = 4  # 0100 in binary

# AND operator
and_result = a & b

# OR operator
or_result = a | b

# XOR operator
xor_result = a ^ b

# NOT operator
```

```
not_result = ~a

# Left Shift operator

left_shift_result = a << 1

# Right Shift operator

right_shift_result = a >> 1

print(f"a & b: {and_result}")

print(f"a | b: {or_result}")

print(f"a ^ b: {xor_result}")

print(f"~a: {not_result}")

print(f"a << 1: {left_shift_result}")

print(f"a >> 1: {right_shift_result}")
```

Output:

a & b: 0

a | b: 14

a ^ b: 14

~a: -11

a << 1: 20

a >> 1: 5

vi) Ternary Operator

AIM

To demonstrate the use of the ternary operator in Python.

ALGORITHM

Step-1: Define a condition and two possible outcomes.

Step-2: Use the ternary operator to assign a value based on the condition.

Step-3: Print the result.

PROGRAM

```
# Ternary Operator

a = 10

b = 20

# Ternary operation

result = "a is greater" if a > b else "b is greater"

print(result)
```

Output:

b is greater

vii) Membership Operators

AIM

To demonstrate the use of membership operators in Python.

ALGORITHM

Step-1: Define a list and a string.

Step-2: Use in and not in membership operators.

Step-3: Print the results.

PROGRAM

```
# Define a list of numbers
numbers = [1, 2, 3, 4, 5]

# Check if a number is in the list
is_in_list = 3 in numbers

# Check if a number is not in the list
is_not_in_list = 6 not in numbers

# Print the results

print("Is 3 in the list?:", is_in_list)

print("Is 6 not in the list?:", is_not_in_list)
```

Output:

Is 3 in the list?: True

Is 6 not in the list?: True

viii) Identity Operators

AIM

To understand how identity operators (is, is not) work in Python.

ALGORITHM

Step-1: Define two variables with the same value.

Step-2: Use identity operators to check if they refer to the same object.

Step-3: Print the results.

PROGRAM

```
# Define variables

a = [1, 2, 3]

b = a

c = [1, 2, 3]

# Identity operator 'is'

result_is = a is b

result_is_not = a is not c

# Print results

print("a is b:", result_is)

print("a is not c:", result_is_not)
```

Output:

a is b: True

a is not c: True

2. What is the difference between the '==' and 'is' operators?
3. Syntax of arithmetic and bitwise operators?
4. What is a ternary operator and Assignment operators?
5. How do membership operators work and when would you use them?

Experiment-5

Date: _____

PROGRAM-5

Write a program to add and multiply complex numbers

AIM

The aim of this program is to perform addition and multiplication of two complex numbers using control flow statements in Python.

ALGORITHM

Step-1: Define a function to add two complex numbers.

Step-2: Define a function to multiply two complex numbers.

Step-3: Prompt the user to input two complex numbers.

Step-6: Display the results

PROGRAM

```
def add_complex(c1, c2):  
    return complex(c1.real + c2.real, c1.imag + c2.imag)  
  
def multiply_complex(c1, c2):  
  
    real_part = c1.real * c2.real - c1.imag * c2.imag  
    imag_part = c1.real * c2.imag + c1.imag * c2.real  
    return complex(real_part, imag_part)  
  
def main():  
    # Input complex numbers from user  
    c1 = complex(input("Enter the first complex number (in the form a+bj): "))  
    c2 = complex(input("Enter the second complex number (in the form a+bj): "))  
    # Add the complex numbers  
    sum_result = add_complex(c1, c2)  
    # Multiply the complex numbers  
    product_result = multiply_complex(c1, c2)  
    # Output the results  
    print(f"The sum of {c1} and {c2} is {sum_result}")  
    print(f"The product of {c1} and {c2} is {product_result}")  
    # Run the main function  
    if __name__ == "__main__":  
        main()
```

OUTPUT

Enter the first complex number (in the form $a+bj$): $2+3j$

Enter the second complex number (in the form $a+bj$): $4+5j$

The sum of $(2+3j)$ and $(4+5j)$ is $(6+8j)$

The product of $(2+3j)$ and $(4+5j)$ is $(-7+22j)$

VIVA QUESTIONS

1. What is a complex number and how is it represented in Python?
2. How do you add and multiply complex numbers in Python?
3. Can you explain the attributes of a complex number in Python?
4. What are the practical applications of complex numbers?
5. How does Python handle complex number arithmetic internally?

Experiment-6

Date: _____

PROGRAM-6

Write a program to print multiplication table of a given number.

AIM

The aim of this program is to generate and print the multiplication table of a given number up to a specified range.

ALGORITHM

Step-3: Use a loop to iterate from 1 to the specified range.

Step-4: For each iteration, multiply the given number by the current loop index.

Step-5: Print the result in a formatted way.

PROGRAM

```
def main():

    # Input the number for which the multiplication table will be generated

    number = int(input("Enter the number for which you want the multiplication table: "))

    # Input the range up to which the table should be printed

    table_range = int(input("Enter the range up to which the table should be printed: "))


    # Print the multiplication table using a for loop


    for i in range(1, table_range + 1):

        result = number * i

        print(f"{number} x {i} = {result}")

# Run the main function

if __name__ == "__main__":

    main()
```

OUTPUT

Enter the number `for` which you want the multiplication `table`: `5`

Enter the `range` up `to` which the `table` should be printed: `10`

`5 x 1 = 5`

`5 x 2 = 10`

`5 x 3 = 15`

`5 x 4 = 20`

`5 x 5 = 25`

`5 x 6 = 30`

`5 x 7 = 35`

`5 x 8 = 40`

`5 x 9 = 45`

`5 x 10 = 50`

1. How do you generate a multiplication table in Python?
2. Can you modify the program to print the multiplication table for a range of numbers?
3. What are the different ways to format the output in Python?
4. How can you take user input for the number and the range for the multiplication table?
5. What are the best practices for writing loops in Python?

Experiment-7

Date: _____

PROGRAM-7

Write a program to define a function with multiple return values.

AIM

To write a Python program that defines a function which returns multiple values.

ALGORITHM

Step-1: Define the function:

- Create a function that performs some operations and returns multiple values.

Perform operations

- Inside the function, perform necessary calculations or operations.

Step-2: Return multiple values:

- Use the return statement to return multiple values as a tuple.

Step-3: Call the function:

- Call the function and unpack the returned values.

Step-4: Display the results:

- Print the results to verify the returned values.

PROGRAM

```
# Define the function that returns multiple values

def calculate(a, b):

    sum_val = a + b

    diff_val = a - b

    prod_val = a * b

    return sum_val, diff_val, prod_val

# Call the function and unpack the returned values

num1 = 10

num2 = 5

sum_result, diff_result, prod_result = calculate(num1, num2)

