SGOU
Higher Education for All

PYTHON PROGRAMMING LAB

COURSE CODE: M25CA02PC

Master of Computer Applications

Practical Core Course

Self Learning Material

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

SGOU

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Python Programming Lab

Course Code: M25CA02PC

Semester - I

SGOU

Practical Core Course
Postgraduate Programme
Master of Computer Applications
Self Learning Material

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

PYTHON PROGRAMMING LAB

Course Code: M25CA02PC
Semester- I
Practical Core Course
Master of Computer Applications

Academic Committee

Prof. (Dr.) Sabu M.K.
Dr. G. Santhoshkumar T
Prof. (Dr.) Vinod Chandra S.S.
Dr. Vinod P.
Dr.Lajish V.L.
Sreekanth M.S.
Dr. Vivek P.
Dr. Arun K.S.
Dr. Abdul Jebbar P.

Development of the Content

Shamin S.
Sreerekha V. K.
Dr. Kanitha D. K.
Greeshma P. P.
Sub Priya Laxhmi S. B. N.
Aswathy V.S.
Anjitha A.V.

Review and Edit

Dr. Sabeena K.

Proofreading

Dr. Deepika M.P.

Scrutiny

Shamin S.
Sreerekha V. K.
Dr. Kanitha D. K.
Greeshma P. P.
Sub Priya Laxhmi S. B. N.
Aswathy V. S.
Anjitha A. V.

Cover Design

Jobin J.

Co-ordination

Dr. Gopakumar C.
Head, School of ICS

Scan this QR Code for reading the SLM on a digital device.

Edition
March 2026

Copyright
© Sreenarayanaguru Open University

SGOU

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from Sreenarayanaguru Open University. Printed and published on behalf of Sreenarayanaguru Open University by Registrar, SGOU, Kollam.

www.sgou.ac.m

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear Learner,

It gives me immense pleasure and a deep sense of pride to warmly welcome you to Sreenarayanaguru Open University—a vibrant and progressive institution committed to transforming lives through inclusive, flexible, and high-quality education.

Established in September 2020 as a forward-looking initiative of the Government of Kerala, the University stands as a beacon of opportunity for learners seeking to advance their academic and professional aspirations through the open and distance learning mode. Guided by our foundational principle that "access and quality define equity," we are steadfast in our mission to democratize education while upholding uncompromising academic standards.

Our university is inspired by the timeless vision and philosophy of Sree Narayana Guru, whose ideals of knowledge, equality, and social transformation continue to guide our academic journey. His enduring legacy instils in us the responsibility to create an educational environment that empowers individuals, nurtures critical thinking, and contributes meaningfully to society.

Understanding the dynamic needs of contemporary learners, we have adopted a robust and learner-centric blended learning model, seamlessly integrating Self-Learning Materials, Academic Counselling, and Advanced Digital Learning Platforms. This holistic approach ensures flexibility without sacrificing academic depth, enabling you to learn at your own pace while remaining meaningfully connected to a vibrant academic ecosystem.

The Master of Computer Applications (MCA) programme you are embarking upon is carefully designed to position you at the forefront of the digital revolution. It uniquely blends strong theoretical foundations with practical, industry-oriented competencies. The curriculum emphasizes algorithmic thinking, system design, programming, database management, networking, and emerging technologies such as artificial intelligence, data science, and cloud computing.
What sets this programme apart is its forward-thinking design, which offers:

- Opportunities for skill enhancement aligned with industry needs
- Multidisciplinary learning pathways for broader intellectual development
- Exposure to innovative and emerging technology domains
- Multiple specialization options tailored to evolving career landscapes
- A strong focus on employability, entrepreneurship, and global career readiness
-

Our Self-Learning Materials are meticulously developed by experts, enriched with contemporary case studies, real-world applications, and practical insights to ensure clarity, engagement, and relevance. We are committed to equipping you not just with knowledge, but with the confidence and competence to excel in a competitive global environment.

At Sreenarayanaguru Open University, you are never alone in your learning journey. Our dedicated learner support system is designed to provide continuous academic guidance, timely assistance, and effective grievance redressal. We encourage you to actively engage with us, share your concerns, and make the most of the resources and support available to you.

As you begin this important phase of your academic journey, I urge you to embrace learning with curiosity, discipline, and determination. The world of technology is ever-evolving, and your willingness to adapt, innovate, and grow will define your success.

Remember, this is not just a programme—it is a pathway to transforming your future.
I wish you a fulfilling learning experience and a successful, inspiring career ahead.

Warm regards,

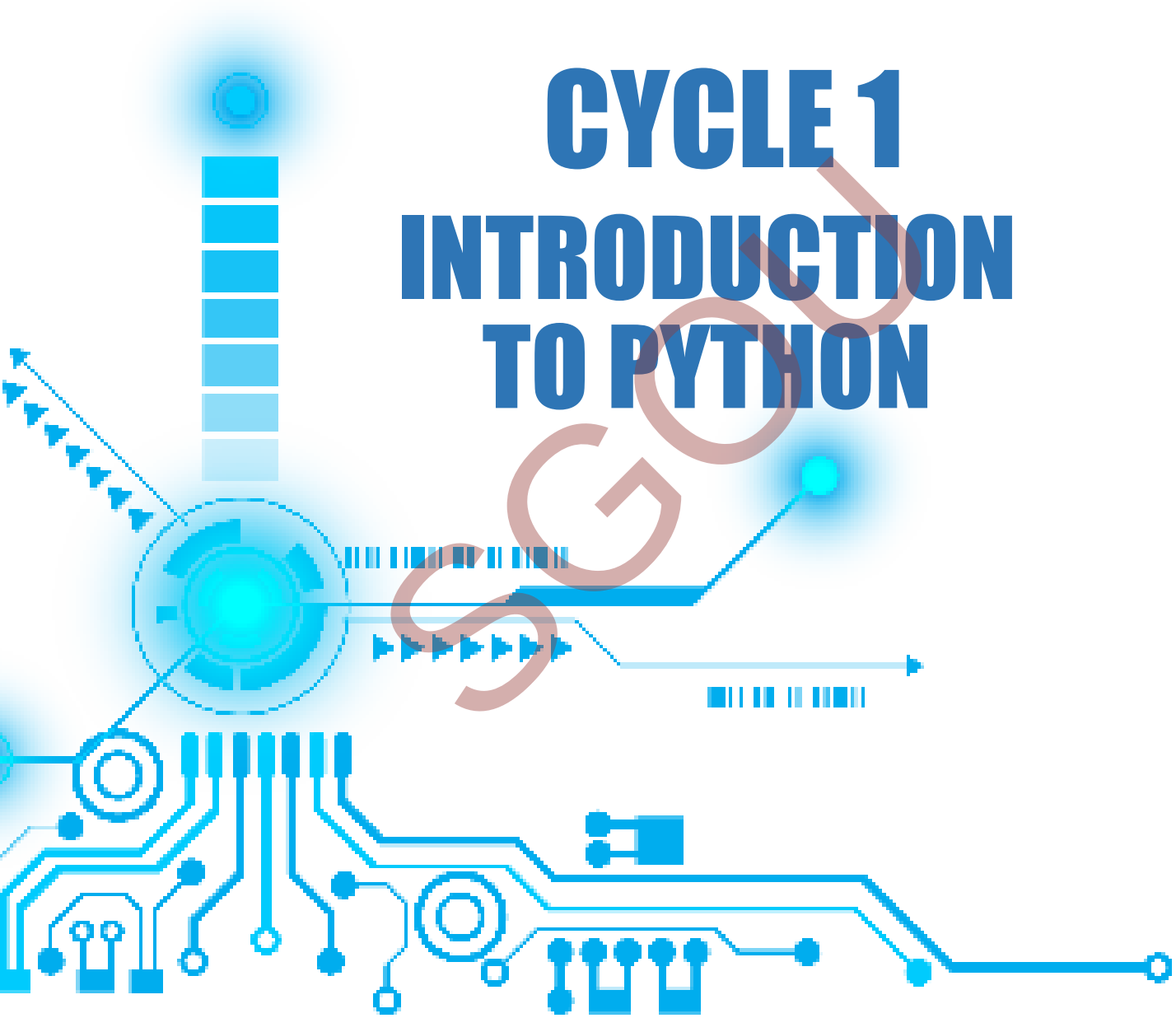
Prof. (Dr.) Jagathy Raj V. P.
Vice Chancellor

SGOU

Contents

SGOU

CYCLE 1
INTRODUCTION TO PYTHON



SGOU

Introduction

Python is a versatile, high-level programming language widely used in software development, automation, web applications, artificial intelligence, data analysis, and system programming. Its simplicity and readability make it an ideal choice for MCA students to strengthen their programming and problem-solving skills. Python is platform-independent, running on Windows, Linux, and macOS, and offers extensive support through libraries and frameworks for different domains such as databases, networking, operating systems, and cloud computing.

In Computer Science, Python is particularly valuable for application development, system administration, network security, and automation. Its integration capabilities with C, C++, Java, and databases make it a strong choice for real-world enterprise projects. The language also supports concurrency and multi-threading, enabling efficient system-level programming. For MCA students, learning Python builds a foundation for developing efficient algorithms, managing data, and working with modern technologies in both academic research and professional practice.

Objectives of the Python Lab

By the end of this lab, students will be able to:

1. Learn the fundamental Python syntax, data types, operators, and execution methods to build a strong programming foundation.
2. Work with data structures such as lists, tuples, sets, and dictionaries to manage and manipulate information efficiently.
3. Implement control structures, functions, and object-oriented programming concepts for structured software development.
4. Perform file handling and exception management to build reliable and error-resistant applications.
5. Connect Python with databases to perform CRUD operations and integrate them with real-world applications.
6. Gain exposure to libraries such as NumPy, Pandas, and Matplotlib for data handling and visualization.
7. Apply Python to develop system-level, web-based, and automation programs relevant to the MCA curriculum and industry needs.

SGOU

Tools and Software Used for Python Programming

Integrated Development Environments (IDEs)

- **PyCharm** – Full-featured IDE for large-scale software development.
- **Visual Studio Code (VS Code)** – Lightweight, extensible editor suitable for MCA projects.
- **Jupyter Notebook** – Useful for demonstration and visualization of Python code.
- **Spyder** – Helpful for debugging and academic coding exercises.
- **Google Colab** – Cloud-based environment, ideal for collaborative programming.

Package Management Tools

- **pip** – Default package manager for installing libraries.
- **conda** – Environment manager useful for advanced library management.

Version Control Tools

- **Git** – Track and manage code changes.
- **GitHub/GitLab** – Platforms for hosting MCA project repositories.

Debugging and Testing Tools

- **pdb** – Built-in Python debugger.
- **pytest** – Framework for testing Python applications.

Scope of the Lab

This Python lab prepares MCA students for a wide range of applications in computer science, including software development, database management, operating system-level programming, and automation. Students will learn to design efficient programs using loops, functions, and classes. The lab also introduces database connectivity, exception handling, and visualization techniques, ensuring a holistic understanding of Python.

Through hands-on practice, students will gain the ability to:

- Write optimized code for solving computational problems.
- Apply object-oriented principles in software development.
- Work with external libraries to handle data and build small projects.
- Connect Python with databases and APIs for real-world applications.

SGOU

Expectations from Students

- Actively participate in lab sessions and complete coding exercises.
- Submit assignments and reports on time with proper documentation.
- Collaborate with peers during group projects to strengthen teamwork.
- Explore beyond the given exercises to apply Python in real-world scenarios.
- Maintain discipline, follow ethical programming practices, and avoid plagiarism.

Lab Guidelines

1. Read the experiment objectives and instructions before coding.
2. Maintain a clean and organized workspace in the lab.
3. Use approved IDEs and tools as recommended by the instructor.
4. Complete and submit all assignments within the deadlines.
5. Write original code and follow professional coding practices.
6. Save and back up all work regularly to avoid data loss.
7. Explore advanced concepts beyond the lab syllabus to strengthen knowledge.
8. Follow academic integrity by avoiding copying or plagiarism.

Steps to Install and Configure Python

Step 1: Download Python

- Visit https://www.python.org.
- Download the latest stable version for Windows, Linux, or macOS.

Step 2: Install Python

- Run the installer and check "**Add Python to PATH**"

  Verify installation using: `python --version`

Step 3: Alternative Installation (For Projects)

- Install the **Anaconda Distribution** for bundled Python, libraries, and Jupyter Notebook.

SGOU

Experiment 1

Data Types, Variables and Operators

Objectives

- To understand and identify different data types in Python, including integers, floats, strings, and Boolean values.
- To learn how to declare, assign, and manipulate variables effectively.
- To gain proficiency in using various operators such as arithmetic, relational, logical, assignment, and bitwise operators.
- To apply Python operators and variables in practical coding exercises for computation and data manipulation.
- To develop the ability to write simple, error-free Python programs using fundamental concepts.

Theory

The fundamental concepts of Python programming are practiced in this experiment, focusing on data types, variables, and operators. Data types define the kind of data that can be stored and manipulated, such as integers, floating-point numbers, strings, and Boolean values. Variables act as named storage containers that hold data, allowing programmers to perform operations and reuse values efficiently. Operators are symbols used to perform computations and comparisons on data, including arithmetic, relational, logical, assignment, and bitwise operations. Through this experiment, learners gain practical experience in declaring variables, assigning values, and applying operators to solve problems, which forms the foundation for writing more complex Python programs. By understanding these core concepts, students develop the skills necessary to handle data effectively and build a strong base for further study in Python programming.

