Shram Sadhana Bombay Trust's

COLLEGE OF ENGINEERING AND TECHNOLOGY,

BAMBHORI POST BOX NO. 94, JALGAON – 425001.(M.S.)

Included under section 2 (f) & 12 (B) of the UGC Act, 1956

NAAC Re-accredited Grade: A (3.14) (2nd Cycle)

ISO 9001: 2008 Certified

Phone No. (0257) 2258393, Fax No. (0257) 2258392

Website- www.sscoetjalgaon.ac.in

Email: sscoetjal@gmail.com

ISO 9001:2008

DEPARTMENT OF COMPUTER APPLICATIONS (M.C.A.)

MCA

SSBT

COET

Laboratory Journal

Class : M.C.A - I

Sem : I

Subject : MCA-416 Lab on Python Programming

Academic Year : 2024-25

University Exam. Seat No :

Name of the Faculty : Asst. Prof. Sapana A. Fegade

Vision of the Institute

Today we carry the flame of quality education, knowledge and progressive technology for global societal development; tomorrow the flame will glow even brighter.

Mission of the Institute

To provide conducive environment for preparing competent, value added and patriotic engineers of integrity of par excellence to meet global standards for societal development.

DEPARTMENT OF COMPUTER APPLICATIONS (M.C.A.)

Vision of the Department

To develop skilled professionals with social values as per industry needs.

Mission of the Department

To provide a student-centred, conducive environment for preparing knowledgeable, skilled and value added professionals to stand out in a competitive world.

Department of Computer Applications (M.C.A.)

Program Outcomes (POs)

PO1: Computational Knowledge: Understand and apply mathematical foundation, computing and domain knowledge for the conceptualization of computing models from defined problems.

PO2: Problem Analysis: Ability to identify, critically analyze and formulate complex computing problems using fundamentals of computer science and application domains.

PO3: **Design / Development of Solutions:** Ability to transform complex business scenarios and contemporary issues into problems, investigate, understand and propose integrated solutions using emerging technologies.

PO4: Conduct Investigations of Complex Computing Problems: Ability to devise and conduct experiments, interpret data and provide well informed conclusions.

PO5: Modern Tool Usage: Ability to select modern computing tools, skills and techniques necessary for innovative software solutions.

PO6: Professional Ethics: Ability to apply and commit professional ethics and cyber regulations in a global economic environment.

PO7: Life-long Learning: Recognize the need for and develop the ability to engage in continuous learning as a Computing professional.

PO8: Project Management and Finance: Ability to understand, management and computing principles with computing knowledge to manage projects in multidisciplinary environments.

PO9: Communication Efficacy: Communicate effectively with the computing community as well as society by being able to comprehend effective documentations and presentations.

PO10: Societal & Environmental Concern: Ability to recognize economic, environmental, social, health, legal, ethical issues involved in the use of computer technology and other consequential responsibilities relevant to professional practice.

PO11: Individual and Team Work: Ability to work as a member or leader in diverse teams in multidisciplinary environment.

PO12: Innovation and Entrepreneurship: Identify opportunities, entrepreneurship vision and use of innovative ideas to create value and wealth for the betterment of the individual and society.

Department of Computer Applications (M.C.A.)

Program Educational Objectives (PEOs)

PEO 1: Core Knowledge -The computing professional graduate will have the knowledge of basic science and Engineering skills, Humanities, social science, management and conceptual and practical understanding of core computer applications area with project development.

PEO 2: Employment/ Continuing Education – The computing professional graduate will have the knowledge of Industry-based technical skills to succeed in entry level professional position at various industries as well as in academics.

PEO 3: Professional Competency - The computing professional graduate will have the ability to communicate effectively in English, to accumulate and disseminate the knowledge and to work effectively in a team with a sense of social awareness.

Department of Computer Applications (M.C.A.)

Program Specific Outcomes (PSOs)

PSO 1: The computing professional graduate will be able to apply knowledge of computer science in practice to identify, critically analyze, formulate and develop computer applications using modern computing tools and techniques and will use these tools with dexterity.

PSO 2: The computing professional graduate will be able to design computing systems to meet desired needs within realistic constraints such as safety, security and applicability. These systems will function professionally with ethical responsibility as an individual as well as in multidisciplinary teams with positive attitude.

PSO 3: The computing professional graduate will be able to appreciate the importance of goal setting and recognize the need for life-long learning with good communication skills.