# Display the results

print(f"Sum: {sum_result}")

print(f"Difference: {diff_result}")

print(f"Product: {prod_result}")
```

OUTPUT

Sum: 15

Difference: 5

Product: 50

1. How can a function return multiple values in Python?
2. What is tuple unpacking, and how is it used with functions returning multiple values?
3. Can you provide an example where returning multiple values from a function is useful?
4. How does Python handle multiple return values internally?
5. Can you return different data types from a function simultaneously?

Experiment-8

Date: _____

PROGRAM-8

Write a program to define a function using default arguments.

AIM

To write a Python program that defines a function using default arguments.

ALGORITHM

Step-1: Define the function

- Create a function that uses default arguments.

Step-2: Set default arguments

- Specify default values for some or all parameters in the function definition.

Step-3: Call the function

- Call the function with and without providing arguments to see the effect of default values.

PROGRAM

Define the function with default arguments

```
def add_numbers(a, b=10):
```

```
    return a + b
```

Call the function with both arguments

```
result1 = add_numbers(5, 3)
```

Call the function with only one argument

```
result2 = add_numbers(7)
```

Display the results

```
print(f"Result when both arguments are provided: {result1}")
```

```
print(f"Result when only one argument is provided (default value used): {result2}")
```

OUTPUT

When the above program is executed, the output will be:

sql

Copy code

Result when both arguments are provided: 8

Result when only one argument is provided (default value used): 17

VIVA QUESTIONS

1. What are default arguments in Python?
2. How do you define a function with default arguments?

values are defined?

4. Can you provide an example where default arguments are particularly useful?
5. What are the potential pitfalls of using mutable default arguments?

Experiment-9

Date: _____

PROGRAM-9

Write a program to find the length of the string without using any library functions

AIM

The aim of this program is to find the length of a string in Python without using any built-in library functions like len().

ALGORITHM

Step-1: Initialization: Start with a counter variable initialized to zero.

Step-2: Iteration: Iterate through each character in the string.

Step-3: Counting: Increment the counter for each character until the end of the string (when there are no more characters left to iterate over).

Step-4: Output: Print or return the counter value, which represents the length of the string.

PROGRAM

```
for _ in s:
```

```
    length += 1
```

```
return length
```

```
def main():
```

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

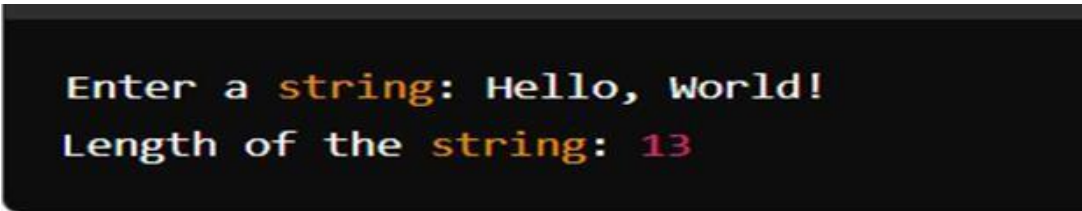
```
    length = string_length(input_string)
```

```
    print(f"Length of the string: {length}")
```

```
if __name__ == "__main__":
```

```
    main()
```

OUTPUT

A screenshot of a terminal window with a dark background. It shows the program's output: "Enter a string: Hello, World!" followed by "Length of the string: 13". The word "string" is highlighted in orange in both lines, and the number "13" is highlighted in red in the second line.

```
Enter a string: Hello, World!  
Length of the string: 13
```

1. How can you determine the length of a string without using built-in functions like len()?
2. Can you explain the logic used to count characters in a string manually?
3. How would you handle different types of input, such as an empty string or non-string data?
4. What is the time complexity of your string length calculation method?
5. Can you think of any edge cases that might affect the accuracy of your string length calculation?

Experiment-10

Date: _____

PROGRAM-10

Write a program to check if the substring is present in a given string or not.

AIM

To write a Python program that uses string concepts to check if a given substring is present in a specified string

ALGORITHM

Step-1: Input: Read the main string and the substring from the user.

Step-2: Check Presence: Use the in operator to determine if the substring is part of the main string.

Step-3: Output: Print a message indicating whether the substring is found in the main string.


```
def is_substring_present(main_string, substring):

    # Using the 'in' operator to check for presence of substring

    return substring in main_string

# Main function to get input and print result

def main():

    # Input: Main string and substring

    main_string = input("Enter the main string: ")

    substring = input("Enter the substring: ")

    # Check if substring is present and print result

    if is_substring_present(main_string, substring):

        print(f"The substring '{substring}' is present in the main string.")

    else:

        print(f"The substring '{substring}' is not present in the main string.")

# Execute the main function

if __name__ == "__main__":

    main()
```

OUTPUT

Enter the main **string**: Hello, welcome to OpenAI!

Enter the substring: OpenAI

The substring 'OpenAI' **is** present **in** the main **string**.

1. How do you check if a substring is present in a given string in Python?
2. What built-in operators or functions can be used to find a substring in a string?
3. How would you handle case sensitivity when checking for substrings?
4. Can you write a function that performs the same task without using built-in operators?
5. How would you improve the efficiency of substring search in large texts?

Experiment-11

Date: _____

PROGRAM-11

Write a program to perform the given operations on a list:

- i. Addition ii. Insertion iii. Slicing

AIM

The aim of this task is to demonstrate basic list operations in Python:

- **Addition (Append):** Add an element to the end of the list.
- **Insertion:** Insert an element at a specified index in the list.
- **Slicing:** Extract a sub list from the list based on given start and end indices.

ALGORITHM

Step-1: Addition (Append)

- Initialize a list with some initial elements.

Step-3: Slicing

- Use slicing (:) to extract a portion of the list based on start and end indices.

PROGRAM

```
def main():  
  
    # Initialize a list  
  
    my_list = [1, 2, 3, 4, 5]  
  
    # i. Addition  
  
    print("Original list:", my_list)  
  
    my_list.append(6) # Append element 6  
  
    print("After addition (append):", my_list)  
  
    # ii. Insertion  
  
    my_list.insert(2, 10) # Insert 10 at index 2  
  
    print("After insertion at index 2:", my_list)  
  