SGOU

1.1.1 Datatypes in Python

Python supports the following built-in data types:

1. Numeric Types: int, float, complex
2. Sequence Types: list, tuple, range, string
3. Mapping Type: dictionary
4. Set Types: set, frozenset
5. Boolean Type: bool
6. Binary Types: bytes, bytearray, memoryview

1.1.2 Variables

- Variables are used to store values.
- No explicit declaration is required; the type is decided at runtime.

Example :

```
x = 10              # integer
y = 3.14            # float
name = "Sam"        # string
```

1.1.3 Operators in Python

- Arithmetic Operators: +, -, *, /, %, //, **
- Relational Operators: <, >, ==, !=, <=, >=
- Logical Operators: and, or, not
- Assignment Operators: =, +=, -=, *=, /=, etc.
- Bitwise Operators: &, |, ^, <<, >>, ~
- Membership Operators: in, not in
- Identity Operators: is, is not

1.1.4 Algorithm for Demonstrating Data Types and Operators

- Start the program.
- Declare and initialize variables with different datatypes.
- Perform operations using different operators.
- Print the results.
- End the program.

1.1.5 Sample Programs

Example 1 : Working with Datatype and Variables

```
a = 10        # integer
b = 5.5       # float
c = "Python"  # string
d = True      # boolean
```

SGOU

```
print("Value of a:", a, "Type:", type(a))
print("Value of b:", b, "Type:", type(b))
print("Value of c:", c, "Type:", type(c))
print("Value of d:", d, "Type:", type(d))
```

Output

Value of a: 10 Type: <class 'int'>

Value of b: 5.5 Type: <class 'float'>

Value of c: Python Type: <class 'str'>

Value of d: True Type: <class 'bool'>

Example 2 : Demonstrating operators

Input

```
x = 15
y = 4
```

Arithmetic operators

```
print("x + y =", x + y)
print("x - y =", x - y)
print("x * y =", x * y)
print("x / y =", x / y)
print("x % y =", x % y)
```

Output

x + y = 19

x - y = 11

x * y = 60

x / y = 3.75

x % y = 3

Relational operators

```
print("x > y:", x > y)
print("x == y:", x == y)
```

Output

x > y: True

x == y: False

Logical operators

```
print("(x > 10) and (y < 5):", (x > 10) and (y < 5))
```

Output

(x > 10) and (y < 5): True

SGOU

Assignment Operators

```
# Assigning a value to a variable using assignment operator
    x = 10
    print(x)
```

Output

```
    10
```

Example with Compound Assignment Operator

```
# Using compound assignment operator
    x = 5
    x += 3   # Equivalent to x = x + 3
    print(x)
```

Output

```
    8
```

Bitwise Operators

1. Bitwise AND (&)

```
    a = 12   # 1100 in binary
    b = 10   # 1010 in binary

    result = a & b
    print("Bitwise AND:", result)
```

Output

```
    Bitwise AND: 8
```

Explanation:

The result bit is 1 only if both corresponding bits are 1.

2. Bitwise OR (|)

```
    a = 12   # 1100
    b = 10   # 1010
    result = a | b
    print("Bitwise OR:", result)
```

Output

```
    Bitwise OR: 14
```

Explanation:

The result bit is 1 if at least one of the corresponding bits is 1.

SGOU

3. Bitwise XOR (^)

```
a = 12
b = 10
result = a ^ b
print("Bitwise XOR:", result)
```

Output

Bitwise XOR: 6

Explanation:

The result bit is 1 only when the bits are different.

4. Bitwise NOT (~)

```
a = 12
result = ~a
print("Bitwise NOT:", result)
```

Output

Bitwise NOT: -13

Explanation:

This inverts all bits (returns the two's complement in Python).

5. Bitwise Left Shift (<<)

```
a = 5   # 0101
result = a << 1
print("Left Shift:", result)
```

Output

Left Shift: 10

Explanation:

Shifts bits to the left and multiplies the number by 2.

6. Bitwise Right Shift (>>)

```
a = 20  # 10100
result = a >> 2
print("Right Shift:", result)
```

Output

Right Shift: 5

Explanation:

Shifts bits to the right and divides the number by 2.

SGOU

7. Combined Bitwise Operators Example

```
a = 8
b = 3
print("AND:", a & b)
print("OR:", a | b)
print("XOR:", a ^ b)
print("Left Shift:", a << 2)
print("Right Shift:", a >> 1)
```

Output

```
AND: 0
OR: 11
XOR: 11
Left Shift: 32
Right Shift: 4
```

Membership Operators

```
my_list = [1, 2, 3, 4, 5]
print("3 in list:", 3 in my_list)
print("10 in list:", 10 in my_list)
print("7 not in list:", 7 not in my_list)
```

Output

```
3 in list: True
10 in list: False
7 not in list: True
```

Identity Operators

```
a = [1, 2, 3]
b = a
c = [1, 2, 3]
print("a is b:", a is b)        # True, same object
print("a is c:", a is c)        # False, different objects
print("a == c:", a == c)        # True, same values
print("a is not c:", a is not c)
```

Output

```
a is b: True
a is c: False
a == c: True
a is not c: True
```

SGOU

1.1.6 List

A list is an ordered, mutable (changeable) collection of items.

It is defined using square brackets [ ].

It can store mixed data types (int, float, string, etc.).

It Supports indexing, slicing, and iteration.

It Allows duplicate values.

Example

```
my_list = [10, 20, 30, "Python", 3.5]
print(my_list[0])       # Access first element
my_list.append(40)      # Add element
my_list.remove(20)      # Remove element
print(my_list[::-1])    # Reverse list
```

Output

```
10
[40, 3.5, 'Python', 30, 10]
```

1.1.7 Tuple

A **tuple** is an ordered, **immutable** collection of items.

It is defined using **parentheses** ( ).

It is used when data should **not change**.

It is faster than lists due to immutability.

Example

```
my_tuple = (1, 2, 3, 4, "MCA")
print(my_tuple[1])      # Access element
print(len(my_tuple))    # Length of tuple
# my_tuple[0] = 10 → ERROR (immutable)
```

Output

```
2
5
```

1.1.8 Set

A set is an unordered, mutable collection of unique elements.

It is defined using curly braces { }.

It does not allow duplicates.

It is useful for mathematical operations (union, intersection, difference).

SGOU

Example

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print(set1 | set2)   # Union → {1, 2, 3, 4, 5, 6}
print(set1 & set2)   # Intersection → {3, 4}
print(set1 - set2)   # Difference → {1, 2}
```

Output

```
{1, 2, 3, 4, 5, 6}
{3, 4}
{1, 2}
```

1.1.9 Dictionary

A **dictionary** is an unordered collection of **key-value pairs**.

It is defined using **curly braces** `{key: value}`

The keys must be **unique and immutable** (such as string, int or tuple).

The values can be of **any type**.

Example

```
student = {"roll": 101, "name": "Anu", "marks": 90}
print(student["name"])        # Access by key
student["marks"] = 95         # Update value
student["dept"] = "MCA"       # Add new key
del student["roll"]           # Delete key
```

Output

```
Anu
```

1.1.10 String

A **string** is a sequence of characters enclosed in quotes (' ' or " ").

It is Immutable → once created, cannot be changed.

It supports **indexing, slicing, concatenation, searching, and formatting**.

Example

```
text = "Hello MCA"
print(text.upper())       # HELLO MCA
print(text.lower())       # hello mca
print(text[::-1])         # Reverse string
print("MCA" in text)      # Membership check → True
```

SGOU

Output

```
HELLO MCA
hello mca
ACM olleH
True
```

String indexing and slicing example

```
text = "PythonProgramming"
```

Forward indexing

```
print("Forward Indexing:")
print("text[0]  =", text[0])    # First character
print("text[6]  =", text[6])    # Character at index 6
```

Output

```
Forward Indexing:
text[0]  = P
text[6]  = P
```

Reverse indexing

```
print("\nReverse Indexing:")
print("text[-1] =", text[-1])   # Last character
print("text[-5] =", text[-5])   # Fifth character from the end
```

Output

```
Reverse Indexing:
text[-1] = g
text[-5] = m
```

String slicing

```
print("\nString Slicing:")
print("text[0:6]   =", text[0:6])     # From index 0 to 5
print("text[6:17]  =", text[6:17])    # From index 6 to 16
print("text[:6]    =", text[:6])      # From start to index 5
print("text[6:]    =", text[6:])      # From index 6 to end
print("text[::-1]  =", text[::-1])    # Reverse the string
```

Output

```
String Slicing:
text[0:6]   = Python
text[6:17]  = Programming
text[:6]    = Python
```

SGOU

```
text[6:]    = Programming
text[::-1]  = gnimmargorPnohtyP
```

Using f-string

```
name = "Anita"
age = 21
marks = 88.5
print(f"Student Name: {name}")
print(f"Age: {age}")
print(f"Marks: {marks}")
print(f"{name} is {age} years old and scored {marks} marks.")
```

Output

```
Student Name: Anita
Age: 21
Marks: 88.5
Anita is 21 years old and scored 88.5 marks.
```

Using format() method

```
name = "Anita"
age = 21
marks = 88.5
print("Student Name: {}".format(name))
print("Age: {}".format(age))
print("Marks: {}".format(marks))
print("{} is {} years old and scored {} marks.".format(name, age, marks))
```

Output

```
Student Name: Anita
Age: 21
Marks: 88.5
Anita is 21 years old and scored 88.5 marks.
```

1.1.11 Lab Practice Questions

- Write a program to declare variables of type int, float, and string and print their values.
- Write a program to swap two variables without using a third variable.
- Demonstrate the use of **arithmetic operators** with two numbers.
- Write a program to find whether a given number is **even or odd** using the modulus operator.
- Create a program to compare two numbers using **relational operators**.
- Write a program to use logical operators with Boolean expressions.

SGOU

- Write a program to calculate the area of a circle using variables and arithmetic operators.
- Demonstrate the use of membership operators (in, not in) with a string.
- Write a program to input a string and display its length, first character, last character, and the reversed string.
- Write a program to create a list of integers, print the list, and display the sum, maximum, and minimum elements
- Write a program to find the largest, smallest, and sum of elements in a numeric tuple.
- Write a program to remove duplicate elements from a list by converting it into a set.Write a program to create a dictionary with student roll numbers as keys and names as values, then display them.

SGOU

Experiment 2

Control and Looping Statements

Objectives

- Understand conditional control statements (if, if-else, if-elif-else).
- Learn iterative looping constructs (for, while).
- Apply nested conditional statements and loops to solving problems.
- Use loop control statements (break, continue, pass).

Theory

1.2.1 Conditional (Control) Statements

- if statement executes when a condition is true.
- if-else statement executes one block if true and another if false.
- if-elif-else statement allows multiple conditions.
- Nested conditions allow decision making inside other conditions.

1.2.2 Looping Statements

- for loop iterates over a sequence.
- while the loop continues until the condition is false.
- Control statements: break, continue, and pass are used to alter the flow.

1.2.3 Sample Programs

Example 1: Write a program to check whether a number is positive, negative, or zero

```
num = int(input("Enter a number: "))
if num > 0:
    print("Positive")
elif num < 0:
    print("Negative")
else:
    print("Zero")
```

SGOU

Output

```
Enter a number: 4
Positive
Enter a number: -7
Negative
```

Example 2: Write a program to find the largest among three numbers

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))


if a >= b and a >= c:
   print("Largest =", a)
elif b >= a and b >= c:
   print("Largest =", b)
else:
   print("Largest =", c)
```

Output

```
Enter first number: 6
Enter second number: 9
Enter third number: 7
Largest = 9
```

Example 3: Write a program for grade evaluation

```
marks = int(input("Enter your marks: "))
if marks >= 90:
   print("Grade: A+")
elif marks >= 75:
   print("Grade: A")
elif marks >= 60:
   print("Grade: B")
elif marks >= 50:
   print("Grade: C")
else:
   print("Fail")
```

SGOU

Output

```
Enter your marks: 56
Grade: C
```

Example 4: Write a program to check voting eligibility using nested if statements

```
age = int(input("Enter age: "))
if age >= 18:
    nationality = input("Are you an Indian citizen? (yes/no): ")
    if nationality.lower() == "yes":
        print("You are eligible to vote")
    else:
        print("Not eligible: Citizenship required")
else:
    print("Not eligible: Age must be 18 or above")
```

Output

```
Enter age: 45
Are you an Indian citizen? (yes/no): yes
You are eligible to vote
```

Example 5: Write a program to print the multiplication table using a for loop

```
n = int(input("Enter a number: "))
for i in range(1, 11):
    print(n, "x", i, "=", n * i)
```

Output

```
Enter a number: 8
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

SGOU

Example 6: Write a program to find the factorial of a number using a while loop

```
n = int(input("Enter a number: "))
fact = 1
i = 1

while i <= n:
    fact *= i
    i += 1
print("Factorial =", fact)
```

Output

```
Enter a number: 5
Factorial = 120
```

Example 7: Write a program to print even numbers using a for loop with the continue statement

```
for i in range(1, 21):
    if i % 2 != 0:
        continue
    print(i, end=" ")
```

Output