Department of Computer Applications (M.C.A.)

MCA-416 Lab on Python Programming

M.C.A-I Sem-I

Course Outcomes:

CO1: Recall Python syntax rules, including variables, data types, and basic control structures (loops, conditionals).

CO2: Demonstrate the use and working of various data types, control structures, files, and exception handling.

CO3: Create, configure, and make use of modules.

CO4: Develop console-based and GUI applications (both procedural and object-oriented) to solve different problems.

CO-PO and PSO Mappings

Course Outcomes	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PS O1	PS O2	PS O3
CO1:	3	2											3	2	
CO2:	3	3		3	3								3	2	3
CO3:	3	2	3		3							2	2	2	
CO4:	3	3	3	3	3						3	3	3	3	
CO5:	3	2											3	2	

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 01

Date of Performance: **Date of Completion:**

Title: Write a process for Python Step-by-step Installation .

Process :-

How to Install Python in Windows?

We have provided step-by-step instructions to guide you and ensure a successful installation. Whether you are new to programming or have some experience, mastering how to install Python on Windows will enable you to utilize this potent language and uncover its full range of potential applications.

To download Python on your system, you can use the following steps

1. Download Python

1. **Visit the Python website:**

   Go to https://www.python.org/.

2. **Navigate to the Downloads page:**

   The homepage usually shows a big “Download Python” button that automatically provides the latest version for your operating system.

   Alternatively:

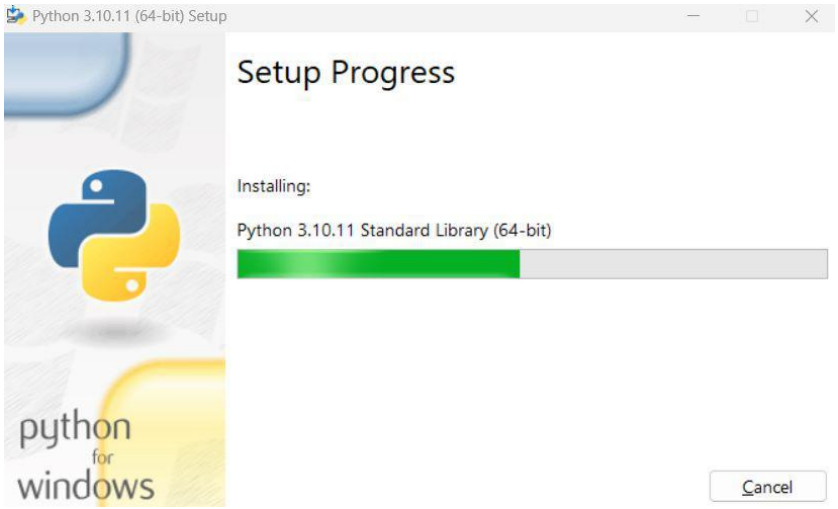
   - Click the "Downloads" menu and select your operating system (Windows, macOS, or Linux).