    # iii. Slicing  
  
    sliced_portion = my_list[1:4] # Slice from index 1 to 3 (exclusive of 4)  
  
    print("Sliced portion (index 1 to 3):", sliced_portion)  
  
if __name__ == "__main__":  
  
    main()
```

OUTPUT

Original list: [1, 2, 3, 4, 5]

After addition (append): [1, 2, 3, 4, 5, 6]

After insertion at index 2: [1, 2, 10, 3, 4, 5, 6]

Sliced portion (index 1 to 3): [2, 10, 3]

VIVA QUESTIONS

1. How do you add elements to a list in Python?
4. Can you explain list slicing with an example?
5. How do negative indices work in list slicing?

Experiment-12

Date: _____

PROGRAM-12

Write a program to perform any 5 built-in functions by taking any list.

AIM

The aim of this task is to showcase the usage of five built-in functions that operate on lists in Python.

ALGORITHM

Step-1: Initialization

- Create a list with some initial elements.

Step-2: Built-in Functions

- `len()`: Determine the length of the list.
- `max()`: Find the maximum element in the list.
- `min()`: Find the minimum element in the list.
- `sum()`: Calculate the sum of all elements in the list.
- `sorted()`: Return a new sorted list from the elements of the given iterable.

PROGRAM

```
def main():  
    # Initialize a list  
    my_list = [7, 2, 9, 1, 5, 3, 6, 4, 8]  
    print("Original List:", my_list)  
    # i. len()  
    print("Length of the list:", len(my_list))  
    # ii. max()  
    print("Maximum element in the list:", max(my_list))  
    # iii. min()  
    print("Minimum element in the list:", min(my_list))  
    # iv. sum()  
    print("Sum of elements in the list:", sum(my_list))  
    # v. sorted()  
    sorted_list = sorted(my_list)  
    print("Sorted list:", sorted_list)  
if __name__ == "__main__":  
    main()
```

OUTPUT

Length of the list: 8

Maximum value in the list: 9

Minimum value in the list: 1

Sorted list: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Sum of all elements in the list: 42

VIVA QUESTIONS

1. What are some common built-in functions available for lists in Python?
2. Syntax of len() & sum() function on a list?
3. How do you find the maximum and minimum values in a list using built-in functions?
4. What is the purpose of the sorted() function, and how does it differ from the sort() method?
5. How do you use the enumerate() function with lists, and what are its advantages?

Experiment-13

Date: _____

PROGRAM-13

Write a program to create tuples (name, age, address, college) for at least two members and concatenate the tuples and print the concatenated tuples.

AIM

To write a Python program to create tuples containing details of members (name, age, address, college), concatenate the tuples, and print the concatenated tuple.

ALGORITHM

Step-1: Define the tuples: Create tuples for at least two members with the details (name, age, address, college).

Step-2: Concatenate the tuples: Use the + operator to concatenate the tuples.

Step-3: Print the concatenated tuple: Display the result of the concatenation.

PROGRAM

```
# Define the tuples for two members
```

```
member1 = ("Alice", 22, "123 Main St, Springfield", "Springfield University")
```

```
print("Concatenated Tuple:", concatenated_tuple)
```

OUTPUT

Concatenated Tuple: ('Alice', 22, '123 Main St, Springfield', 'Springfield University', 'Bob', 23, '456 Elm St, Shelbyville', 'Shelbyville College')

VIVA QUESTIONS

1. What is a tuple in Python?
2. How do you define a tuple in Python?
3. What does immutability mean in the context of tuples?
4. How can you concatenate two tuples?
5. Write Syntax of tuple?

Experiment-14

Write a program to count the number of vowels in a string (No control flow allowed).

AIM

To write a program that counts the number of vowels in a given string without using control flow statements (like if, for, while).

ALGORITHM

Step-1: Define Input String: Define the input string containing the text to be analysed.

Step-2: Create Vowel Set: Create a set of vowels for easy lookup.

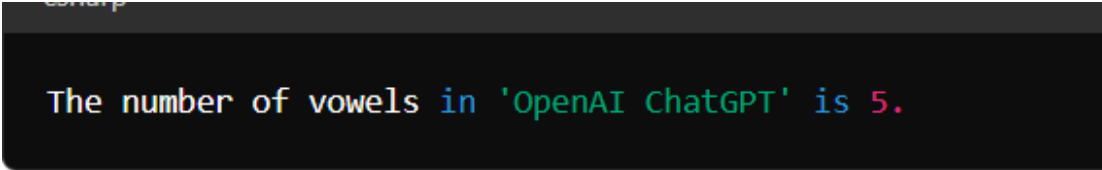
Step-3: Count Vowels Using Built-in Functions: Use Python's built-in functions and methods to count the number of vowels without explicit control flow.

PROGRAM

```
def count_vowels(s):  
  
    vowels = set('aeiouAEIOU')  
  
    return sum(map(s.count, vowels))  
  
# Example usage  
  
input_string = "OpenAI ChatGPT"  
  
vowel_count = count_vowels(input_string)  
  
print(f"The number of vowels in '{input_string}' is {vowel_count}.")
```

Lab Record

OUTPUT



```
The number of vowels in 'OpenAI ChatGPT' is 5.
```

VIVA QUESTIONS

1. What is a set in Python?
2. Why did you use a set for the vowels?

3. How does the count method work in Python?
4. Why is there no need for explicit control flow in this program?
5. What are the advantages of using functional programming concepts like map and sum?

Experiment-15

Date: _____

PROGRAM-15

Write a program to check if a given key exists in a dictionary or not.

AIM

To write a Python program that checks if a given key exists in a dictionary.

ALGORITHM

Step-1: Create a dictionary

- Define a dictionary with some key-value pairs.

Step-2: Define the function

- Create a function that takes a dictionary and a key as arguments.

Step-3: Check if the key exists

- Use the in keyword to check if the key exists in the dictionary.

Step-4: Return the result

- Return a message indicating whether the key exists or not.

Step-5: Call the function

- Call the function with a dictionary and a key to test the implementation.

Step-6: Display the result

Python Program to Check if a Given Key Exists in a Dictionary or Not

PROGRAM

```
# Define the function to check if a key exists in a dictionary

def check_key_existence(dictionary, key):

    if key in dictionary:

        return f"The key '{key}' exists in the dictionary."

    else:

        return f"The key '{key}' does not exist in the dictionary."