```
2 4 6 8 10 12 14 16 18 20
```

Example 8: Write a program to print the Fibonacci Series using a while loop

```
n = int(input("Enter number of terms: "))

a, b = 0, 1
count = 0
while count < n:
print(a, end=" ")
a, b = b, a + b
count += 1
```

SGOU

Output

```
Enter number of terms: 5
0 1 1 2 3
```

1.2.4 Lab Practice Questions

1. Write a program to check whether a number is even or odd.
2. Write a program to check whether a given year is a leap year.
3. Write a program to display all prime numbers between 1 and 50.
4. Write a program to count the digits of a given number.
5. Write a program to reverse a number using a loop.
6. Write a program to check whether a string is a palindrome.
7. Write a program to print multiplication tables from 1 to 10.
8. Write a program to find the sum of the digits of a number.
9. Write a program to print a pyramid pattern using nested loops.
10. Write a program to calculate the sum of the first n natural numbers.

SGOU

Experiment 3

Implement Functions

Objectives

- To understand the concept of functions and their importance in modular programming.
- To implement different types of user-defined functions in Python.
- To practice comprehensions for compact and efficient code.
- To explore higher-order functions for functional programming.
- To understand and use closures and decorators for extending function behavior.
- To apply lambda functions for quick and anonymous operations.
- To implement iterators and generators for efficient sequence handling.

Theory

Functions are one of the most important building blocks of Python programming. A function is a block of reusable code designed to perform a particular task. Instead of repeating code multiple times, we can place it inside a function and call it whenever required. This improves code reusability, readability, maintainability, and modularity.

In Python, functions can be broadly classified into:

- Built-in functions – Already provided by Python (e.g., len(), print(), sum()).
- User-defined functions – Created by the programmer using the **def** keyword.

In addition, Python provides advanced function concepts such as comprehensions, higher-order functions, closures, decorators, lambda functions, generators, and iterators. Each of these plays a role in writing concise, powerful, and memory-efficient programs.

1.3.1 User-defined Functions

A user-defined function is created using the **def** keyword. It allows programmers to define their own logic inside a named block of code. Functions may or may not accept input parameters, and they may or may not return values.

Syntax

```
def function_name(parameters):
    # statements
    return value
```

SGOU

Types of user-defined functions:

- Function with no arguments and no return value.
- Function with arguments but no return value.
- Function with no arguments but returns a value.
- Function with arguments and return value.

1.3.2 Comprehensions

Comprehensions are a compact way of creating lists, sets, or dictionaries in a single line. They reduce the need for loops and make code cleaner.

Types of comprehensions:

- **List comprehension** → [expression for item in iterable if condition]
- **Set comprehension** → {expression for item in iterable if condition}
- **Dictionary comprehension** → {key: value for item in iterable if condition}

Comprehensions provide a concise way to create new collections by applying an operation to each item in an existing sequence, such as a list or a range of numbers. Instead of using a traditional loop to process each item one by one and add the results to a new list, we can write the entire operation in a single line. This not only makes the code shorter but also easier to read and understand. For instance, if we want to calculate the squares of numbers from 1 to 5, we can use a list comprehension to perform the calculation for each number in the range and automatically collect the results into a new list.

Example :

```
squares = [x*x for x in range(1, 6)]
Output: [1, 4, 9, 16, 25]
```

1.3.3 Higher-order Functions

A **higher-order function** is a function that can either:

- Take another function as an argument, or
- Return a function as its result.

SGOU

Python has several higher-order functions such as:

- map(function, iterable) → Applies a function to all elements of an iterable.
- filter(function, iterable) → Returns elements that satisfy a condition.
- reduce(function, sequence) (from functools) → Applies a rolling computation to reduce the sequence to a single value.

1.3.4 Closures

A closure is a function defined inside another function that remembers the values from its enclosing scope, even after the outer function has finished execution.

Closures are widely used in decorators and callback functions.

1.3.5 Decorators

A decorator is a special type of higher-order function that modifies or extends the behavior of another function without changing its source code.

It is applied using the **@decorator_name** syntax before the function definition. Decorators are heavily used in logging, authentication, performance measurement, and web frameworks like Django and Flask.

1.3.6 Lambda Functions

A lambda function is a small anonymous function without a name, defined using the lambda keyword. They are often used for short operations where defining a full function is unnecessary.

Syntax

```
lambda arguments: expression
```

1.3.7 Iterators

An iterator is an object that can be iterated upon. It implements the methods __iter__() and __next__().

Once all items are exhausted, a **StopIteration** exception is raised.

1.3.8 Generators

A generator is a special type of iterator defined using the **yield** keyword inside a function. Unlike normal functions that return all values at once, a generator yields values one at a time during iteration. This makes them memory-efficient because they do not store the entire sequence in memory. Generators are especially useful for handling large datasets or infinite sequences, where storing all values at once would be impractical.

SGOU

1.3.9 Sample Programs

1. Write a Python program to implement a user-defined function that calculates the factorial of a given number.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n-1)
print("Factorial of 5:", factorial(5))
```

Output

```
Factorial of 5: 120
```

2. Write a Python program to generate the squares of the first five natural numbers using list comprehension.

```
squares = [x**2 for x in range(1, 6)]
print("Squares:", squares)
```

Output

```
Squares: [1, 4, 9, 16, 25]
```

3. Write a Python program to use map() and filter() to double the numbers in a list and extract the even numbers.

```
nums = [1, 2, 3, 4, 5]
doubles = list(map(lambda x: x*2, nums))
evens = list(filter(lambda x: x % 2 == 0, nums))
print("Doubles:", doubles)
print("Even numbers:", evens)
```

SGOU

Output

```
Doubles: [2, 4, 6, 8, 10]
Even numbers: [2, 4]
```

4. Write a Python program to demonstrate closure that multiplies a given number by a fixed multiplier.

```
def multiplier(n):
    def inner(x):
        return x * n
    return inner
times3 = multiplier(3)
print("3 * 5 =", times3(5))
```

Output

```
3 * 5 = 15
```

5. Write a Python program to demonstrate the use of a decorator that executes additional code before and after a function call.

```
def my_decorator(func):
    def wrapper():
        print("Before function execution")
        func()
        print("After function execution")
    return wrapper

@my_decorator
def greet():
    print("Hello, Python!")

greet()
```

Output

```
Before function execution
```

SGOU

```
Hello, Python!
After function execution
```

6. Write a Python program to use a lambda function for adding two numbers.

```
add = lambda a, b: a + b
print("Sum:", add(10, 20))
```

Output

```
Sum: 30
```

7. Write a Python program to implement a generator that yields the Fibonacci sequence up to a given number of terms.

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a+b
for num in fibonacci(5):
    print(num)
```

Output

```
0
1
1
2
3
```

8. Write a Python program to demonstrate iteration using an iterator object created from a list.

```
my_list = [1, 2, 3]
it = iter(my_list)

print(next(it))
print(next(it))
print(next(it))
```

SGOU

Output

```
1
2
3
```

1.3.10 Lab Practice Questions

1. Write a user-defined function to check whether a given number is an Armstrong number.
2. Use list comprehension to generate all even numbers between 1 and 50.
3. Write a Python program to use map() and convert a list of strings to uppercase.
4. Use filter() to extract prime numbers from a given list.
5. Write a program using a closure to calculate the power of a number.
6. Create a decorator that logs the start and end time of a function execution.
7. Use a lambda function to find the maximum of two numbers.
8. Implement a generator to produce multiples of 7 up to 100.
9. Write a program to create a custom iterator that iterates through odd numbers.
10. Combine comprehensions and lambda to generate a dictionary of squares of odd numbers from 1–15.

SGOU

Experiment 4

Classes and Objects

Objectives

- To understand the concept of object-oriented programming in Python through classes and objects.
- To learn how to define classes with attributes (variables) and methods (functions).
- To develop the ability to create and manipulate objects for problem-solving.
- To recognize the principles of encapsulation and abstraction using constructors (__init__) and access specifiers.
- To practice writing modular, reusable, and structured Python programs using classes and objects.

Theory

Python is a high-level, object-oriented programming language. One of its most powerful features is the ability to structure programs around objects, which makes code more modular, reusable, and easier to maintain. The foundation of this approach lies in classes and objects.

1.4.1 Class in Python

A class is a user-defined blueprint or prototype from which objects are created. It groups data (attributes) and functions (methods) into a single unit. A class defines what an object will be, but does not allocate memory until an object is created. Some basic points are:

- Defined using the class keyword.
- May contain variables (attributes) to store data.
- May contain functions (methods) to define behavior.

Syntax:

```
class ClassName:
    # attributes and methods
    def method1(self):
        # code
```

SGOU

1.4.2 Object in Python

An object is an instance of a class. When a class is defined, no memory is allocated until an object is created. Each object has its own copy of attributes but shares the class's methods.

- Objects are created by calling the class like a function.
- Each object can store unique values in its attributes.

Syntax

```
# Creating an object
obj = ClassName()
```

1.4.3 The __init__ Method (Constructor)

In Python, the constructor method __init__ is automatically called when an object is created. It is mainly used to initialize attributes.

Example:

```
class Student:

    def __init__(self, name, roll):
        self.name = name
        self.roll = roll
```

Here, *self* represents the instance (object) of the class and is used to access attributes and methods.

1.4.4 Attributes and Methods

In Python, a class is made up of attributes and methods. Attributes are the variables that hold data inside a class, representing the state or properties of an object. For example, a Car class may have attributes like brand and color to describe each car. Methods, on the other hand, are functions defined inside a class that operate on these attributes or perform specific actions. They define the behavior of the objects created from the class. Together, attributes and methods allow objects to combine both data and functionality in a structured way.

Example:

```
class Student:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll

        def display(self):   # method
        print(f"Name: {self.name}, Roll No: {self.roll}")
```

SGOU

1.4.5 Python Sample Programs

1. Write a Python program to create a Student class with attributes name and roll_number. Add a method to display student details.

Source code:

```
class Student:
    def __init__(self, name, roll_number):
        self.name = name
        self.roll_number = roll_number

    def display(self):
        print(f"Name: {self.name}, Roll No: {self.roll_number}")

# Creating objects
s1 = Student("Anita", 101)
s2 = Student("Rahul", 102)

s1.display()
s2.display()
```

Output:

```
Name: Anita, Roll No: 101
Name: Rahul, Roll No: 102
```

2. Write a Python program to create a *Car class* with attributes *brand* and *color*. Add a method *show_details()* to display car information.

Source code:

```
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

def show_details(self):
        print(f"Car Brand: {self.brand}, Color: {self.color}")

car1 = Car("Toyota", "Red")
car2 = Car("Honda", "Blue")

car1.show_details()
car2.show_details()
```

Output:

```
Car Brand: Toyota, Color: Red
Car Brand: Honda, Color: Blue
```

SGOU

3. Create a *class Rectangle* with attributes *length* and *width*. Add a method to calculate and display the area of the rectangle.

Source Code:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
rect1 = Rectangle(5, 3)
rect2 = Rectangle(7, 2)

print("Area of Rectangle 1:", rect1.area())
print("Area of Rectangle 2:", rect2.area())
```

Output:

```
Area of Rectangle 1: 15
Area of Rectangle 2: 14
```

4. Write a Python program to create an *Employee class* with attributes *name* and *salary*. Add a method to display employee details.

Source Code:

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

def display(self):
        print(f"Employee Name: {self.name}, Salary: {self.salary}")

emp1 = Employee("John", 50000)
emp2 = Employee("Sara", 60000)

emp1.display()
emp2.display()
```

Output:

```
Employee Name: John, Salary: 50000
Employee Name: Sara, Salary: 60000
```

SGOU

1.4.6 Lab Practice Questions

1. Write a Python *class Book* with attributes *title* and *author.* Write a method to display the book details. Create two objects and display their details.

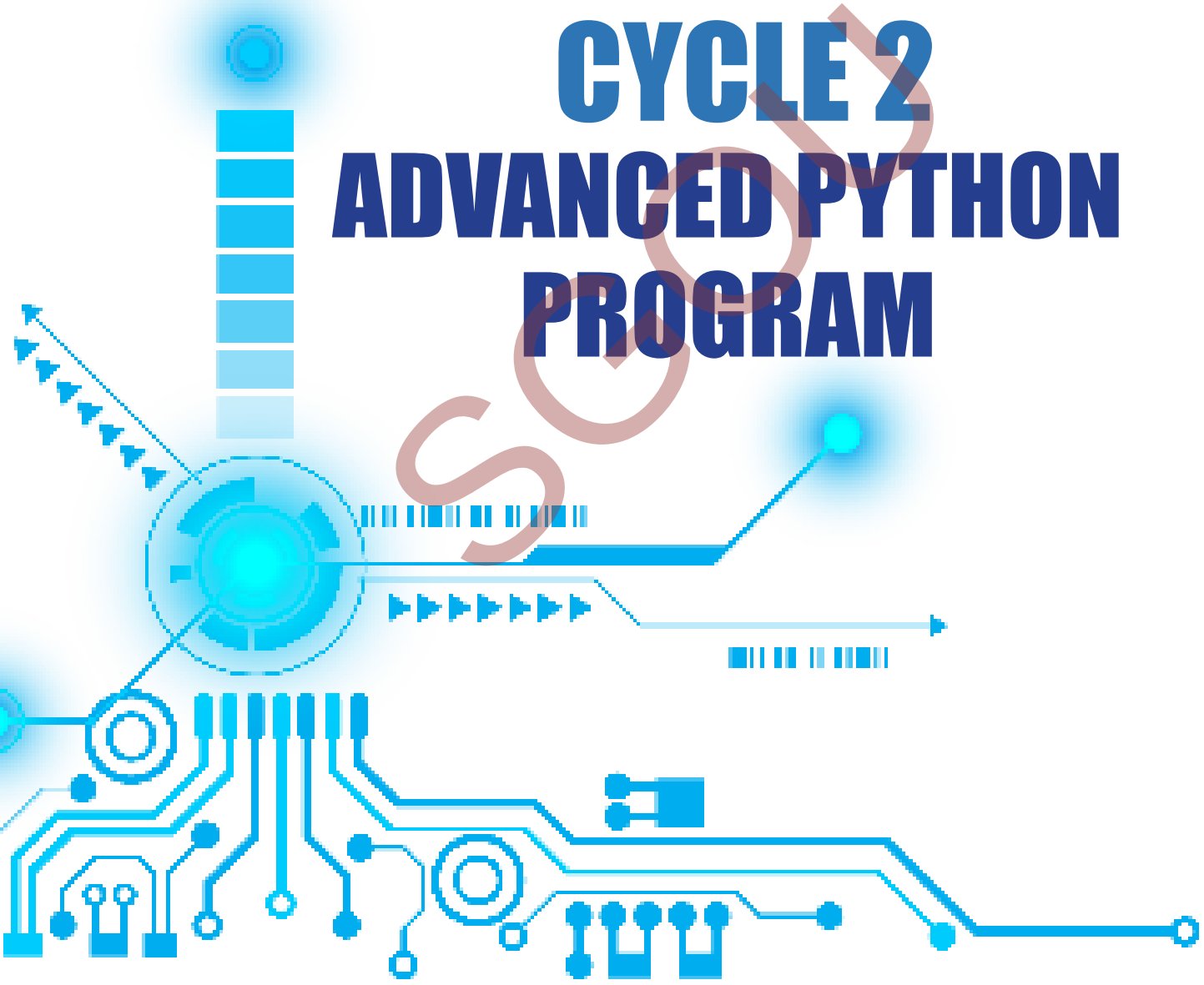
2. Define a *class Circle* with an attribute *radius*. Write methods to calculate the area and circumference of the circle. Create objects for different radii and display the results.

3. Write a Python program to create a *class Employee* with attributes *name* and *salary*. Add a method to display employee information. Create multiple employee objects and display their details.

4. Implement a *class Calculator* with methods for addition, subtraction, multiplication, and division of two numbers. Demonstrate its use by creating an object and calling its methods.

SGOU

CYCLE 2
ADVANCED PYTHON PROGRAM



SGOU

Experiment 1

To develop a program using Inheritance and Polymorphism

Objectives

- To understand the concept of inheritance and how it promotes code reusability in object-oriented programming.
- To implement single and multilevel inheritance in a Python program.
- To explore the concept of method overriding and demonstrate runtime polymorphism.
- To develop programs that utilizes polymorphic behavior through base and derived class interactions.

Theory

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data (attributes) and code (methods). The main principles of OOP include **encapsulation**, **abstraction**, **inheritance**, and **polymorphism**. This lab focuses on two core principles: **inheritance** and **polymorphism**.

2.1.1 Inheritance

Inheritance is the mechanism by which one class (known as the child class or derived class) acquires the properties and behaviors (methods and variables) of another class (known as the parent class or base class). This promotes code reusability and logical hierarchy in class design.

2.1.1.1 Types of Inheritance in Python

- **Single Inheritance**: One child class inherits from one parent class.
- **Multilevel Inheritance**: A class inherits from a derived class, forming a chain of inheretance.
- **Multiple Inheritance**: A class inherits from more than one parent class (supported in Python).

2.1.2 Polymorphism

Polymorphism means "many forms" and refers to the ability of different objects to respond in their own way to the same method call. It enhances flexibility and interoperability in code.

SGOU

2.1.2.1 Types of Polymorphism

- **Compile-Time Polymorphism**: Achieved through **method overloading** (not natively supported in Python).
- **Run-Time Polymorphism**: Achieved through **method overriding**, where a method in a child class overrides a method in the parent class.

2.1.2.2 Method Overriding

When a method in the child class has the same name and parameters as a method in the parent class, the child class's method overrides the parent class's method. This enables the object to call the appropriate method based on its type at runtime, demonstrating dynamic polymorphism.

2.1.3 Python Sample Programs

1. Write a Python program to demonstrate single inheritance where the child class inherits from a parent class. The parent class should have a method speak(), and the child class should add a new method bark().

```
class Animal:
    def speak(self):
        print("Animal speaks")

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

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

Output

```
Animal speaks
Dog barks
```

2. Create a Python program to demonstrate multilevel inheritance. Define three classes: Animal, Mammal, and Dog, where each class has its own method. Create an object of the Dog class and call all the methods inherited from the parent classes as well as its own method.

SGOU

```
class Animal:
    def move(self):
        print("Animal moves")

class Mammal(Animal):
    def feed_milk(self):
        print("Mammal feeds milk")

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

d = Dog()
d.move()
d.feed_milk()
d.bark()
```

Output

```
Animal moves
Mammal feeds milk
Dog barks
```

3. Demonstrate method overriding in Python using polymorphism. Create a base class Shape with an area() method and override it in the derived classes Circle and Square classes.

```
print("Calculating area in Shape")

class Circle(Shape):
    def area(self):
        print("Area of Circle = π × r²")

class Square(Shape):
    def area(self):
        print("Area of Square = side × side")

s = Shape()
c = Circle()
sq = Square()

s.area()
c.area()
sq.area()
```

SGOU

Output

```
Calculating area in Shape
Area of Circle = π × r²
Area of Square = side × side
```

4. Demonstrate multiple inheritance in Python. Create two parent classes Father and Mother, each having a method skills(). Create a child class Child that inherits from both parent classes and overrides the skills() method.

```
class Father:
    def skills(self):
        print("Father: Gardening, Cooking")

class Mother:
    def skills(self):
        print("Mother: Painting, Singing")

class Child(Father, Mother):
    def skills(self):
        print("Child: Sports, Coding")

c = Child()
c.skills()
```

Output

```
Child: Sports, Coding
```

SGOU

2.1.4 Lab Practice Questions

1. Create a multilevel inheritance structure involving a Person class, a Student class that inherits from Person, and a GraduateStudent class that inherits from Student. Demonstrate how attributes and methods are inherited across levels.

2. Demonstrate runtime polymorphism by creating a base class Vehicle with a method start(). Override this method in derived classes Car and Bike and invoke the method using a single base class reference.

3. Illustrate method overriding by creating a superclass BankAccount with a method calculate_interest(), and a subclass SavingsAccount that overrides this method with its own implementation.

Experiment 2

File handling and Exception Handling

Objectives

- To understand the concept of file handling in Python and differentiate between text and binary files.
- To perform basic operations on files such as creating, reading, writing, and appending data, using built-in Python functions.
- To implement file handling techniques in real-world scenarios.

Theory

2.2.1 Introduction to Files

File handling in Python enables us to create, update, read, and delete files stored on the file system. A file is a named location on a disk to store related data. File handling consists of following three steps:

- Open the file.
- Process file i.e., perform read or write operations.
- Close the file.

2.2.1.2 Types of File

1. Text Files- A file whose contents can be viewed using a text editor is called a text file. A text file is simply a sequence of ASCII or Unicode characters.
2. Binary Files-A binary file stores the data in the same way as it is stored in the memory. Executable (.exe) files, MP3 file, image files, word documents are some of the examples of binary files.

   Text Files – .txt, .csv (data is stored in a human-readable format)

   Binary Files – .dat, .bin (data is stored in a machine-readable format)

2.2.1.3 File Operations in Python

a. open() - opens the file

b. read() – reads the entire content of the file

c. readline() – reads one line at a time

SGOU

d. readlines() – returns a list of lines

e. write() – writes string data to the file

f. writelines() – writes a list of strings to the file

g. close() - closes the file

h. File opening using the **with** Statement - Automatically handles closing of the file (with open("sample.txt", "r") as f:)

2.2.1.4 File Opening Modes

Table 2.2.1 File Opening Modes

Mode	Description
"r"	Read mode(default); error if the file does not exist
"w"	Write mode; creates new file or overwrites the existing file
"a"	Append; adds data to existing content
"r+"	Read and Write mode
"w+"	Write and Read mode
"a+"	Append and Readmode
"rb"	Read binary file
"wb"	Write binary file

2.2.1.5 Python Sample Programs

1. Write a Python program to read an entire text file.

```
# Create the file "merge.txt"
    def file_read(fname):
            txt = open(fname)
            print(txt.read())

    file_read('merge.txt')
```

Output

```
----------------------
Python File Handling
>>
```

SGOU

2. Write a Python program to read the contents of a text file named merge.txt and count the following:

   - the total number of uppercase letters.
   - the total number of lowercase letters.
   - the total number of digits

```
# Create the file "merge.txt"
def program1():
    with open("merge.txt","r") as f1:
      data=f1.read()
    cnt_ucase =0
    cnt_lcase=0
    cnt_digits=0
    for ch in data:
        if ch.islower():
            cnt_lcase+=1
        if ch.isupper():
            cnt_ucase+=1
        if ch.isdigit():
            cnt_digits+=1

    print("Total Number of UpperCase letters are:",cnt_ucase)
    print("Total Number of LowerCase letters are:",cnt_lcase)
    print("Total Number of  digits are:",cnt_digits)
program1()
```

Output

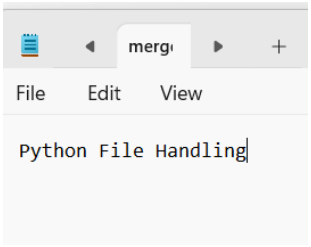
```
========== RESTART: D:\p1
Total Number of Upper Case letters are: 9
Total Number of Lower Case letters are: 10
Total Number of  digits are: 5
```

SGOU

3. Write a Python program that takes a text file as input and returns the number of words in a given text file.

```
def count_words(filepath):
    with open(filepath) as f:
        data = f.read()
        data.replace(",", " ")
        return len(data.split(" "))
print(count_words("merge.txt"))
```

Sample input



Output

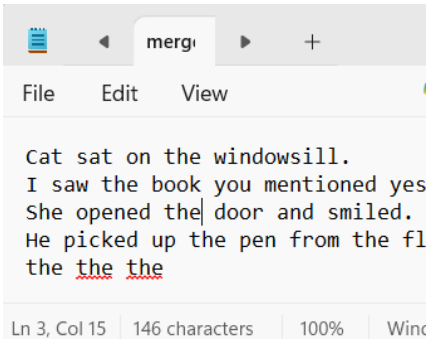
```
========
3
>>
```

4. Write a Python program to read the contents of a text file named `merge.txt` and count how many times the word **"the"** appears in the file.

```
def count_the_word():
    with open("merge.txt", "r") as f:
        content = f.read()
    # Convert all text to lowercase
    content = content.lower()
    # Split into words
    words = content.split()
    # Count the word 'the'
    count = words.count("the")
    print("The word 'the' appears", count, "times.")
# Call the function
count_the_word()
```

SGOU

Sample input



Output

```
The word 'the' appears 8 times.
>>
```

2.2.2 Exception handling

In Python, exception handling is a mechanism to handle errors that occur during runtime. Without proper exception handling, a program may terminate unexpectedly when an error occurs. Exception handling allows the program to continue or exit in a controlled manner. An exception is an error that occurs during the execution of a program and disrupts the normal flow of instructions.

2.2.2.1 Common Exceptions in Python

Table 2.2.2 Common Exceptions

Exception	Description
ZeroDivisionError	Raised when division by zero occurs
ValueError	Raised when a function receives an invalid value
IndexError	Raised when a list index is out of range
KeyError	Raised when a dictionary key is not found
FileNotFoundError	Raised when trying to open a file that does not exist
TypeError	Raised when an operation or function is applied to an inappropriate data type

Syntax

```
try:
    # Code that might raise an exception
except SomeException:
    # Code to handle the exception
```

SGOU

```
else:
    # Code to run if no exception occurs
finally:
    # Code to run regardless of whether an exception occurs
```

- try Block - test a block of code for errors.
- except Block- to handle the error or exception
- else Block: is optional and if included, must follow all except blocks.
- finally Block: finally block always runs, regardless of whether an exception occurred or not.

2.2.2.2 General Built-in Python Exceptions

- EOFError-Raised when one of the built-in functions (input( )) encounters an end-of-file condition (EOF) without reading any data.
- IO Error-Raised when an I/O operation fails due to an I/O-related reason.
- NameError-Raised when a local or global name is not found.
- ImportError-Raised when an import statement fails to find the module definition or when a *from ... import* fails to find the specified name.