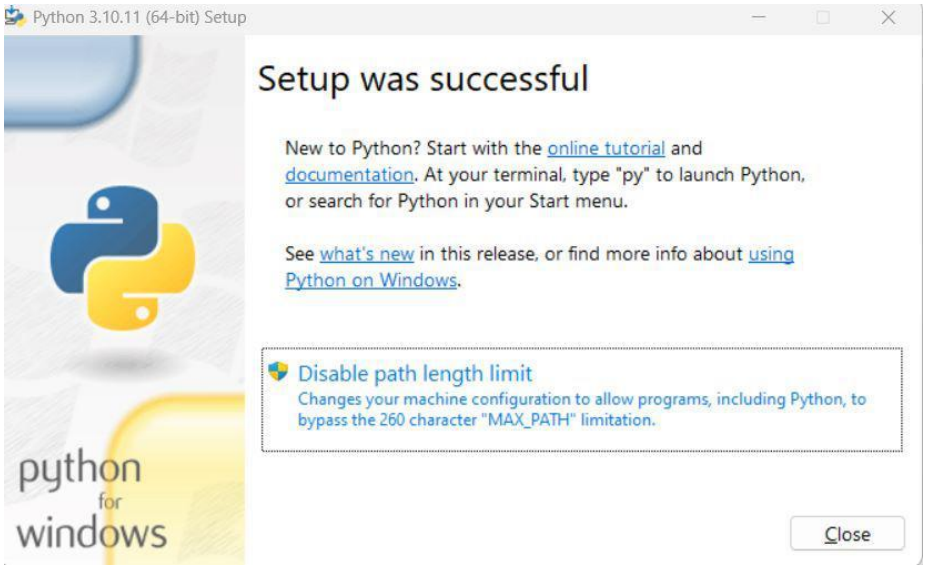
2. Install Python on Windows

1. **Run the Installer:**
   - After downloading the .exe file, double-click it to start the installation.
2. **Select "Add Python to PATH":**
   - On the first screen of the installer, make sure to check the box that says ″Add Python x.x to PATH″ (this is very important).

3. **Choose the Installation Type:**

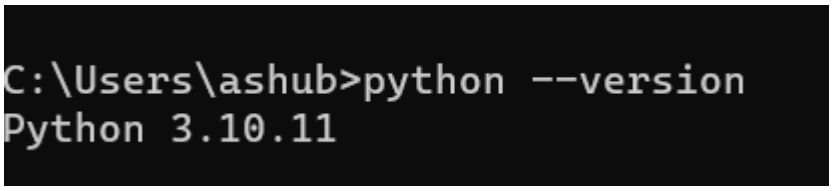
- Click "Install Now" for a default installation, or choose "Customize Installation" to select specific options and installation directories.





4. **Complete the Installation:**
   - After installation, click "Close." You can verify the installation by opening Command Prompt and typing:

     python –version



5. **Install an Integrated Development Environment (IDE) (Optional)**

To write and execute Python code easily, install an IDE:

- **PyCharm:** Download PyCharm
- **VS Code:** Download Visual Studio Code
- **IDLE:** Comes pre-installed with Python.

6. **Verify Python Installation**

1. Open a terminal (Command Prompt, PowerShell, or Terminal).
2. Run:

   python --version

   or

   python3 --version

   It should display the Python version you installed.

You're ready to start coding in Python! 🎉

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 02

Date of Performance: **Date of Completion:**

Title: Implement a program based on strings.(slicing, casting, string function)

Code:

```
Original string
text = "Learning Python is fun and educational!"
# 1. String Slicing
print("Original text:", text)
# Slicing the first 8 characters
print("Sliced text (first 8 characters):", text[:8])
# Slicing the last 4 characters
print("Sliced text (last 4 characters):", text[-4:])
# Slicing from index 9 to 15
print("Sliced text (index 9 to 15):", text[9:16])
# Slicing every 2nd character
print("Sliced text (every 2nd character):", text[::2])
# Slicing with step (reverse the string)
print("Sliced text (reversed):", text[::-1])
# 2. Casting
number = 1234
number_str = str(number)  # Casting integer to string
print("Number as a string:", number_str)
# Casting float to string
pi = 3.14159
pi_str = str(pi)
```

```
print("Pi as a string:", pi_str)
# Casting string to integer
number_str2 = "5678"
number_int = int(number_str2)
print("String '5678' as an integer:", number_int)
# Casting string to float
float_str = "12.34"
float_num = float(float_str)
print("String '12.34' as a float:", float_num)
# Casting float to integer
pi_int = int(pi)
 # Casting float to integer
print("Float 3.14159 as an integer:", pi_int)
# Casting integer to float
number_float = float(number)
# Casting integer to float
print("Integer 1234 as a float:", number_float)
# 3. String Functions
print("Uppercase text:", text.upper())            # Convert to uppercase
print("Lowercase text:", text.lower())            # Convert to lowercase
print("Replacing 'fun' with 'exciting':", text.replace("fun", "exciting"))
 # Replacing a substring
print("Is the text alphanumeric?", text.isalnum())
# Check if the string is alphanumeric
print("Is the text alphabetic?", text.isalpha())
 # Check if the string is alphabetic (False in this case)
# Finding the length of the string
print("Length of the text:", len(text))
```

Output:

Original text: Learning Python is fun and educational!
Sliced text (first 8 characters): Learning
Sliced text (last 4 characters): nal!
Sliced text (index 9 to 15): Python
Sliced text (every 2nd character): Lann yhni u n dctoa!
Sliced text (reversed): !lanoitacude dna nuf si nohtyP gninraeL
Number as a string: 1234
Pi as a string: 3.14159
String '5678' as an integer: 5678
String '12.34' as a float: 12.34
Float 3.14159 as an integer: 3
Integer 1234 as a float: 1234.0
Uppercase text: LEARNING PYTHON IS FUN AND EDUCATIONAL!
Lowercase text: learning python is fun and educational!
Replacing 'fun' with 'exciting': Learning Python is exciting and educational!
Is the text alphanumeric? False
Is the text alphabetic? False
Length of the text: 39

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 03

Date of Performance: **Date of Completion:**

Title: Implement a program based on Simple statements using Variables, Scope of variable, Built-in Data Types, Types of operator, etc.