# Create a dictionary with some key-value pairs

sample_dict = {

    'name': 'Alice',

    'age': 25,

    'city': 'New York'

}

# Test the function with an existing key

result1 = check_key_existence(sample_dict, 'age')

# Test the function with a non-existing key

result2 = check_key_existence(sample_dict, 'country')

# Display the results

print(result1)

print(result2)
```

OUTPUT

The key 'age' exists in the dictionary.

The key 'country' does not exist in the dictionary.

VIVA QUESTIONS

1. What is a dictionary in Python?

3. What does the function `check_key` do?
4. Can dictionaries in Python contain duplicate keys?
5. How does Python handle operations like checking if a key exists efficiently in dictionaries?

Experiment-16

Date: _____

PROGRAM-16

Write a program to add a new key-value pair to an existing dictionary.

AIM

To write a Python program that adds a new key-value pair to an existing dictionary.

ALGORITHM

Step-1: Create a dictionary

- Define a dictionary with some initial key-value pairs.

Step-2: Define the function

- Create a function that takes a dictionary, a key, and a value as arguments.

Step-3: Add the new key-value pair

- Inside the function, add the new key-value pair to the dictionary.

Step-4: Return the updated dictionary

- Return the dictionary with the new key-value pair added.

Step-5: Call the function

- Print the updated dictionary to verify the new key-value pair has been added

PROGRAM

```
# Define the function to add a new key-value pair to a dictionary
```

```
def add_key_value_pair(dictionary, key, value):
```

```
    dictionary[key] = value
```

```
    return dictionary
```

```
# Create a dictionary with some initial key-value pairs
```

```
sample_dict = {
```

```
    'name': 'Alice',
```

```
    'age': 25,
```

```
    'city': 'New York'
```

```
}
```

```
# Add a new key-value pair to the dictionary
```

```
updated_dict = add_key_value_pair(sample_dict, 'country', 'USA')
```

```
# Display the updated dictionary
```

```
print("Updated Dictionary:", updated_dict)
```

OUTPUT

```
Updated Dictionary: {'name': 'Alice', 'age': 25, 'city': 'New York', 'country': 'USA'}
```

VIVA QUESTIONS

1. How do you add a new key-value pair to an existing dictionary in Python?

3. Why did you use a function `add_to_dict` instead of adding the key-value pair directly?
4. Is the order of items preserved when adding a new key-value pair to a dictionary?
5. How can you update an existing key's value in a dictionary without overwriting other keys?

Experiment-17

Date: _____

PROGRAM-17

Write a program to sum all the items in a given dictionary.

AIM

To write a Python program that sums all the values in a given dictionary.

ALGORITHM

Step-1: Input: Define or read the dictionary.

Step-2: Initialize Sum: Initialize a variable to hold the sum of the values.

Step-3: Iterate and Sum: Loop through the dictionary values and add each value to the sum.

Step-4: Output: Print the total sum of the values.

PROGRAM

```
# Function to sum all the values in the dictionary
```

```
def sum_dictionary_items(input_dict):
```

```
    total_sum = sum(input_dict.values())
```



```
# Main function to define a dictionary and print the sum of its values
```

```
def main():
```

```
    # Example dictionary
```

```
    my_dict = {
```

```
        'a': 10,
```

```
        'b': 20,
```

```
        'c': 30,
```

```
        'd': 40
```

```
    }
```

```
    # Calculate the sum of all items
```

```
    total = sum_dictionary_items(my_dict)
```

```
    # Print the result
```

```
    print(f"The sum of all values in the dictionary is: {total}")
```

```
# Execute the main function
```

```
if __name__ == "__main__":
```

```
    main()
```

OUTPUT

The sum of all values in the dictionary is: 100

VIVA QUESTIONS

1. How do you sum all items in a dictionary in Python?
2. What type does dictionary values() return?
3. Can dictionaries in Python contain values of different types?
4. What does the sum_dictionary_items function do?
5. What is the time complexity of summing all values in a dictionary using sum?

Experiment-18

Date: _____

PROGRAM-18

Write a program to sort words in a file and put them in another file. The output file should have only lower-case words, so any upper-case words from source must be lowered.

AIM

To create a program that reads words from a file, sorts them alphabetically, converts any uppercase words to lowercase, and then writes the sorted lowercase words into another file.

ALGORITHM

Step-1: Open the source file in read mode.

Step-2: Read all the words from the file.

Step-5: Open a destination file in write mode.

Step-6: Write each sorted lowercase word into the destination file.

Step-7: Close both files after operations are completed.

PROGRAM

```
def sort_words(input_file, output_file):

    # Open source file in read mode

    with open(input_file, 'r') as file:

        # Read all words and split them by whitespace

        words = file.read().split()

        # Convert each word to lowercase

        words_lower = [word.lower() for word in words]

        # Sort lowercase words alphabetically

        words_lower.sort()

        # Open destination file in write mode

        with open(output_file, 'w') as file:

            # Write sorted lowercase words to the destination file

            file.write("\n".join(words_lower))

            print(f"Sorted lowercase words written to {output_file}")

    # Example usage:

input_filename = 'input.txt' # Replace with your input file name

output_filename = 'output.txt' # Replace with your output file name

sort_words(input_filename, output_filename)
```

OUTPUT

Assume the contents of `input.txt` are:

```
csharp
```

```
Python is an amazing language  
OpenAI provides powerful AI models  
Sorting words in a file
```

After running the program, `output.txt` will contain:

```
csharp
```

```
ai  
amazing  
an  
file  
in  
is  
language  
models  
openai  
powerful  
provides  
python  
sorting  
words
```

VIVA QUESTIONS

1. How does your program handle reading from and writing to files in Python?
2. What does the program do to ensure that all words in the output file are in lowercase?

5. What happens if the input file contains non-alphabetic characters or punctuation?

Experiment-19

Date: _____

PROGRAM-19

Python program to print each line of a file in reverse order.

AIM

The aim is to read a file, reverse each line of text, and print each reversed line to the console.

ALGORITHM

Step-1: Open the file: Use Python's open() function to open the file in read mode.

Step-2: Read lines: Use a loop to iterate through each line of the file.

Step-3: Reverse each line: For each line read, reverse its characters.

Step-4: Print reversed lines: Output each reversed line to the console.

Step-5: Close the file: Ensure the file is properly closed after reading.

PROGRAM

```
def print_lines_in_reverse(file_name):
```

```
    try:
```

```
        # Open the file in read mode
```

```
        with open(file_name, 'r') as file:
```

```
            ...
```

```
# Process each line in reverse order
```

```
for line in reversed(lines):
```

```
    # Strip newline characters and reverse the line
```

```
    reversed_line = line.strip()[::-1]
```

```
    # Print the reversed line
```

```
    print(reversed_line)
```

```
except FileNotFoundError:
```

```
    print(f'Error: The file '{file_name}' was not found.'))
```

```
except IOError:
```

```
    print(f'Error: Could not read from the file '{file_name}'.')
```

```
# Example usage:
```

```
file_name = 'example.txt' # Replace with your file name
```

```
print_lines_in_reverse(file_name)
```

OUTPUT

Hello, world!

This is line 2.

Another line.

.enil rehtonA

.2 enil si sihT

!dlrow ,olle

2. Explain the method you used to reverse each line of text.
3. What is the expected output format when printing each line in reverse order?
4. How does your program handle files with varying line lengths or empty lines?
5. Can you explain any potential performance considerations for large files when implementing this program?

Experiment-20

Date: _____

PROGRAM-20

Python program to compute the number of characters, words and lines in a file.

AIM

The aim is to create a Python program that reads a file and computes the number of characters, words, and lines present in that file.

ALGORITHM

Step-1: Open the file in read mode.

Step-2: Initialize counters for characters, words, and lines.

Step-3: Loop through each line in the file:

Step-4: Close the file once done.

Step-5: Print the counts of characters, words, and lines.

PROGRAM