2.2.2.3 Python Sample Programs

1. Write a Python function to perform the division of two numbers entered by the user. Use exception handling to manage invalid inputs and division by zero errors.

```
def divide():
    try:
        num1 = int(input("Enter first number: "))
        num2 = int(input("Enter second number: "))
        result = num1 / num2
        print("Result is:", result)
    except Exception as e:
        print("An error occurred:", e)

divide()
```

Output

```
================================ RESTAR
Enter first number: 40
Enter second number: 0
An error occurred: division by zero
>>
```

SGOU

2. Write a Python program using a function that accepts a list of integers and returns the sum. Handle cases where the input list may contain non-integer values.

```
def sum_list_elements(lst):
    try:
        total = sum(lst)
        print("Sum:", total)
    except Exception as e:
        print("Error:", e)

data = [10, 20, 'a', 30]
sum_list_elements(data)
```

Output

```
========== RESTART: D:\p1.py ==========
Error: unsupported operand type(s) for +: 'int' and 'str'
>>
```

3. Create a function to calculate the square root of a number. Use exception handling to manage negative values and non-numeric inputs.

```
import math

def safe_sqrt():
    try:
        num = float(input("Enter a number: "))
        if num < 0:
            raise ValueError("Cannot compute square root of a negative number.")
        print("Square Root:", math.sqrt(num))
    except Exception as e:
        print("Error:", e)

safe_sqrt()
```

Output

```
========== RESTART: D:\p1.py ==========
Enter a number: -3
Error: Cannot compute square root of a negative number.
>>
```

SGOU

4. Write a Python program that accepts student marks and handles exceptions for invalid input data

```
def get_student_marks():
    try:
        marks = float(input("Enter student marks (0 - 100): "))
              if marks < 0 or marks > 100:
            raise ValueError("Marks must be between 0 and 100.")
             print(f"Marks entered: {marks}")
        except ValueError as ve:
        print("Invalid input:", ve)
    except Exception as e:
        print("An unexpected error occurred:", e)
# Call the function
get_student_marks()
```

2.2.3 Lab Practice Questions

1. Write a Python program to read a text file and count the following:
   - ♦ the total number of lines
   - ♦ the total number of words
   - ♦ the total number of characters
2. Write a Python program to read a text file and count how many times the word **"a"** appears in the file.
3. Write a Python program that accepts the age of a person from the user. Handle exceptions to ensure that the input is a valid positive integer. If the user enters non-numeric data or a negative number, display an appropriate error message.
4. Write a Python program that accepts an exam score from the user. The score must be a number between 0 and 100 (inclusive). Handle exceptions if the score is negative or greater than 100.

SGOU

Experiment 3

Develop program to connect database with python

Objectives

- To understand how to connect Python with a MySQL database using mysql-connector.
- To learn the process of creating databases, creating tables, inserting records, and displaying data using Python.
- To demonstrate database connectivity operations step by step.

Theory

2.3.1 Python MySQL Connectivity

Connecting a real-world application to a database is essential for managing data. This database connectivity allows the application to seamlessly manipulate, retrieve, and update information, making sure the application and the database can communicate and interact effectively.

Python can be used in database applications. One of the most popular databases is MySQL. Basically the process of transfer data between python programs and MySQL databases is known as Python Database Connectivity. When an SQL query is executed, the data that is retrieved from the database forms a result set. This result set is a logical collection of records that is then made available for use by the application program. There are a few steps you have to follow to perform Python Database Connectivity. These steps are as follow:

1. Import the required packages
2. Establish a connection
3. Execute SQL command
4. Process as per the requirements

2.3.2 Steps for Python MySQL Connectivity

Step 1: Install **Python** on your system

Step 2: Install **MySQL Server** (e.g., MySQL Workbench or MySQL Community Server)

Step 3: Open **Command Prompt** and ensure you have an **active internet connection**

Step 4: Type and execute the command:

SGOU

pip install mysql-connector-python

Step 5: Open **Python IDLE** or your code editor

Step 6: Start writing your Python program and use it.

import mysql.connector

Step7: Establish a connection

```
import mysql.connector
conn = mysql.connector.connect(
host="localhost",
user="root",
password="system",
database="your_db_name"
)
```

Step 8 : Create a cursor object and execute SQL commands.

Step 9 : Use commit() to save changes in the database.

Step10 : Close the connection after operations.

2.3.3 Python Sample Programs

1. Write a MySQL connectivity program in Python

   - Create a database school
   - Create a table students with the specifications - ROLLNO integer, STNAME character(10) in MySQL and perform the following operations:

     1. Insert two records in it
     2. Display the contents of the table

```
import mysql.connector
conn = mysql.connector.connect(host="localhost", user="root", password="system")

cursor = conn.cursor()
cursor.execute("CREATE DATABASE school")
print("Database 'school' created ")
conn.database = "school"
```

SGOU

```
    cursor.execute("CREATE TABLE students (ROLLNO INT, STNAME CHAR(10)")
    print("Table 'students' created.")

    cursor.execute("INSERT INTO students (ROLLNO, STNAME) VALUES (101, 'Alice')")
    cursor.execute("INSERT INTO students (ROLLNO, STNAME) VALUES (102, 'Bob')")
    conn.commit()
    print("2 records inserted into 'students' table.")

    cursor.execute("SELECT * FROM students")
    records = cursor.fetchall()

    print("\nContents of 'students' table:")
    for row in records:
       print("ROLLNO:", row[0], "STNAME:", row[1])

    cursor.close()
   conn.close()
```

Output

```
Database 'school' created.
Table 'students' created.
2 records inserted into 'students' table.

Contents of 'students' table:
ROLLNO: 101 STNAME: Alice
ROLLNO: 102 STNAME: Bob
```

3. Write a Python program using MySQL connectivity to:

   - ♦ Create a database named college
   - ♦ Create a table named faculty with the following specifications:

     FID (integer), FNAME (character of length 20)
   - ♦ Insert two records into the table
   - ♦ Display the contents of the table

SGOU

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="system"
)
cursor = conn.cursor()

cursor.execute("CREATE DATABASE college")
print("Database 'college' created .")

conn.database = "college"

# Step 4: Create Table 'faculty'
cursor.execute("
CREATE TABLE faculty (
    FID INT,
    FNAME CHAR(20)
)
")
print("Table 'faculty' created.")

cursor.execute("INSERT INTO faculty (FID, FNAME) VALUES (201, 'Dr. Smith')")
cursor.execute("INSERT INTO faculty (FID, FNAME) VALUES (202, 'Prof. Alice')")
conn.commit()
print("2 records inserted into 'faculty' table.")

cursor.execute("SELECT * FROM faculty")
rows = cursor.fetchall()

print("\nContents of 'faculty' table:")
for row in rows:
    print("FID:", row[0], "FNAME:", row[1])
cursor.close()
conn.close()
```

SGOU

Output

```
Database 'college' created.
Table 'faculty' created .
2 records inserted into 'faculty' table.

Contents of 'faculty' table:
FID: 201 FNAME: Dr. Smith
FID: 202 FNAME: Prof. Alice
```

3. Write a Python program using MySQL connectivity to:

- Create a database named library.
- Create a table named books with fields:

  BOOK_ID (integer), TITLE (char(30)), AUTHOR (char(20))
- Insert at least two book records into the table.
- Display all the book details from the table.

```
import mysql.connector

conn = mysql.connector.connect(host="localhost", user="root", password="system")
cursor = conn.cursor()

cursor.execute("CREATE DATABASE library")
print("Database 'library' created.")

conn.database = "library"

cursor.execute("CREATE TABLE books (BOOK_ID INT, TITLE CHAR(30), AUTHOR CHAR(20))")
print("Table 'books' created.")

cursor.execute("INSERT INTO books (BOOK_ID, TITLE, AUTHOR) VALUES (301, 'Python Basics', 'John Doe')")

cursor.execute("INSERT INTO books (BOOK_ID, TITLE, AUTHOR) VALUES (302, 'Data Science', 'Jane Smith')")
```

SGOU

```
conn.commit()
print("2 records inserted into 'books' table.")

cursor.execute("SELECT * FROM books")
rows = cursor.fetchall()

print("\nContents of 'books' table:")
for row in rows:
    print("BOOK_ID:", row[0], "TITLE:", row[1], "AUTHOR:", row[2])

cursor.close()
conn.close()
```

Output

```
Database 'library' created.
Table 'books' created.
2 records inserted into 'books' table.

Contents of 'books' table:
BOOK_ID: 301 TITLE: Python Basics AUTHOR: John Doe
BOOK_ID: 302 TITLE: Data Science AUTHOR: Jane Smith
```

2.3.4 Lab Practice Questions

1. Write a Python program using MySQL connectivity to:
   - ♦ Create a database named store
   - ♦ Create a table named products with the following fields:
     PID (integer), PNAME (char(30)), PRICE (float)
   - ♦ Insert two product records
   - ♦ Display all product details

2. Write a Python program using MySQL connectivity to:
   - ♦ Create a database named hospital
   - ♦ Create a table named patients with the following structure:
     PATIENT_ID (integer), NAME (char(30)) DISEASE (char(30))
   - ♦ Insert two patient records
   - ♦ Display all patient records

SGOU

Experiment 4

Data Processing and Visualization in Python Using NumPy, Pandas and Matplotlib.

Objectives

- Understand the purpose of NumPy, Pandas, and Matplotlib in data analysis and visualization.
- Apply NumPy operations to create, manipulate, and analyze arrays in Python.
- Analyze datasets using Pandas by performing data cleaning, filtering, and aggregation.
- Develop different types of charts and visualizations using Matplotlib to represent data effectively.
- Integrate NumPy, Pandas, and Matplotlib to build simple data processing and visualization projects.

Theory

In Python, data processing and visualization can be efficiently performed using the powerful libraries NumPy, Pandas, and Matplotlib. NumPy provides high-performance multidimensional arrays and mathematical functions, enabling fast computation and numerical analysis. Pandas offers data structures like Series and DataFrame for easy manipulation, cleaning, and analysis of structured data, including reading from and writing to files such as CSV or Excel. Matplotlib is used for visualizing data through a wide variety of plots, including line graphs, bar charts, histograms, and scatter plots, helping to interpret trends and patterns effectively. Together, these libraries allow programmers to load data from files, process and analyze it efficiently, and create meaningful visualizations to support data-driven decision-making.

2.4.1 NumPy in Python

NumPy (Numerical Python) is a Python library for scientific computing. It provides multidimensional arrays (called ndarray) that store elements of the same type efficiently. NumPy enables fast mathematical operations, array manipulations, and linear algebra computations. It is widely used in data processing, analysis, and serves as a foundation for libraries like Pandas and Matplotlib. NumPy operations are vectorized, making them faster and more efficient than standard Python lists for large datasets.

SGOU

Requirements:

- Python 3.x must be installed
- NumPy library installed (pip install numpy)
- A Text editor or IDE (e.g., VS Code, PyCharm, or Jupyter Notebook)

Built-In Numpy Functions to Create Arrays

np.zeros(shape)Creates an array filled with zeros.
np.ones(shape)Creates an array filled with ones.
np.full(shape, fill_value)Creates an array filled with a specified value.
np.arange(start, stop, step)Creates an array with evenly spaced values.
np.linspace(start, stop, num)Creates an array with a specified number of evenly spaced values over an interval
np.random.rand(d0, d1, ...) Creates an array with random values

Array Attributes

arr.shape A tuple representing the dimensions of the array
arr.ndim The number of dimensions (axes) of the array.
arr.size The total number of elements in the array.
arr.dtype The data type of the array's elements.

Aggregation Functions

np.sum(arr) Returns the sum of all elements.
np.mean(arr)Returns the mean of all elements.
np.min(arr) Returns the minimum value.
np.max(arr) Returns the maximum value.
np.std(arr) Returns the standard deviation.

2.4.2 Pandas in Python

Pandas is a powerful open-source Python library used for data manipulation, analysis, and cleaning. It provides two main data structures: Series (a one-dimensional labeled array) and DataFrame (a two-dimensional labeled data structure like a table). Pandas makes it easy to handle structured data, such as CSV, Excel, SQL databases, and JSON files. It allows operations like filtering, grouping, merging, reshaping, and handling missing values efficiently. The two main data structures in Pandas are:

SGOU

1. **Series** – A one-dimensional labeled array (like a column in Excel).

A Series can be created from a Python list, a NumPy array, or a dictionary.

- From a list: pd.Series([4, 7, -5, 3])
- From a NumPy array: import numpy as np; pd.Series(np.array([1, 2, 3]))
- From a dictionary: pd.Series({'a': 1, 'b': 2, 'c': 3})

2. **DataFrame** – A two-dimensional labeled data structure with rows and columns (like an Excel sheet or SQL table).

A DataFrame can be created from a dictionary of lists.

- data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
- df = pd.DataFrame(data)

Use pd.read_csv() to load a sample dataset

2.4.3 Matplotlib in Python

Matplotlib is a powerful Python library used for data visualization. It allows you to create a wide range of static, animated, and interactive plots. With Matplotlib, users can convert raw data into easy-to-understand graphs such as line charts, bar charts, histograms, scatter plots, and pie charts. The library is widely used in scientific computing, data analysis, and machine learning to visually represent trends and patterns.

The most commonly used module in Matplotlib is pyplot, which provides functions to create and customize plots easily. A graph typically consists of data points, axes, labels, titles, legends, and colors, all of which can be customized for better understanding.

Matplotlib also integrates well with NumPy and Pandas, making it ideal for handling and visualizing large datasets.

Software Requirements

- Python 3.x must be installed
- Matplotlib Library (pip install matplotlib)
- NumPy and Pandas installed (for dataset handling)
- A Text editor or IDE (e.g., VS Code, PyCharm, or Jupyter Notebook)

SGOU

Table 2.4.1Charts in Matplotlib

Chart Type	Function in Matplotlib	Purpose / Use
Line Chart	plt.plot(x, y)	Displays data trends over time or a sequence
Bar Chart	plt.bar(x, height) (vertical) plt.barh(y, width) (horizontal)	Compares data across categories
Histogram	plt.hist(data, bins)	Shows the frequency distribution of data
Scatter Plot	plt.scatter(x, y)	Shows the relationship between two variables
Pie Chart	plt.pie(values, labels=labels)	Displays data as percentages of a whole
Stack Plot	plt.stackplot(x, y1, y2, …)	Visualizes the cumulative contribution over time
Area Chart	plt.fill_between(x, y)	Shows the magnitude of data filled under a curve
Box Plot	plt.boxplot(data)	Summarizes data distribution with quartiles and outliers
Stem Plot	plt.stem(x, y)	Visualizes discrete data with vertical lines
Subplots	plt.subplot(rows, cols, index)	Allows multiple charts in one figure
3D Plot	ax.plot3D(x, y, z)	Provides 3D visualization of data

Table 2.4.2 Customization of Graphs in Matplotlib

Customization	Function	Purpose	Example
Title	plt.title("Title")	Adds a title to the graph	plt.title("Sales Report")
X-axis Label	plt.xlabel("X-axis")	Adds label to the X-axis	plt.xlabel("Time (days)")

SGOU

Y-axis Label	plt.ylabel("Y-axis")	Adds label to the Y-axis	plt.ylabel("Revenue ($)")
Legends	plt.legend()	Identifies multiple plots on the same graph	plt. legend(["2023","2024"])
Grid	plt.grid(True)	Adds grid lines for readability	plt.grid(color="gray", linestyle="--")
Line Style	linestyle='--', '-.', ':'	Changes line appearance	plt.plot(x,y, linestyle="--")
Line Color	color='r' (or 'blue', '#FF5733')	Sets line color	plt. plot(x,y,color="green")
Line Width	linewidth=2	Adjusts the thickness of line	plt.plot(x,y,linewidth=3)
Markers	marker='o', '*', 's'	Shows data points with symbols	plt.plot(x,y,marker='*')
Marker Size	markersize=10	Adjusts the size of markers	plt.plot (x,y,marker='o', markersize=12)
Text Annotation	plt.text(x,y,"label")	Places custom text on the graph	plt.text(2,20,"Peak Value")
Axis Limits	plt.xlim(), plt.ylim()	Sets the min and max range for axes	plt.xlim(0,10)
Figure Size	plt. figure(figsize=(w,h))	Sets the size of graph	plt.figure(figsize=(8,5))
Background Color	plt. gca().set_ facecolor("lightyellow")	Changes the plot area background	Makes charts more attractive

SGOU

2.4.4 Python Sample Programs

1. Write a Python program using NumPy to:

   a. Create a 1D array with elements [1, 2, 3, 4, 5].

   b. Create a 2D array with elements [[1, 2, 3], [4, 5, 6]].

   c. Create a 2x3 array of zeros and a 3x2 array of ones.

   d. Create an array of integers from 1 to 10 using arange().

   e. Create an array of five evenly spaced values between 0 and 1 using linspace()

   f. Display all the arrays and verify their shapes and contents.