Code:

```
# 1. Variables and Built-in Data Types
name = "Jake"              # String
age = 30                   # Integer
salary = 50000.50          # Float (Global Variable)
is_employed = True         # Boolean
skills = ["Python", "Java"]   # List
personal_info = ("Male", 6.1)  # Tuple (Gender, Height)
profile = {"city": "Chicago", "role": "Engineer"}  # Dictionary

# 2. Arithmetic Operators
bonus = 5000
deduction = 1000
total_income = salary + bonus - deduction  # Subtraction operator added

# Print the total income after bonus and deduction
print(f"Total income after bonus and deduction is: {total_income}")

# 3. Comparison Operators
if age > 18:
    print(f"{name} is an adult.")
```

```
# 4. Logical Operators
if is_employed and salary > 30000:
    print(f"{name} is employed with a good salary.")

if not is_employed or salary < 30000:
    print(f"{name} either is not employed or has a low salary.")
else:
    print(f"{name} is employed and has a sufficient salary.")

# 5. Scope of Variable (Local vs Global)
def update_salary():
    global salary  # Declare salary as global to modify it
    increment = 2000  # Local variable
    salary = salary + increment  # Modify global variable 'salary'
    return salary

new_salary = update_salary()  # Call function to update global salary
print(f"{name}'s new salary after increment is: {new_salary}")

# 6. Accessing List, Tuple, and Dictionary
print(f"{name} has skills in {skills[0]} and {skills[1]}.")  # Accessing List
print(f"{name} is {personal_info[1]} feet tall.")          # Accessing Tuple
print(f"{name} lives in {profile['city']} and works as a {profile['role']}.")  # Accessing Dictionary

# 7. Bitwise Operators
a = 10  # 1010 in binary
```

```
b = 4   # 0100 in binary
bitwise_and = a & b  # Bitwise AND
bitwise_or = a | b   # Bitwise OR
bitwise_xor = a ^ b  # Bitwise XOR
bitwise_shift_left = a << 1  # Shift left by 1
bitwise_shift_right = a >> 1  # Shift right by 1
print(f"Bitwise AND: {bitwise_and}, OR: {bitwise_or}, XOR: {bitwise_xor}")
print(f"Left Shift: {bitwise_shift_left}, Right Shift: {bitwise_shift_right}")

# 8. Membership Operators
if "Python" in skills:
    print(f"{name} knows Python.")
else:
    print(f"{name} doesn't know Python.")

if "C++" not in skills:
    print(f"{name} is not proficient in C++.")

# 9. Identity Operators
if salary is new_salary:
    print("salary and new_salary refer to the same object.")
else:
    print("salary and new_salary are different objects.")

if name is not age:
    print("name and age are not the same object.")
```

Output :

Total income after bonus and deduction is: 54000.5
Jake is an adult.
Jake is employed with a good salary.
Jake is employed and has a sufficient salary.
Jake's new salary after increment is: 52000.5
Jake has skills in Python and Java.
Jake is 6.1 feet tall.
Jake lives in Chicago and works as a Engineer.
Bitwise AND: 0, OR: 14, XOR: 14
Left Shift: 20, Right Shift: 5
Jake knows Python.
Jake is not proficient in C++.
salary and new_salary refer to the same object.
name and age are not the same object.

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 04

Date of Performance: **Date of Completion:**

Title: Implement a program based on decision and looping statements.

Code:

```
# Get user input for a number
number = int(input("Enter a number: "))

# Example of an if statement
if number > 0:
    print("The number is positive.")
else:
    print("The number is not positive.")

# Example of an if-else statement
if number % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")

# Example of an if-elif-else statement
if number < 10:
    print("The number is low.")
elif number <= 20:
    print("The number is medium.")
else:
```

```
    print("The number is high.")

# Using a for loop to count from 1 to the number
print("\nCounting from 1 to", number, ":")
for i in range(1, number + 1):
    print(i)

# Using a while loop to count down from the number to 1
print("Counting down from", number, ":")
while number > 0:
    print(number)
    number -= 1
```

Output:

Enter a number: 10

The number is positive.