```
def count_chars_words_lines(filename):

    try:

        with open(filename, 'r') as file:

            num_chars = 0

            num_words = 0

            num_lines = 0

            for line in file:

                num_lines += 1

                words = line.split()

                num_words += len(words)

                num_chars += len(line)

            print("Number of characters:", num_chars)

            print("Number of words:", num_words)

            print("Number of lines:", num_lines)

    except FileNotFoundError:

        print("File not found. Please check the file path and try again.")

    except IsADirectoryError:

        print("The path specified is a directory, not a file.")

    except Exception as e:

        print("An error occurred:", str(e))


# Example usage:

if __name__ == "__main__":
```

```
filename = 'sample.txt' # Replace with your file name or path
```

```
count_chars_words_lines(filename)
```

OUTPUT

This is a sample text file.

It has multiple lines.

Each line contains several words.

Number of characters: 89

Number of words: 14

Number of lines

VIVA QUESTIONS

1. How does your program count the number of characters in a file?
2. What method do you use to count words? How do you define a word in your program?

4. What happens if the file contains special characters, symbols, or non-alphanumeric characters?
5. Can your program handle very large files efficiently? If so, how?

Experiment-21

Date: _____

PROGRAM-21

Write a program to create, display, append, insert and reverse the order of the items in the array.

AIM

To create a program that allows manipulation of a list including creating, displaying, appending, inserting, and reversing its contents.

ALGORITHM

Step-1: Create List: Initialize an empty list to store items.

Step-2: Display List: Print all items currently in the list.

Step-3: Append Item: Add a new item to the end of the list.

Step-4: Insert Item: Insert a new item at a specified position in the list.

Step-5: Reverse List: Reverse the order of items in the list.

Step-6: Display List (Again): Print all items in the list after each operation to verify changes.

PROGRAM

```
# Initialize an empty list
```

```
my_list = []
```

```
# Display initial list
```

```
print("Initial List:", my_list)
```

```
# Append item to the list
```

```
my_list.append('apple')
```

```
print("After appending 'apple':", my_list)
```

```
# Insert item at a specific position
```

```
my_list.insert(0, 'banana')
```

```
print("After inserting 'banana' at position 0:", my_list)
```

```
# Reverse the list
```

```
my_list.reverse()
```

```
print("After reversing the list:", my_list)
```

```
if __name__ == "__main__":
```

```
    main()
```

OUTPUT

Initial List: []

After appending 'apple': ['apple']

After inserting 'banana' at position 0: ['banana', 'apple']

After reversing the list: ['apple', 'banana']

VIVA QUESTIONS

1. What is the purpose of initializing my_list as an empty list at the beginning of the program?
2. How does the append method differ from the insert method when adding new items to a list?
3. What happens if you try to insert an item at an index that is out of range for the list?
4. How does the reverse method change the order of elements in a list?
5. Write Syntax of append()?

Experiment-22

Date: _____

PROGRAM-22

Write a program to add, transpose and multiply two matrices.

AIM

To create a program that performs basic matrix operations such as addition, transpose, and multiplication.

ALGORITHM

Step-1: Input Matrices: Prompt the user to enter dimensions and elements for two matrices.

Step-2: Matrix Transpose: Transpose the matrices if they are compatible (columns of one

first matrix equal rows of the second matrix).

Step-5: Display Results: Print the original matrices, result of addition, transpose, and multiplication.

PROGRAM

```
def input_matrix(rows, cols):  
    matrix = []  
  
    print(f"Enter {rows}x{cols} matrix elements:")
```

111

```

for i in range(rows):

    row = list(map(int, input().split()))

    matrix.append(row)

return matrix

def display_matrix(matrix):

    for row in matrix:

        print(row)

def add_matrices(matrix1, matrix2):

    result = []

    for i in range(len(matrix1)):

        row = []

        for j in range(len(matrix1[0])):

            row.append(matrix1[i][j] + matrix2[i][j])

        result.append(row)

    return result

```

```

def transpose_matrix(matrix):

    result = []

    for j in range(len(matrix[0])):

        row = []

        for i in range(len(matrix)):

            row.append(matrix[i][j])

        result.append(row)

    return result

def multiply_matrices(matrix1, matrix2):

    rows1, cols1 = len(matrix1), len(matrix1[0])

    rows2, cols2 = len(matrix2), len(matrix2[0])

    if cols1 != rows2:

        return "Matrices cannot be multiplied."

    result = [[0 for _ in range(cols2)] for _ in range(rows1)]

    for i in range(rows1):

        for j in range(cols2):

            for k in range(cols1):

                result[i][j] += matrix1[i][k] * matrix2[k][j]

    return result

def main():

    print("Matrix 1:")

```

```
rows1 = int(input("Enter number of rows: "))

cols1 = int(input("Enter number of columns: "))

matrix1 = input_matrix(rows1, cols1)

print("\nMatrix 2:")

rows2 = int(input("Enter number of rows: "))

cols2 = int(input("Enter number of columns: "))

matrix2 = input_matrix(rows2, cols2)


print("\nMatrix 1:")

display_matrix(matrix1)

print("\nMatrix 2:")

display_matrix(matrix2)

print("\nMatrix Addition:")

if rows1 == rows2 and cols1 == cols2:

    addition_result = add_matrices(matrix1, matrix2)

    display_matrix(addition_result)

else:

    print("Matrices are not of the same dimensions. Addition not possible.")