```
import numpy as np

# 1D array
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)

# 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2)
# Array of zeros
zeros = np.zeros((2,3))
print(zeros)
# Array of ones
ones = np.ones((3,2))
print(ones)
# Array with range
arr_range = np.arange(1, 11)
print(arr_range)
# Array with evenly spaced values
arr_linspace = np.linspace(0, 1, 5)
print(arr_linspace)
```

SGOU

Output

```
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]]
[ 1  2  3  4  5  6  7  8  9 10]
[0.   0.25 0.5  0.75 1.  ]
```

2. Write a Python program using NumPy to :

a. Create two 1D arrays a = [1, 2, 3] and b = [4, 5, 6].

b. Perform and display element-wise addition, subtraction, multiplication, and division of the arrays.

c. Calculate and display the square root of array a.

d. Calculate and display the sum of array a and the mean of array b.

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", b / a)
print("Square root of a:", np.sqrt(a))
print("Sum of a:", np.sum(a))
print("Mean of b:", np.mean(b))
```

SGOU

Output

```
Addition: [5 7 9]
Subtraction: [-3 -3 -3]
Multiplication: [ 4 10 18]
Division: [4.  2.5 2. ]
Square root of a: [1.         1.41421356 1.73205081]
Sum of a: 6
Mean of b: 5.0
```

3. Write a Python program using NumPy to:

- Create a 1D array of integers from 1 to 15.
- Display the first element, last element, and fifth element.
- Print elements from index 4 to 9.
- Print every second element of the array.
- Reverse the array and display it.
- Create a 2D array of size 3x5 using the numbers 1 to 15.
- Display the first row, last column, and the sub-array of middle 2x3 elements.

```
import numpy as np

# Create a 1D array of integers from 1 to 15
arr1d = np.arange(1, 16)
print("1D Array:", arr1d)

# Display the first element, last element, and fifth element
print("First element:", arr1d[0])
print("Fifth element:", arr1d[4])
print("Last element:", arr1d[-1])

# Print elements from index 4 to 9
print("Elements from index 4 to 9:", arr1d[4:10])

# Print every second element of the array
print("Every second element:", arr1d[::2])

# Reverse the array and display it
print("Reversed array:", arr1d[::-1])

# Create a 2D array of size 3x5 using numbers 1 to 15
arr2d = arr1d.reshape(3, 5)
print("\n2D Array (3x5):\n", arr2d)

# Display the first row
print("First row:", arr2d[0])

# Display the last column
print("Last column:", arr2d[:, -1])

# Display the sub-array of middle 2x3 elements
print("Middle 2x3 sub-array:\n", arr2d[1:, 1:4])
```

SGOU

Output

```
1D Array: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
First element: 1
Fifth element: 5
Last element: 15
Elements from index 4 to 9: [ 5  6  7  8  9 10]
Every second element: [ 1  3  5  7  9 11 13 15]
Reversed array: [15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]

2D Array (3x5):
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
First row: [1 2 3 4 5]
Last column: [ 5 10 15]
Middle 2x3 sub-array:
 [[ 7  8  9]
 [12 13 14]]
```

4. Write a Python program using Pandas to create a Series from a list of numbers [10, 20, 30, 40, 50]. Display the first three elements using indexing. Perform basic operations: sum, mean, and maximum.

```
import pandas as pd

# Create Series
s = pd.Series([10, 20, 30, 40, 50])
print("Series:\n", s)

# Indexing
print("\nFirst three elements:\n", s[:3])

# Operations
print("\nSum:", s.sum())
print("Mean:", s.mean())
print("Maximum:", s.max())
```

SGOU

Output

```
Series:
 0   10
1   20
2   30
3   40
4   50
dtype: int64

First three elements:
 0   10
1   20
2   30
dtype: int64

Sum: 150
Mean: 30.0
Maximum: 50
```

5. Write a Python program using Pandas to create a DataFrame with student details: Name, Age, and Marks. Display the first two rows using head(). Access the "Marks" column.

```
import pandas as pd

# Create DataFrame
data = {
   'Name': ['Akku', 'vinu', 'sachu', 'David'],
   'Age': [20, 21, 19, 22],
   'Marks': [85, 90, 78, 92]
}

df = pd.DataFrame(data)
print("DataFrame:\n", df)

# Display first two rows
print("\nFirst two rows:\n", df.head(2))

# Access column
print("\nMarks column:\n", df['Marks'])
```

SGOU

Output

```
DataFrame:
     Name  Age  Marks
0   Akku   20    85
1    vinu  21    90
2  sachu   19    78
3  David   22    92

First two rows:
    Name  Age  Marks
0  Akku   20    85
1  vinu   21    90

Marks column:
 0    85
1    90
2    78
3    92
Name: Marks, dtype: int64
```

6. Write a Python program using Pandas to create a DataFrame of employees with columns: ID, Name, Salary, and Department. Display the first two rows. Filter employees with Salary greater than 40,000.

```
import pandas as pd

# Create DataFrame
data = {
    'ID': [101, 102, 103, 104],
    'Name': ['John', 'Anna', 'Mike', 'Sara'],
    'Salary': [35000, 45000, 30000, 50000],
    'Department': ['HR', 'IT', 'Finance', 'IT']
}

df = pd.DataFrame(data)
print("DataFrame:\n", df)

# Display first two rows
print("\nFirst two rows:\n", df.head(2))

# Filter by condition
print("\nEmployees with Salary > 40000:\n", df[df['Salary'] > 40000])
```

SGOU

Output

```
DataFrame:
   ID  Name  Salary Department
0  101  John   35000         HR
1  102  Anna   45000         IT
2  103  Mike   30000    Finance
3  104  Sara   50000         IT

First two rows:
   ID  Name  Salary Department
0  101  John   35000         HR
1  102  Anna   45000         IT

Employees with Salary > 40000:
   ID  Name  Salary Department
1  102  Anna   45000         IT
3  104  Sara   50000         IT
```

7. Write a Python program using Pandas to create a DataFrame of books with Title, Author, and Price. Save the DataFrame to a CSV file named books.csv. Read the file back and display the contents.

```
import pandas as pd

# Create a DataFrame of books
data = {
    'Title': ['Python Basics', 'Data Science Handbook', 'Machine Learning 101', 'AI for Beginners'],
    'Author': ['John Smith', 'Emily Davis', 'Michael Brown', 'Sarah Johnson'],
    'Price': [350, 500, 600, 450]
}

df = pd.DataFrame(data)
print("Original DataFrame:\n", df)

# Save DataFrame to CSV
df.to_csv("books.csv", index=False)
print("\nDataFrame has been saved to 'books.csv'")

# Read the file back
df_read = pd.read_csv("books.csv")
print("\nData read from CSV file:\n", df_read)
```

SGOU

Output

```
Original DataFrame:
                Title          Author  Price
0        Python Basics       John Smith    350
1  Data Science Handbook       Emily Davis    500
2   Machine Learning 101    Michael Brown    600
3     AI for Beginners    Sarah Johnson    450

DataFrame has been saved to 'books.csv'

Data read from CSV file:
                Title          Author  Price
0        Python Basics       John Smith    350
1   Data Science Handbook       Emily Davis    500
2   Machine Learning 101    Michael Brown    600
3  AI for Beginners    Sarah Johnson    450
```

8. Write a Python program using Pandas to create a DataFrame of products with columns: ID, Name, and Price. Add a new column, "Discounted Price" (10% off). Drop the "Price" column

```
import pandas as pd

# Create a DataFrame of products
data = {
   'ID': [101, 102, 103, 104],
   'Name': ['Laptop', 'Smartphone', 'Tablet', 'Headphones'],
   'Price': [50000, 20000, 15000, 3000]
}

df = pd.DataFrame(data)
print("Original DataFrame:\n", df)

# Add a new column 'Discounted Price' (10% off)
df['Discounted Price'] = df['Price'] * 0.9
print("\nDataFrame after adding Discounted Price:\n", df)

# Drop the 'Price' column
df = df.drop('Price', axis=1)
print("\nDataFrame after dropping Price column:\n", df)
```

SGOU

Output

```
Original DataFrame:
    ID       Name  Price
0  101     Laptop  50000
1  102 Smartphone  20000
2  103     Tablet  15000
3  104 Headphones   3000

DataFrame after adding Discounted Price:
    ID       Name  Price  Discounted Price
0  101     Laptop  50000           45000.0
1  102 Smartphone  20000           18000.0
2  103     Tablet  15000           13500.0
3  104 Headphones   3000            2700.0

DataFrame after dropping Price column:
    ID       Name  Discounted Price
0  101     Laptop           45000.0
1  102 Smartphone           18000.0
2  103     Tablet           13500.0
3  104 Headphones            2700.0
```

9. Write a Python program to create a Pandas DataFrame for analyzing the number of Government and Private medical colleges, their total seats, and fees for different states using the dataset (e.g., from www.data.gov.in). Perform the following operations:

   - Change the name of the state **AP** to **Andhra**.
   - Count and display the **non-NaN values** of each column.
   - Count and display the **non-NaN values** of each row.
   - Increase the fees of all colleges by **5%**.
   - Replace all **NaN values with 0**.