The number is even.

The number is medium.

Counting from 1 to 10 :

1 6

2 7

3 8

4 9

5 10

Counting down from 10 :

10 5

9 4

8 3

7 2

6 1

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 05

Date of Performance: **Date of Completion:**

Title : Implement a program based on Tuples, Lists, Dictionaries and Sets etc.

Code:

```
# 1. Tuple: Storing student information (name, roll number)
student1_info = ("John Doe", 101)
student2_info = ("Jane Smith", 102)

# 2. List: Marks of the student in different subjects
student1_marks = [85, 92, 78]  # Math, English, Science
student2_marks = [89, 95, 80]  # Math, English, Science

# 3. Set: Unique subjects (ensures no duplicates)
subjects = {"Math", "English", "Science"}

# 4. Dictionary: Storing all student information with roll number as key
students_data = {
    student1_info[1]: {
        "name": student1_info[0],
        "marks": student1_marks,
        "average": sum(student1_marks) / len(student1_marks)
    },
    student2_info[1]: {
        "name": student2_info[0],
        "marks": student2_marks,
```

```
        "average": sum(student2_marks) / len(student2_marks)
    }
}

# Performing operations on List (student1_marks)
print("Operations on List (student1_marks):")
print("Original Marks:", student1_marks)

# Adding a new mark
student1_marks.append(88)
print("After appending a new mark:", student1_marks)

# Sorting marks in ascending order
student1_marks.sort()
print("After sorting in ascending order:", student1_marks)

# Removing the last mark
student1_marks.pop()
print("After popping last mark:", student1_marks)

# Reversing the list
student1_marks.reverse()
print("After reversing the list:", student1_marks)
print()

# Performing operations on Set (subjects)
print("Operations on Set (subjects):")
print("Original Subjects:", subjects)
```

```
# Adding a new subject
subjects.add("History")
print("After adding a new subject:", subjects)

# Removing a subject
subjects.remove("Math")
print("After removing a subject:", subjects)

# Checking if a subject exists
print("Is 'Science' in subjects?", 'Science' in subjects)
print()

# Performing operations on Tuple (student1_info)
print("Operations on Tuple (student1_info):")
print("Name:", student1_info[0])
print("Roll Number:", student1_info[1])

# Tuples are immutable, so no modification operations are allowed
print("Tuples are immutable. We cannot add, remove, or change values.")
print()

# Performing operations on Dictionary (students_data)
print("Operations on Dictionary (students_data):")
print("Original Data:", students_data)

# Adding a new student to the dictionary
student3_info = ("Emily Davis", 103)
student3_marks = [91, 85, 88]
```

```
students_data[student3_info[1]] = {
    "name": student3_info[0],
    "marks": student3_marks,
    "average": sum(student3_marks) / len(student3_marks)
}
print("After adding a new student:", students_data)

# Updating marks for a student
students_data[101]["marks"] = [90, 88, 95]
students_data[101]["average"] = sum(students_data[101]["marks"]) /
len(students_data[101]["marks"])
print("After updating marks for student 101:", students_data)

# Removing a student from the dictionary
del students_data[102]
print("After removing student 102:", students_data)

# Accessing specific data
print("Student 101's name:", students_data[101]["name"])
print("Student 101's marks:", students_data[101]["marks"])
print("Student 101's average marks:", students_data[101]["average"])
```

Output:

Operations on List (student1_marks):
Original Marks: [85, 92, 78]
After appending a new mark: [85, 92, 78, 88]
After sorting in ascending order: [78, 85, 88, 92]
After popping last mark: [78, 85, 88]
After reversing the list: [88, 85, 78]

Operations on Set (subjects):
Original Subjects: {'Math', 'Science', 'English'}
After adding a new subject: {'Math', 'Science', 'English', 'History'}
After removing a subject: {'Science', 'English', 'History'}
Is 'Science' in subjects? True

Operations on Tuple (student1_info):
Name: John Doe
Roll Number: 101
Tuples are immutable. We cannot add, remove, or change values.