print("\nTranspose of Matrix 1:")

transpose_result = transpose_matrix(matrix1)

display_matrix(transpose_result)

print("\nMatrix Multiplication:")

multiplication_result = multiply_matrices(matrix1, matrix2)
```

```
if isinstance(multiplication_result, str):

    print(multiplication_result)

else:

    display_matrix(multiplication_result)

if __name__ == "__main__":

    main()
```

OUTPUT

Matrix 1:

Enter number of rows: 2

Enter number of columns: 2

Enter 2x2 matrix elements:

1 2

3 4

Matrix 2:

Enter number of rows: 2

Enter number of columns: 2

Enter 2x2 matrix elements:

5 6

7 8

Matrix 1:

[1, 2]

[3, 4]

Matrix 2:

[5, 6]

[7, 8]

Matrix Addition:

[6, 8]

[10, 12]

Transpose of Matrix 1:

[1, 3]

[2, 4]

Matrix Multiplication:

[19, 22]

[43, 50]

1. What is a matrix and how is it represented in programming?
2. How do you perform matrix multiplication, and what are the necessary conditions for it?
3. What are some potential edge cases to consider when working with matrices in programming?
4. How does the program handle invalid input or incompatible matrix dimensions?
5. What is a matrix and how is it represented in programming?

Experiment-23

Date: _____

PROGRAM-23

Write a Python program to create a class that represents a shape. Include methods to calculate its area and perimeter. Implement subclasses for different shapes like circle, triangle, and square.

AIM

To write a Python program that creates a class representing a shape, including methods to calculate its area and perimeter, and implement subclasses for different shapes like circle, triangle, and square.

ALGORITHM

Step-1: Define the base class Shape

- Create a class called Shape with methods to calculate the area and perimeter. These methods

Step-2: Define subclasses for specific shapes

- Create subclasses for Circle, Triangle, and Square, each inheriting from the Shape class.

Step-3: Implement methods in each subclass:

- Override the methods to calculate the area and perimeter for each specific shape in its respective subclass.

Step-4: Create instances and test

- Create instances of each subclass and call the methods to calculate the area and perimeter.

Step-5: Display the results

- Print the calculated area and perimeter for each shape.

PROGRAM

```
from math import pi, sqrt

# Define the base class `Shape`

class Shape:

    def area(self):

        pass

    def perimeter(self):

        pass

# Define the subclass for `Circle`

class Circle(Shape):

    def __init__(self, radius):

        self.radius = radius

    def area(self):

        return pi * self.radius**2

    def perimeter(self):

        return 2 * pi * self.radius

# Define the subclass for `Triangle`

class Triangle(Shape):

    def __init__(self, a, b, c):

        self.a = a

        self.b = b
```



```

    self.c = c

def area(self):

    # Using Heron's formula

    s = (self.a + self.b + self.c) / 2

    return sqrt(s * (s - self.a) * (s - self.b) * (s - self.c))

def perimeter(self):

    return self.a + self.b + self.c

# Define the subclass for `Square`

class Square(Shape):

    def __init__(self, side):

        self.side = side

    def area(self):

        return self.side**2

    def perimeter(self):

        return 4 * self.side

# Create instances of each shape

circle = Circle(5)

triangle = Triangle(3, 4, 5)

square = Square(4)


# Display the results

print("Circle:")

print(f"Area: {circle.area()}")

print(f"Perimeter: {circle.perimeter()}")

print("\nTriangle:")

print(f"Area: {triangle.area()}")

print(f"Perimeter: {triangle.perimeter()}")

```

```
print(f"Perimeter: {square.perimeter()}")
```

OUTPUT

Circle:

Area: 78.53981633974483

Perimeter: 31.41592653589793

Triangle:

Area: 6.0

Perimeter: 12

Square:

Area: 16

Perimeter: 16

VIVA QUESTIONS

1. Syntax of classes and objects in Python?
2. How do you define a class in Python?
3. What are the differences between abstract classes and concrete classes?
4. How do you enforce that a subclass implements certain methods?
5. What is the use of the `__init__` method in a class?

Experiment-24

Date: _____

PROGRAM-24

Write a Python program that checks whether a JSON string contains a complex object or not

AIM

To write a Python program that checks whether a JSON string contains a complex object or not.

ALGORITHM

Step-1: Import necessary modules

- Import the json module for handling JSON data.

Step-2: Define a function to check for complex objects

- Create a function that takes a JSON string as input.
- Parse the JSON string into a Python dictionary or list.
- Recursively check the parsed object for nested dictionaries or lists.

Step-3: Implement the recursive checking function

- If the object is a dictionary or list, iterate through its elements.
- If any element is a dictionary or list, return True indicating it is complex.
- Otherwise, return False.

Step-4: Test the function

- Provide JSON strings with simple and complex objects.
- Call the function with these strings and print the results.

PROGRAM

```
import json

def is_complex(json_str):

    # Parse the JSON string into a Python object

    try:

        obj = json.loads(json_str)

    except json.JSONDecodeError:

        return False

    # Define a recursive function to check for nested dictionaries or lists

    def check_complexity(obj):

        if isinstance(obj, dict):

            for value in obj.values():

                if isinstance(value, (dict, list)):

                    return True

            elif isinstance(obj, list):

                for item in obj:

                    if isinstance(item, (dict, list)):

                        return True

            return False

        return check_complexity(obj)

    # Test cases

    json_str_simple = '{"name": "Alice", "age": 25, "city": "New York"}'
```

```
json_str_complex = '{"name": "Alice", "address": {"city": "New York", "zip": "10001"},  
"phones": ["123-456-7890", "987-654-3210"]}'  
  
# Display the results  
  
print("Simple JSON string:")  
  
print(f"Is complex: {is_complex(json_str_simple)}")  
  
print("\nComplex JSON string:")  
  
print(f"Is complex: {is_complex(json_str_complex)}")
```

OUTPUT

Simple JSON string:

Is complex: False

Complex JSON string:

Is complex: True

VIVA QUESTIONS

1. How do you parse a JSON string in Python?
2. What constitutes a “complex object” in the context of JSON?

4. What is the significance of the json.loads() function in Python?
5. How do you convert a Python dictionary to a JSON string?

Experiment-25

Date: _____

PROGRAM-25

Python Program to demonstrate NumPy arrays creation using array () Function.

AIM

To demonstrate the creation of NumPy arrays using the numpy.array() function.

ALGORITHM

Step-1: Import the NumPy library.

Step-2: Use the numpy.array() function to create different types of NumPy arrays:

- Create a 1-dimensional array.
- Create a 2-dimensional array.
- Create an array with specific data types.

Step-3: Print and display the created arrays.

PROGRAM

```
# Importing NumPy library
```

```
import numpy as np
```

```
arr1 = np.array([1, 2, 3, 4, 5])
```

```
# Creating a 2-dimensional array
```

```
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
```

```
# Creating an array with specific data type
```

```
arr3 = np.array([1, 2, 3], dtype=float)
```

```
# Printing the arrays
```

```
print("1-dimensional array:")
```

```
print(arr1)
```

```
print("\n2-dimensional array:")
```

```
print(arr2)
```

```
print("\nArray with specific data type:")
```

```
print(arr3)
```

OUTPUT

lua

1-dimensional array:

[1 2 3 4 5]

2-dimensional array:

[[1 2 3]

[4 5 6]]

Array with specific data type:

[1. 2. 3.]

VIVA QUESTIONS

1. What is NumPy and why is it important in Python programming?
2. What is a NumPy array and how does it differ from a Python list?
3. How do you create a NumPy array using the array() function?
4. What is the purpose of the dtype parameter in the array() function?
5. How do you convert an existing Python sequence into a NumPy array?

Experiment-26

Date: _____

PROGRAM-26

Python program to demonstrate use of ndim, shape, size, dtype.

AIM

To demonstrate the use of ndim, shape, size, and dtype attributes in Python to understand the dimensions, shape, size, and data type of NumPy arrays.

ALGORITHM

Step-1: Import the NumPy library.

Step-4: Display the output.

PROGRAM