SGOU

```
import pandas as pd
import numpy as np

# Sample DataFrame (you can replace this with actual dataset from data.gov.in)
data = {
    "State": ["AP", "TS", "KA", "TN", "MH"],
    "Govt_Colleges": [10, 12, 15, np.nan, 18],
    "Private_Colleges": [20, 25, np.nan, 30, 28],
    "Total_Seats": [3000, 4000, 5000, 4500, np.nan],
    "Fees": [50000, 60000, np.nan, 55000, 65000]
}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# (i) Change the name of the state 'AP' to 'Andhra'
df["State"] = df["State"].replace("AP", "Andhra")
print("\nAfter renaming AP to Andhra:\n", df)

# (ii) Count and Display Non-NaN values of each column
print("\nCount of Non-NaN values in each column:\n", df.count())

# (iii) Count and Display Non-NaN values of each row
print("\nCount of Non-NaN values in each row:\n", df.count(axis=1))

# (iv) Increase the fees of all colleges by 5%
df["Fees"] = df["Fees"] * 1.05
print("\nAfter increasing Fees by 5%:\n", df)

# (v) Replace all NaN values with 0
df = df.fillna(0)
print("\nAfter replacing NaN with 0:\n", df)
```

SGOU

Output

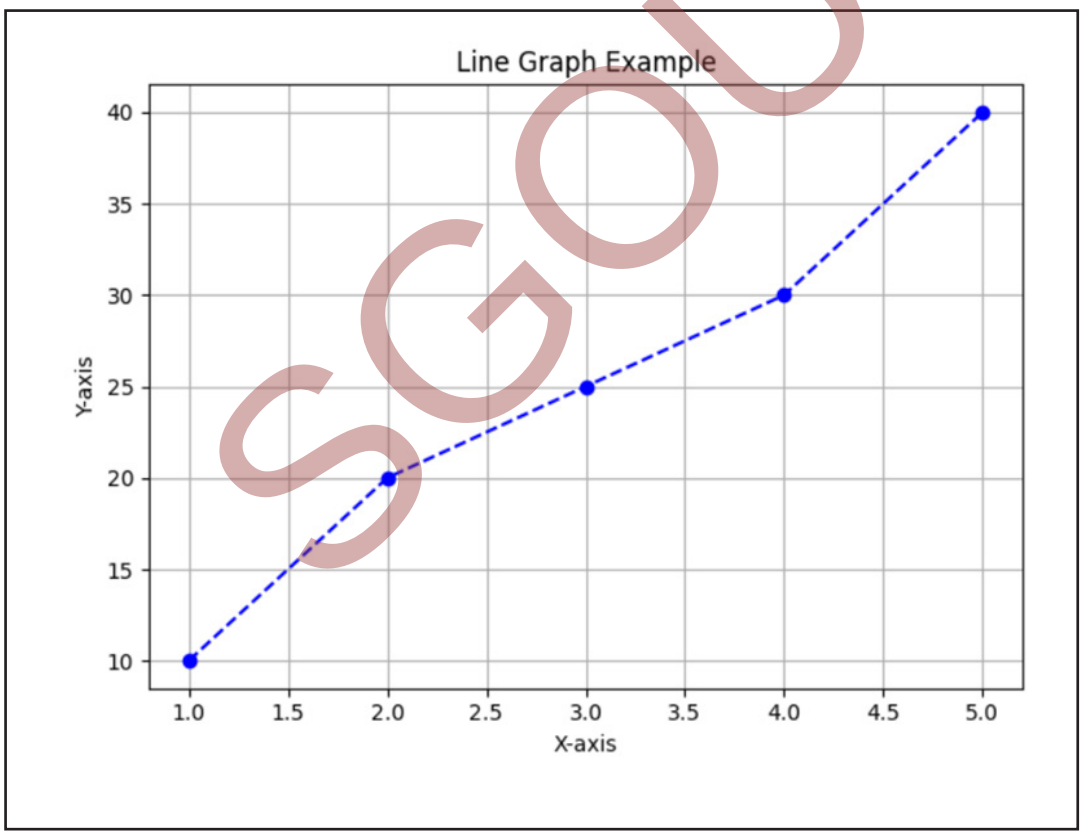
```
Original DataFrame:
   State  Govt_Colleges  Private_Colleges  Total_Seats     Fees
0     AP           10.0              20.0       3000.0  50000.0
1     TS           12.0              25.0       4000.0  60000.0
2     KA           15.0               NaN       5000.0      NaN
3     TN            NaN              30.0       4500.0  55000.0
4     MH           18.0              28.0          NaN  65000.0

After renaming AP to Andhra:
     State  Govt_Colleges  Private_Colleges  Total_Seats     Fees
0  Andhra           10.0              20.0       3000.0  50000.0
1      TS           12.0              25.0       4000.0  60000.0
2      KA           15.0               NaN       5000.0      NaN
3      TN            NaN              30.0       4500.0  55000.0
4      MH           18.0              28.0          NaN  65000.0

Count of Non-NaN values in each column:
 State             5
Govt_Colleges      4
Private_Colleges   4
Total_Seats        4
Fees               4
dtype: int64

Count of Non-NaN values in each row:
 0   5
1    5
2    3
3    4
4    4
dtype: int64

After increasing Fees by 5%:
     State  Govt_Colleges  Private_Colleges  Total_Seats     Fees
0  Andhra           10.0              20.0       3000.0  52500.0
1      TS           12.0              25.0       4000.0  63000.0
2      KA           15.0               NaN       5000.0      NaN
3      TN            NaN              30.0       4500.0  57750.0
4      MH           18.0              28.0          NaN  68250.0

After replacing NaN with 0:
     State  Govt_Colleges  Private_Colleges  Total_Seats     Fees
0  Andhra           10.0              20.0       3000.0  52500.0
1      TS           12.0              25.0       4000.0  63000.0
2      KA           15.0               0.0       5000.0      0.0
3      TN            0.0              30.0       4500.0  57750.0
4      MH           18.0              28.0          0.0  68250.0
```

SGOU

10. Write a Python program using Matplotlib to plot a line graph for the values of x = [1,2,3,4,5] and y = [10,20,25,30,40]. Add labels, title, and a grid.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y, marker='o', color='b', linestyle='--')
plt.title("Line Graph Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.show()
```

Output

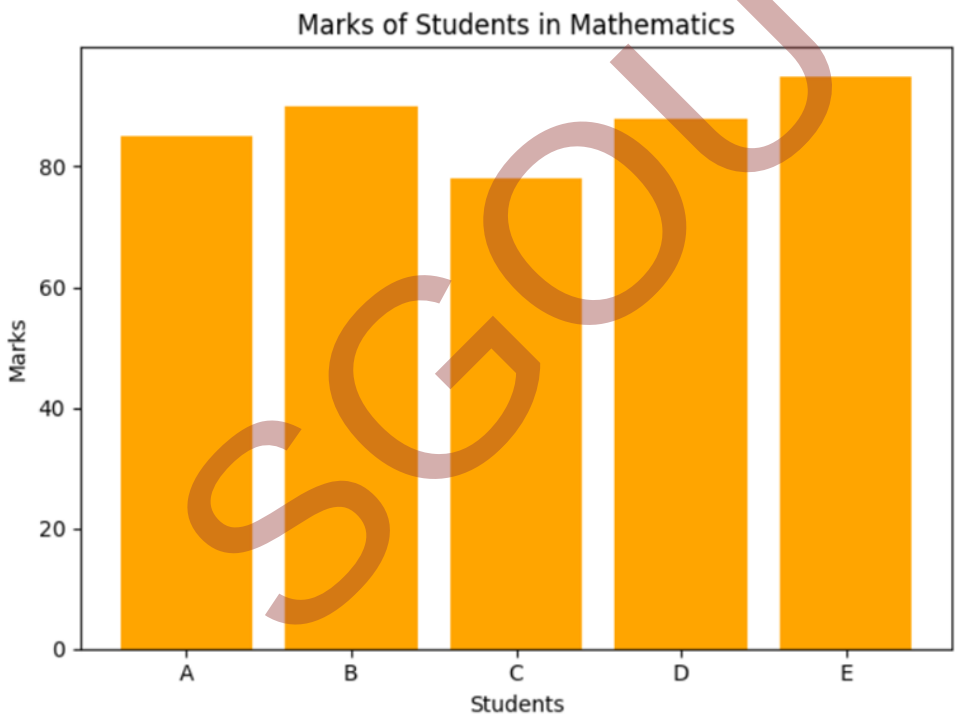


SGOU

11. Write a Python program to create a bar chart to represent the marks of five students in Mathematics.

```
import matplotlib.pyplot as plt

students = ["A", "B", "C", "D", "E"]
marks = [85, 90, 78, 88, 95]

plt.bar(students, marks, color="orange")
plt.title("Marks of Students in Mathematics")
plt.xlabel("Students")
plt.ylabel("Marks")
plt.show()
```

Output

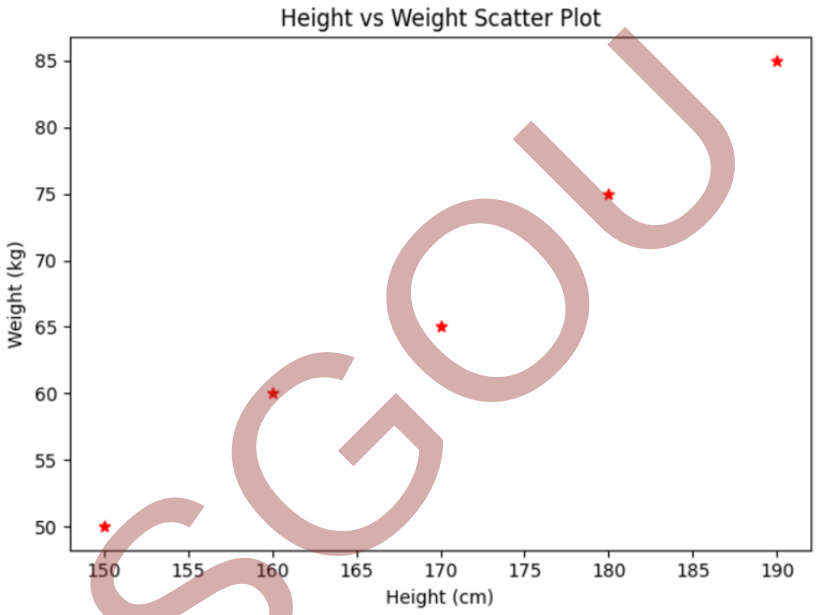


SGOU

12. Plot a scatter graph for two lists: height = [150, 160, 170, 180, 190] and weight = [50, 60, 65, 75, 85].

```
import matplotlib.pyplot as plt

height = [150, 160, 170, 180, 190]
weight = [50, 60, 65, 75, 85]

plt.scatter(height, weight, color="red", marker="*")
plt.title("Height vs Weight Scatter Plot")
plt.xlabel("Height (cm)")
plt.ylabel("Weight (kg)")
plt.show()
```

Output

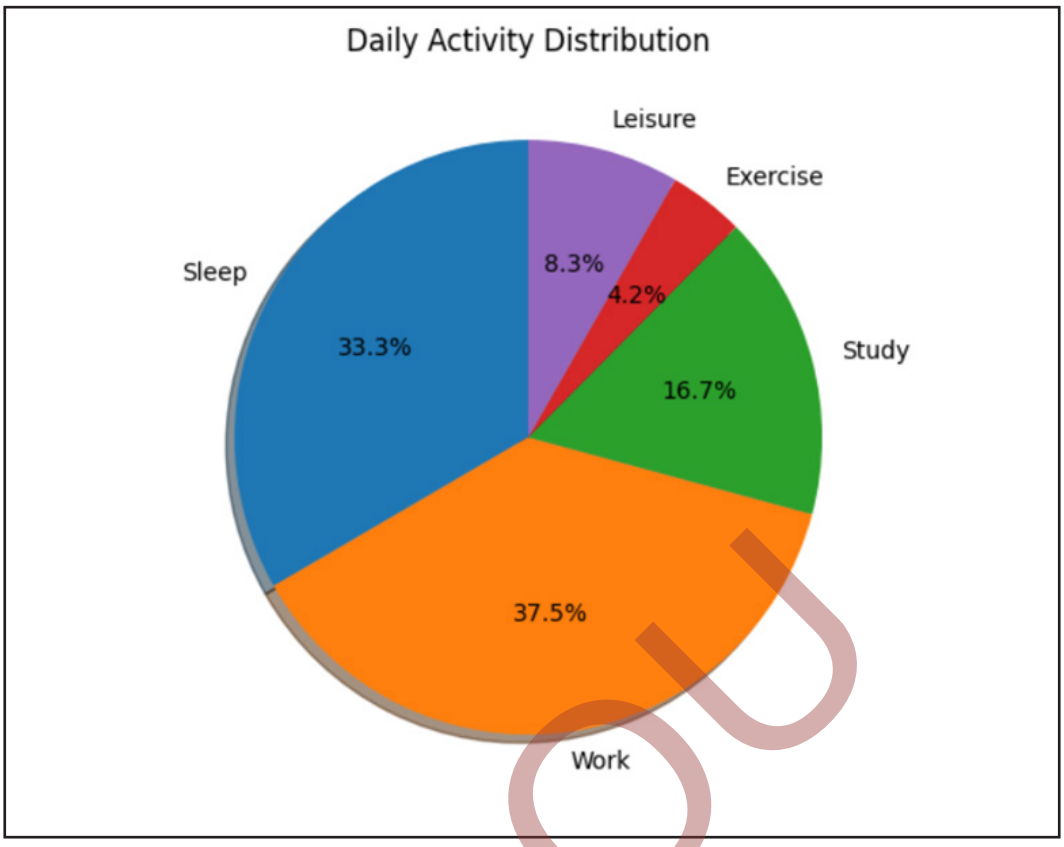


SGOU

13. Create a pie chart showing the percentage distribution of daily activities: Sleep (8 hours), Work (9 hours), Study (4 hours), Exercise (1 hour), Leisure (2 hours).