Operations on Dictionary (students_data):
Original Data: {101: {'name': 'John Doe', 'marks': [88, 85, 78], 'average': 85.0}, 102: {'name': 'Jane Smith', 'marks': [89, 95, 80], 'average': 88.0}}
After adding a new student: {101: {'name': 'John Doe', 'marks': [88, 85, 78], 'average': 85.0}, 102: {'name': 'Jane Smith', 'marks': [89, 95, 80], 'average': 88.0}, 103: {'name': 'Emily Davis', 'marks': [91, 85, 88], 'average': 88.0}}
After updating marks for student 101: {101: {'name': 'John Doe', 'marks': [90, 88, 95], 'average': 91.0}, 102: {'name': 'Jane Smith', 'marks': [89, 95, 80],

'average': 88.0}, 103: {'name': 'Emily Davis', 'marks': [91, 85, 88], 'average': 88.0}}

After removing student 102: {101: {'name': 'John Doe', 'marks': [90, 88, 95], 'average': 91.0}, 103: {'name': 'Emily Davis', 'marks': [91, 85, 88], 'average': 88.0}}

Student 101's name: John Doe

Student 101's marks: [90, 88, 95]

Student 101's average marks: 91.0

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 06

Date of Performance: **Date of Completion:**

Title: Implement a program based on Functions, Function with parameters.

Code:

```
# Function to calculate the area of a circle
def area_of_circle(radius):
    return 3.14159 * radius * radius
# Function to calculate the area of a rectangle
def area_of_rectangle(length, width):
    return length * width

# Function to calculate the area of a triangle
def area_of_triangle(base, height):
    return 0.5 * base * height

# Main program
while True:
    print("\nChoose a shape to calculate the area:")
    print("1. Circle")
    print("2. Rectangle")
    print("3. Triangle")

    choice = int(input("Enter your choice (1-3): "))

    if choice == 1:
```

```
        r = float(input("Enter the radius of the circle: "))
        print("Area of the circle:", area_of_circle(r))

    elif choice == 2:
        l = float(input("Enter the length of the rectangle: "))
        w = float(input("Enter the width of the rectangle: "))
        print("Area of the rectangle:", area_of_rectangle(l, w))

    elif choice == 3:
        b = float(input("Enter the base of the triangle: "))
        h = float(input("Enter the height of the triangle: "))
        print("Area of the triangle:", area_of_triangle(b, h))

    else:
        print("Invalid choice")

    # Ask the user if they want to perform another calculation
    continue_choice = input("\nDo you want to perform another calculation?
(yes/no): ").lower()

    if continue_choice != 'yes':
        print("Exiting the program.")
        break
```

Output:

Choose a shape to calculate the area:

1. Circle
2. Rectangle
3. Triangle

Enter your choice (1-3): 1

Enter the radius of the circle: 5

Area of the circle: 78.53975

Do you want to perform another calculation? (yes/no): yes

Choose a shape to calculate the area:

1. Circle
2. Rectangle
3. Triangle

Enter your choice (1-3): 2

Enter the length of the rectangle: 10

Enter the width of the rectangle: 8

Area of the rectangle: 80.0

Do you want to perform another calculation? (yes/no): no

Exiting the program**.**

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 07

Date of Performance: **Date of Completion:**

Title: . Implement a program based on Functions Inside of Functions.

Code:

```
def calculator(a, b):
    # Outer function

    def add():
        # Inner function to add two numbers
        return a + b
    def subtract():
        # Inner function to subtract two numbers
        return a - b

    def multiply():
        # Inner function to multiply two numbers
        return a * b
    def divide():
        # Inner function to divide two numbers (with a check to avoid division by
zero)
        if b != 0:
            return a / b
        else:
            return "Division by zero is not allowed"
```

```
    # Returning results of all operations
    return add(), subtract(), multiply(), divide()

# Calling the outer function
a = 10
b = 2
result_add, result_subtract, result_multiply, result_divide = calculator(a, b)

print("Addition:", result_add)
print("Subtraction:", result_subtract)
print("Multiplication:", result_multiply)
print("Division:", result_divide)
```

Output:

Addition: 12

Subtraction: 8

Multiplication: 20

Division: 5.0

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 08

Date of Performance: **Date of Completion:**

Title: Implement a program based on built-in functions.