```
import numpy as np

# Creating a NumPy array

arr = np.array([[1, 2, 3], [4, 5, 6]])


# Displaying the attributes

print("Array:")

print(arr)

print("\nAttributes:")

print("Number of dimensions (ndim):", arr.ndim)

print("Shape:", arr.shape)

print("Size:", arr.size)

print("Data type (dtype):", arr.dtype)
```

OUTPUT

Array:

```
[[1 2 3]
 [4 5 6]]
```

Attributes:

Number of dimensions (ndim): 2

Shape: (2, 3)

Size: 6

Data type (dtype): int64

VIVA QUESTIONS

1. What does the ndim attribute of a NumPy array represent and how is it used?
2. How can you find the shape of a NumPy array using the shape attribute?
3. What information does the size attribute of a NumPy array provide?
4. How do you check the data type of elements in a NumPy array using the dtype attribute?
5. Write syntaxes of ndim, shape, size, dtype in the context of NumPy arrays?

Experiment-27

Date: _____

PROGRAM-27

Python program to demonstrate basic slicing, integer and Boolean indexing.

AIM

To demonstrate basic slicing, integer indexing, and Boolean indexing in Python lists.

ALGORITHM

Step-1: Basic Slicing

- Create a list of integers.

- Create a list of strings.
- Use integer indices to access specific elements from the list.

Step-3: Boolean Indexing

- Create a list of numbers.
- Use Boolean conditions to filter elements from the list.

PROGRAM

Basic Slicing Example

```
def demonstrate_basic_slicing():
```

```
    print("--- Basic Slicing Example ---")
```

```
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
    # Slicing to get subsets
```

```
    subset1 = numbers[2:7] # elements from index 2 to 6 (index 7 is exclusive)
```

```
    subset2 = numbers[:5] # elements from start to index 4
```

```
    subset3 = numbers[7:] # elements from index 7 to end
```

```
    print("Original List:", numbers)
```

```
    print("Subset 1 (numbers[2:7]):", subset1)
```

```
    print("Subset 2 (numbers[:5]):", subset2)
```

```
    print("Subset 3 (numbers[7:]):", subset3)
```

```
    print()
```

Integer Indexing Example

```
def demonstrate_integer_indexing():
```

```
    print("--- Integer Indexing Example ---")
```

```
    fruits = ["apple", "banana", "cherry", "date", "elderberry"]
```

```
# Accessing elements using integer indices
```

```
fruit1 = fruits[0] # first element
```

```
fruit2 = fruits[3] # fourth element
```

```
print("Original List:", fruits)
```

```
print("First Fruit (fruits[0]):", fruit1)
```

```
print("Fourth Fruit (fruits[3]):", fruit2)
```

```
print()
```

```
# Boolean Indexing Example
```

```
def demonstrate_boolean_indexing():
```

```
    print("--- Boolean Indexing Example ---")
```

```
    numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
    # Filtering elements using Boolean indexing
```

```
    filtered_numbers = [num for num in numbers if num > 50]
```

```
    print("Original List:", numbers)
```

```
    print("Filtered Numbers (numbers > 50):", filtered_numbers)
```

```
    print()
```

```
# Main function to run examples
```

```
def main():
```

```
    demonstrate_basic_slicing()
```

```
demonstrate_integer_indexing()
```

```
demonstrate_boolean_indexing()
```

```
if __name__ == "__main__":
```

```
    main()
```

OUTPUT

--- Basic Slicing Example ---

Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Subset 1 (numbers[2:7]): [3, 4, 5, 6, 7]

Subset 2 (numbers[:5]): [1, 2, 3, 4, 5]

Subset 3 (numbers[7:]): [8, 9, 10]

--- Integer Indexing Example ---

Original List: ['apple', 'banana', 'cherry', 'date', 'elderberry']

First Fruit (fruits[0]): apple

Fourth Fruit (fruits[3]): date

--- Boolean Indexing Example ---

Original List: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Filtered Numbers (numbers > 50): [60, 70, 80, 90, 100]

VIVA QUESTIONS

1. What is slicing in NumPy, and how is it different from slicing in regular Python lists?
2. How do you perform basic slicing on a NumPy array to extract a subarray?

4. How is Boolean indexing used in NumPy, and what are its typical use cases?
5. Write Syntaxes of slicing, integer indexing, and Boolean indexing in NumPy array?

Experiment-28

Date: _____

PROGRAM-28

Python program to find min, max, sum, cumulative sum of array

AIM

To compute and display the minimum value, maximum value, sum, and cumulative sum of elements in an array.

ALGORITHM

Step-1: Initialize the Array: Create an array of integers.

Step-2: Find Minimum and Maximum Values

- Initialize variables min_value and max_value with the first element of the array.
- Iterate through the array to update min_value and max_value as needed.

Step-3: Calculate Sum of Array

- Initialize a variable array_sum to zero.
- Iterate through the array and accumulate the sum in array_sum.

Step-4: Compute Cumulative Sum

- Initialize an empty list cumulative_sum.
- Iterate through the array, keeping a running total of the cumulative sum and append each cumulative sum to cumulative_sum.

Step-5: Display Results: Print the minimum value, maximum value, sum, and cumulative sum.

PROGRAM

Python program to find min, max, sum, and cumulative sum of an array

```
def array_operations(arr):  
  
    # Initialize variables  
  
    min_value = arr[0]  
  
    max_value = arr[0]  
  
    array_sum = 0  
  
    cumulative_sum = []  
  
    current_cumulative_sum = 0  
  
    # Find min, max, and sum  
  
    for num in arr:  
  
        if num < min_value:  
  
            min_value = num  
  
        if num > max_value:  
  
            max_value = num  
  
        array_sum += num  
  
    # Calculate cumulative sum  
  
    for num in arr:  
  
        current_cumulative_sum += num  
  
        cumulative_sum.append(current_cumulative_sum)
```

```
# Display results

print(f"Original Array: {arr}")

print(f"Minimum Value: {min_value}")

print(f"Maximum Value: {max_value}")

print(f"Sum of Array: {array_sum}")

print(f"Cumulative Sum: {cumulative_sum}")