```
import matplotlib.pyplot as plt

activities = ['Sleep', 'Work', 'Study', 'Exercise', 'Leisure']
hours = [8, 9, 4, 1, 2]

plt.pie(hours, labels=activities, autopct='%1.1f%%', startangle=90, shadow=True)

plt.title("Daily Activity Distribution")
plt.show()
```

Output



14. Write a Python program to:

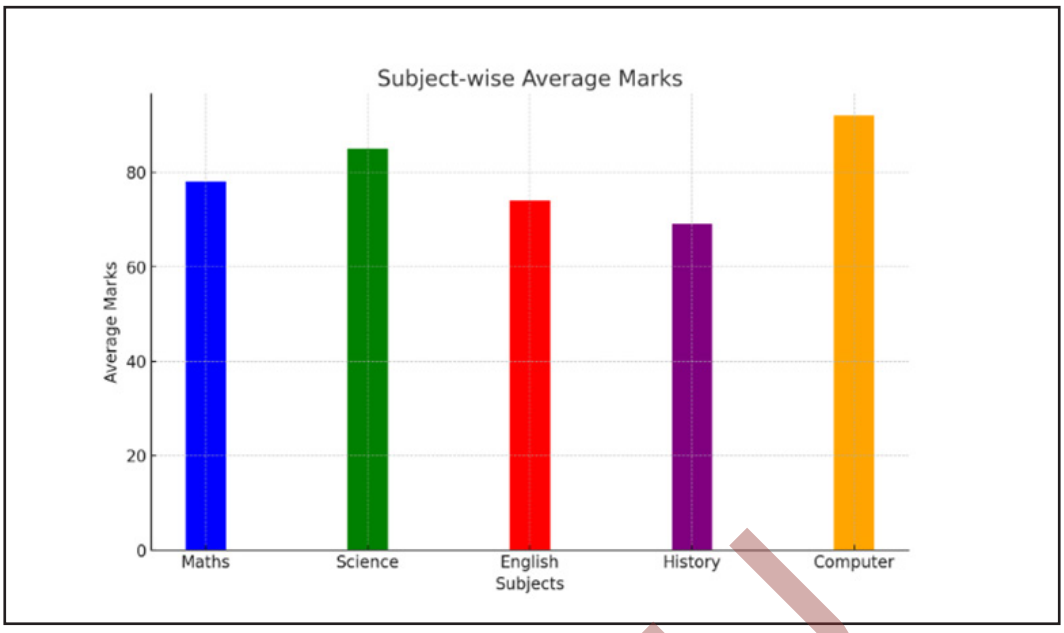
- Create a Pandas DataFrame containing the average marks of students in different subjects.
- Save this DataFrame into a CSV file.
- Read the CSV file and plot a bar chart using Matplotlib where:
  - The width of each bar is 0.25.
  - Each bar has a different color.
- Add a title and proper labels for X-axis and Y-axis to the chart.

SGOU

```
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Create a DataFrame for subject-wise average marks
data = {
   'Subject': ['Maths', 'Science', 'English', 'History', 'Computer'],
   'Average Marks': [78, 85, 74, 69, 92]
}
df = pd.DataFrame(data)

# Step 2: Save DataFrame to a CSV file
df.to_csv("subject_average.csv", index=False)

# Step 3: Read the CSV file
df_read = pd.read_csv("subject_average.csv")

# Step 4: Plot a bar chart
plt.bar(df_read['Subject'], df_read['Average Marks'], width=0.25,
        color=['blue', 'green', 'red', 'purple', 'orange'])

# Step 5: Add title and axis labels
plt.title("Subject-wise Average Marks")
plt.xlabel("Subjects")
plt.ylabel("Average Marks")

# Display the chart
plt.show()
```

SGOU

Output

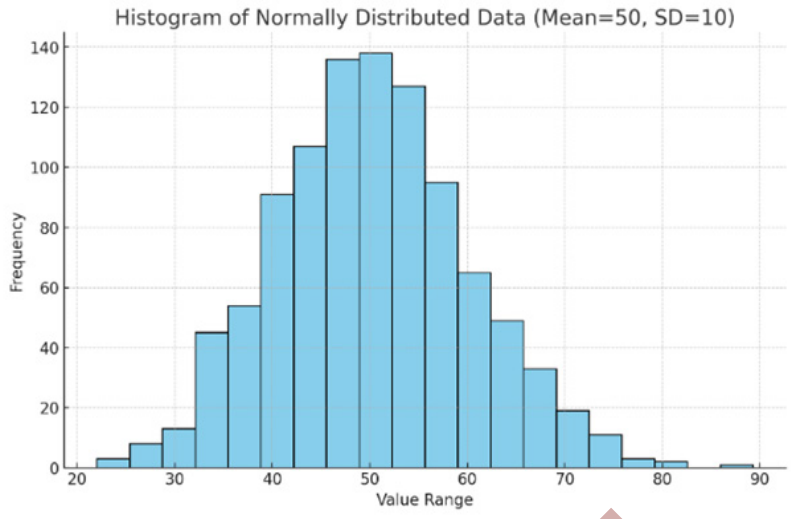


15. Write a Python program to generate 1,000 random numbers following a normal distribution (mean = 50, standard deviation = 10) using NumPy, and plot their histogram with 20 bins. Use different colors and add grid lines.

```
import numpy as np
import matplotlib.pyplot as plt

# Generate 1000 random numbers with normal distribution
# Mean = 50, Standard Deviation = 10
data = np.random.normal(50, 10, 1000)

# Plot histogram
plt.hist(data, bins=20, color='skyblue', edgecolor='black')

# Add grid, title, and labels
plt.grid(True, linestyle='--', alpha=0.7)
plt.title("Histogram of Normally Distributed Data (Mean=50, SD=10)")
plt.xlabel("Value Range")
plt.ylabel("Frequency")

# Display plot
plt.show()
```

SGOU

Output

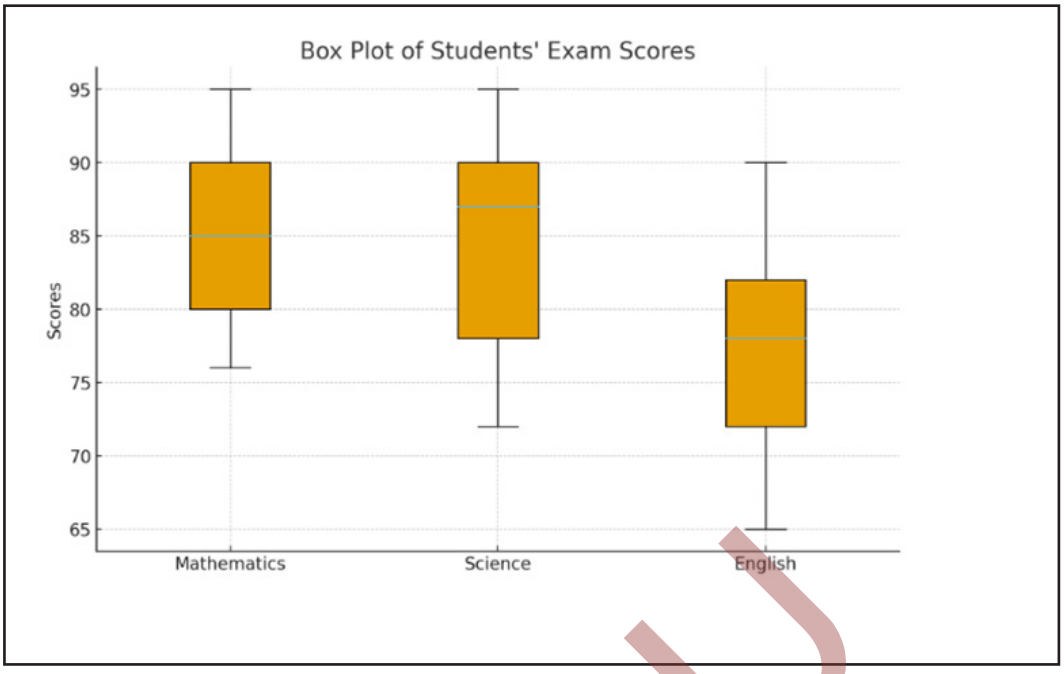


16. Write a Python program using Matplotlib to create a box plot of students' exam scores in three subjects: Mathematics, Science, and English. The program should display the median, quartiles, minimum, maximum, and any outliers for each subject. Add appropriate titles and axis labels.

```
import matplotlib.pyplot as plt

# Exam scores of students in 3 subjects
math_scores = [78, 85, 90, 95, 88, 76, 92, 85, 89, 77, 93, 80, 84]
science_scores = [72, 75, 78, 85, 89, 90, 95, 92, 88, 76, 84, 91, 87]
english_scores = [65, 70, 72, 68, 75, 80, 82, 78, 85, 90, 88, 76, 79]

# Combine data
data = [math_scores, science_scores, english_scores]

# Create box plot
plt.boxplot(data, labels=['Mathematics', 'Science', 'English'], patch_artist=True)

# Add title and labels
plt.title("Box Plot of Students' Exam Scores")
plt.ylabel("Scores")

# Display the plot
plt.show()
```

SGOU

Output



2.4.5 Lab Practice Questions

1.Write a Python program using NumPy to:

- Create a 1D array of the first 15 natural numbers.
- Create a 2D array of size 3x3 with numbers from 1 to 9.
- Create a 3x3 array of zeros and a 2x4 array of ones.
- Display all arrays and print their shapes.

2. Write a program to:

- Create an array of numbers from 5 to 50 with a step of 5 using arange().
- Create an array of 8 evenly spaced values between 0 and 2 using linspace().
- Display the arrays and their shapes.

3. Write a Python program to create a 1D array of 10 random integers between 1 and 100. Print the array, its shape, size, maximum, and minimum values.

4. Write a Python program to create two arrays [2, 4, 6] and [1, 3, 5] and perform addition, subtraction, multiplication, division, and square root operations.

5. Write a Python program to create an array [1, 2, 3, 4] and multiply all elements by 3 using broadcasting and display the original and resulting arrays.

SGOU

6. Write a Python program using NumPy to perform the following tasks:

   - Create a 1D array of even integers from 2 to 20.
   - Display the second element, seventh element, and last element of the array.
   - Print elements from index 3 to 8.
   - Print every third element of the array.
   - Reverse the array and display it.
   - Create a 2D array of size 4x5 using integers from 1 to 20.
   - Display the second row, first column, and a sub-array of elements from rows 2–3 and columns 2–4.

7. Write a Python program using Pandas to create a DataFrame of 5 students with columns: Name, Age, and Marks. Sort the DataFrame by Marks in descending order. Sort the DataFrame by Age in ascending order.

8. Write a Python program using Pandas to create a DataFrame of students with columns: RollNo, Name, and Marks. Add a new column Grade based on Marks (≥ 50: "Pass", otherwise: "Fail"). Remove the Marks column and display the final DataFrame.

9. Write a Python program using Pandas to:

   a. Create a DataFrame of employees with columns: EmpID, Name, Department, and Salary.

   b. Calculate and display the **average salary** of all employees.

   c. Filter and display employees who earn more than the average salary.

   d. Save the filtered DataFrame to a CSV file named **high_salary.csv**.

10. Sudha, an employee in a multinational company, has been asked to present a line chart comparing the number of matches played by India and Pakistan in different cricket series. The data is given below:

Cricket Series Name	Pataudi	Kapil	Gavaskar	Imran	Shane
Matches Played by India	15	20	18	9	5
Matches Played by Pakistan	10	15	12	15	8

The chart should be created with the following specifications:

SGOU

- Series names must appear on the X-axis.
- The line representing India should be in green color.
- The line representing Pakistan should be in red color.
- The chart must have a title: “Series Comparison between India and Pakistan”.
- The X-axis label should be “Series Name”.
- The Y-axis label should be “Matches Played”.

11. An organization conducted a survey to analyze how employees spend their working hours in a day. The data collected is shown below:

Activity	Hours Spent
Meetings	3
Project Work	5
Emails & Calls	2
Breaks	1
Training	1

The company wants to present this data in a **Pie Chart** with the following specifications:

- Each activity should be displayed as a separate slice of the pie.
- The slice representing **Project Work** should be highlighted using the **explode function**.
- Each slice should display its percentage value.
- The chart should have a title: **“Employee Workday Activity Distribution”**.

12. Write a Python program to create a Pandas DataFrame for analyzing the number of **Government and Private schools**, their **Total Students**, and **Annual Fees** for different states using sample data. Perform the following operations:

- Change the name of the state **UP** to **Uttar Pradesh**.
- Count and display the **non-NaN values** of each column.

SGOU

- Count and display the **non-NaN values** of each row.
- Increase the annual fees of all schools by **8%**.
- Replace all **NaN values with the average of the respective column**.

13. Write a Python program to generate 500 random numbers following a uniform distribution between 10 and 100 using NumPy, and plot their histogram with 15 bins. Use green color for the bars, add axis labels, a title, and display grid lines.

14. Write a Python program to draw a cumulative histogram of exam scores. Use bins of 10, add edge color to the bars, and title as *"Cumulative Distribution of Exam Scores"*.

15. Write a Python program to generate 1000 random numbers following a normal distribution (mean = 50, standard deviation = 10) using NumPy, and plot their histogram with 20 bins. Use different colors and add grid lines.

SGOU

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

DON'T LET IT BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

PYTHON PROGRAMMING LAB

COURSE CODE: M25CA02PC

SGOU

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841