Code:

```
# List of numbers
numbers = [3, 8, 1, 6, 0, 8, 4, 10, 12, 11]

# Using built-in functions

# 1. Find the length of the list
length = len(numbers)

# 2. Find the maximum value in the list
maximum = max(numbers)

# 3. Find the minimum value in the list
minimum = min(numbers)

# 4. Calculate the sum of all values in the list
total_sum = sum(numbers)

# 5. Sort the list in ascending order
sorted_numbers = sorted(numbers)

# 6. Find the absolute value of a negative number
```

```
negative_number = -20
absolute_value = abs(negative_number)

# 7. Check if all elements are positive
all_positive = all(num > 0 for num in numbers)

# 8. Check if any element is zero
any_zero = any(num == 0 for num in numbers)

# 9. Calculate the power of a number (e.g., 2^3)
power_result = pow(2, 3)

# 10. Round a floating point number to two decimal places
float_number = 5.6789
rounded_value = round(float_number, 2)

# 11. Get quotient and remainder from division using divmod
quotient, remainder = divmod(10, 3)

# 12. Enumerate over the list to get index and value
enumerated_list = list(enumerate(numbers))

# 13. Zip two lists together
letters = ['A', 'B', 'C', 'D', 'E']
numbers_small = [1, 2, 3, 4, 5]
zipped_list = list(zip(letters, numbers_small))

# Output results
print(f"List of numbers: {numbers}")
```

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


print(f"Maximum value: {maximum}")
print(f"Minimum value: {minimum}")
print(f"Sum of all values: {total_sum}")
print(f"Sorted list: {sorted_numbers}")
print(f"Absolute value of {negative_number}: {absolute_value}")
print(f"Are all numbers positive? {all_positive}")
print(f"Is any number zero? {any_zero}")
print(f"Power of 2^3: {power_result}")
print(f"Rounded value of {float_number}: {rounded_value}")
print(f"Quotient and remainder of 10 divided by 3: {quotient}, {remainder}")
print(f"Enumerated list (index, value): {enumerated_list}")
print(f"Zipped list (letter, number): {zipped_list}")
```

Output:

List of numbers: [3, 8, 1, 6, 0, 8, 4, 10, 12, 11]

Length of the list: 10

Maximum value: 12

Minimum value: 0

Sum of all values: 63

Sorted list: [0, 1, 3, 4, 6, 8, 8, 10, 11, 12]

Absolute value of -20: 20

Are all numbers positive? False

Is any number zero? True

Power of 2^3: 8

Rounded value of 5.6789: 5.68

Quotient and remainder of 10 divided by 3: 3, 1

Enumerated list (index, value): [(0, 3), (1, 8), (2, 1), (3, 6), (4, 0), (5, 8), (6, 4), (7, 10), (8, 12), (9, 11)]

Zipped list (letter, number): [('A', 1), ('B', 2), ('C', 3), ('D', 4), ('E', 5)]

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 09

Date of Performance: **Date of Completion:**

Title : Implement a program using Anonymous Functions - Lambda and Filter.

Code:

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

# Example 1: Filter even numbers
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print("Even numbers:", even_numbers)

# Example 2: Filter odd numbers
odd_numbers = list(filter(lambda x: x % 2 != 0, numbers))
print("Odd numbers:", odd_numbers)

# Example 3: Filter numbers greater than 5
greater_than_five = list(filter(lambda x: x > 5, numbers))
print("Numbers greater than 5:", greater_than_five)

# Example 4: Filter numbers that are multiples of 3
multiples_of_three = list(filter(lambda x: x % 3 == 0, numbers))
print("Multiples of 3:", multiples_of_three)
```

Output:

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

Odd numbers: [1, 3, 5, 7, 9]

Numbers greater than 5: [6, 7, 8, 9, 10]

Multiples of 3: [3, 6, 9]

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 10

Date of Performance: **Date of Completion:**

Title : Implement a program using Maps, List Comprehension and Dictionaries.