# Example usage:

if __name__ == "__main__":

    array = [5, 2, 9, 1, 7, 3]

    array_operations(array)
```

OUTPUT

Original Array: [5, 2, 9, 1, 7, 3]

Minimum Value: 1

Maximum Value: 9

Sum of Array: 27

Cumulative Sum: [5, 7, 16, 17, 24, 27]

VIVA QUESTIONS

1. How do you find the minimum value in a NumPy array using a built-in function?
2. What function would you use to determine the maximum value in a NumPy array?
3. How can you calculate the sum of all elements in a NumPy array?
4. What is the purpose of the cumsum function in NumPy, and how does it work?
5. Write the use NumPy to find the cumulative sum of a 2D array along different axes?

Experiment-29

Date: _____

PROGRAM-29

Create a dictionary with at least five keys and each key represent value as a list where this list contains at least ten values and convert this dictionary as a pandas data frame and explore the data through the data frame as follows.

- a) Apply head () function to the pandas data frame
- b) Perform various data selection operations on Data Frame

AIM

To create a dictionary with at least five keys, where each key has a list of at least ten values, convert this dictionary into a Pandas DataFrame, and explore the data using the DataFrame's head() function and various data selection operations.

ALGORITHM

Step-1: Create the Dictionary

- Define a dictionary with five keys.

- Assign each key a list of at least ten values.

Step-2: Convert Dictionary to DataFrame

- Use the `pandas.DataFrame` function to convert the dictionary into a DataFrame.

Step-3: Explore the DataFrame

- Use the `head()` function to view the first few rows of the DataFrame.
- Perform various data selection operations, such as selecting specific columns, rows, or subsets of the DataFrame.

PROGRAM

```
import pandas as pd

# Step 1: Create the dictionary

data = {

    'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

    'B': [11, 12, 13, 14, 15, 16, 17, 18, 19, 20],

    'C': [21, 22, 23, 24, 25, 26, 27, 28, 29, 30],

    'D': [31, 32, 33, 34, 35, 36, 37, 38, 39, 40],

    'E': [41, 42, 43, 44, 45, 46, 47, 48, 49, 50]

}

# Step 2: Convert the dictionary to a pandas DataFrame

df = pd.DataFrame(data)

# Step 3a: Apply the head() function to the DataFrame

print("First five rows of the DataFrame using head():")

print(df.head())

# Step 3b: Perform various data selection operations

# Select a specific column

print("\nColumn 'B':")
```

```
print(df['B'])

# Select multiple columns

print("\nColumns 'A' and 'C':")

print(df[['A', 'C']])

# Select a specific row by index

print("\nRow with index 2:")

print(df.iloc[2])

# Select multiple rows

print("\nRows with indices 1 to 3:")

print(df.iloc[1:4])

# Select rows based on a condition

print("\nRows where column 'A' is greater than 5:")

print(df[df['A'] > 5])
```

OUTPUT

First five rows of the DataFrame using head():

	A	B	C	D	E
0	1	11	21	31	41
1	2	12	22	32	42

2 3 13 23 33 43

3 4 14 24 34 44

4 5 15 25 35 45

Column 'B':

0 11

1 12

2 13

3 14

4 15

5 16

6 17

7 18

8 19

9 20

Name: B, dtype: int64

Columns 'A' and 'C':

A C

0 1 21

1 2 22

2 3 23

3 4 24

4 5 25

5 6 26

6 7 27

7 8 28

8 9 29

9 10 30

Row with index 2:

A 3

B 13

C 23

D 33

E 43

Name: 2, dtype: int64

Rows with indices 1 to 3:

A B C D E

1 2 12 22 32 42

2 3 13 23 33 43

3 4 14 24 34 44

Rows where column 'A' is greater than 5:

A B C D E

5 6 16 26 36 46

6 7 17 27 37 47

7 8 18 28 38 48

8 9 19 29 39 49

9 10 20 30 40 50

VIVA QUESTIONS

1. How do you create a dictionary in Python, and what are some common data types that can be used as values in a dictionary?
2. What is the process of converting a dictionary into a Pandas DataFrame?
3. Write Syntaxes of head() function in Pandas, and how does it help in exploring the initial rows of a DataFrame?
4. What are some common methods and attributes used for data selection in a Pandas DataFrame?
5. How would you select specific columns from a Pandas DataFrame, and what methods would you use for slicing rows based on specific conditions?

Experiment-30

Date: _____

PROGRAM-30

Select any two columns from the above data frame, and observe the change in one attribute with respect to other attribute with scatter and plot operations in matplotlib

AIM

To observe the relationship between two selected columns from a Pandas DataFrame by creating a scatter plot using Matplotlib.

ALGORITHM

Step-1: Create a Pandas DataFrame: Initialize a dictionary with at least five keys, where each key represents a list with at least ten values.

Step-2: Convert to DataFrame: Convert the dictionary into a Pandas DataFrame.

Step-3: Select Columns: Choose any two columns from the DataFrame.

Step-4: Plotting: Use Matplotlib to create a scatter plot where one column is plotted against the other.

Step-5: Observation: Analyze the scatter plot to observe any relationship or pattern between the two variables.

PROGRAM

```
import pandas as pd
```

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

```
# Step 1: Create a dictionary with at least five keys and lists of at least ten values each
```

```
data = {
```

```
    'A': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
```

```
    'B': [5, 15, 25, 35, 45, 55, 65, 75, 85, 95],
```

```
    'C': [12, 24, 36, 48, 60, 72, 84, 96, 108, 120],
```

```
    'D': [8, 16, 24, 32, 40, 48, 56, 64, 72, 80],
```

```
    'E': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
}  
  
# Step 2: Convert dictionary to Pandas DataFrame  
  
df = pd.DataFrame(data)  
  
# Step 3: Select two columns to observe relationship  
  
column1 = 'A'  
  
column2 = 'B'  
  
# Step 4: Plotting using Matplotlib  
  
plt.figure(figsize=(8, 6))  
  
plt.scatter(df[column1], df[column2], color='b', marker='o', label=f'{column1} vs  
{column2}')  
plt.title(f'Scatter Plot of {column1} vs {column2}')  
plt.xlabel(column1)  
  
plt.ylabel(column2)  
  
plt.legend()  
  
plt.grid(True)  
  
plt.show()
```

Output:

The scatter plot will display a graph where values from column 'A' are plotted against values from column 'B'. Each point represents a pair of values from the two columns, showing their relationship visually.

VIVA QUESTIONS

1. How do you select specific columns from a Pandas DataFrame, and what methods would you use to ensure data integrity when working with selected columns?
 2. What is the purpose of a scatter plot in data visualization, and how does it differ from other types of plots?
 3. How to create a scatter plot using Matplotlib, and what parameters are essential for plotting two variables against each other?
 4. How would you interpret the relationship between two variables based on a scatter plot generated from a Pandas DataFrame?
 5. What are some advantages of using Matplotlib for data visualization, and how can you customize plots to improve readability and understanding?
-