Code:

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

# ---------- 1. map() Examples ----------

# 1.1 Using map() to square each number in the list
squared_numbers = list(map(lambda x: x ** 2, numbers))
print("Squared numbers using map():", squared_numbers)

# 1.2 Using map() to convert numbers to their string representation
string_numbers = list(map(str, numbers))
print("String representation of numbers using map():", string_numbers)

# 1.3 Using map() with multiple iterables to sum two lists element-wise
list1 = [1, 2, 3, 4, 5]
list2 = [10, 20, 30, 40, 50]
summed_list = list(map(lambda x, y: x + y, list1, list2))
print("Sum of elements in list1 and list2 using map():", summed_list)
```

```
# ---------- 2. List Comprehension Examples ----------

# 2.1 Using List Comprehension to create a list of cubes of numbers
cubed_numbers = [x ** 3 for x in numbers]
print("Cubed numbers using list comprehension:", cubed_numbers)

# 2.2 Using List Comprehension to get only even numbers
even_numbers = [x for x in numbers if x % 2 == 0]
print("Even numbers using list comprehension:", even_numbers)

# 2.3 Using List Comprehension to create a list of tuples (number, square)
number_square_pairs = [(x, x ** 2) for x in numbers]
print("List of (number, square) pairs using list comprehension:",
number_square_pairs)

# ---------- 3. Dictionary Examples ----------
# 3.1 Using a Dictionary to map each number to its square and cube
number_dict = {x: {"square": x ** 2, "cube": x ** 3} for x in numbers}
print("Dictionary mapping numbers to their squares and cubes:")
for num, values in number_dict.items():
    print(f"{num} -> Square: {values['square']}, Cube: {values['cube']}")

# 3.2 Using a Dictionary to create a lookup of even and odd numbers
parity_dict = {x: ("even" if x % 2 == 0 else "odd") for x in numbers}
print("Dictionary mapping numbers to their parity (even/odd):")
for num, parity in parity_dict.items():
    print(f"{num} is {parity}")
```

Output:

Squared numbers using map(): [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
String representation of numbers using map(): ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
Sum of elements in list1 and list2 using map(): [11, 22, 33, 44, 55]
Cubed numbers using list comprehension: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
Even numbers using list comprehension: [2, 4, 6, 8, 10]
List of (number, square) pairs using list comprehension: [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100)]

Dictionary mapping numbers to their squares and cubes:
1 -> Square: 1, Cube: 1
2 -> Square: 4, Cube: 8
3 -> Square: 9, Cube: 27
4 -> Square: 16, Cube: 64
5 -> Square: 25, Cube: 125
6 -> Square: 36, Cube: 216
7 -> Square: 49, Cube: 343
8 -> Square: 64, Cube: 512
9 -> Square: 81, Cube: 729
10 -> Square: 100, Cube: 1000

Dictionary mapping numbers to their parity (even/odd):

1 is odd	6 is even
2 is even	7 is odd
3 is odd	8 is even
4 is even	9 is odd
5 is odd	10 is even

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 11

Date of Performance: **Date of Completion:**

Title : Implement a program to demonstrate Class and Object.

Code:

```
# Defining a class named "Car"
class Car:
    # Constructor method to initialize the Car object
    def __init__(self, make, model, year):
        self.make = make  # Brand of the car (e.g., Toyota, Ford)
        self.model = model  # Model of the car (e.g., Corolla, Mustang)
        self.year = year  # Year the car was manufactured

    # Method to display car details
    def display_info(self):
        print(f"Car Information: {self.year} {self.make} {self.model}")

    # Method to simulate starting the car
    def start_engine(self):
        print(f"The engine of the {self.make} {self.model} is now running.")

    # Method to simulate stopping the car
    def stop_engine(self):
        print(f"The engine of the {self.make} {self.model} is now off.")
```

```
# Creating objects (instances) of the Car class
car1 = Car("Toyota", "Corolla", 2020)
car2 = Car("Ford", "Mustang", 2022)

# Calling methods on car1 object
car1.display_info()
car1.start_engine()
car1.stop_engine()

# Calling methods on car2 object
car2.display_info()
car2.start_engine()
car2.stop_engine()
```

Output :

Car Information: 2020 Toyota Corolla

The engine of the Toyota Corolla is now running.

The engine of the Toyota Corolla is now off.

Car Information: 2022 Ford Mustang

The engine of the Ford Mustang is now running.

The engine of the Ford Mustang is now off.

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 12

Date of Performance: **Date of Completion:**

Title: Implement a program to demonstrate Array.

Code:

```
# Import the array module
import array as arr

# Using an array
print("Using an array:")  # Indicate the section for arrays
my_array = arr.array('i', [1, 2, 3, 4, 5])  # 'i' indicates the type is integer
print("Original Array:", my_array)  # Display the original array

# Adding an element
my_array.append(6)  # Append 6 to the end of the array
print("Array after appending 6:", my_array)  # Display the array after appending

# Traversing the array
print("Traversing the array:")  # Indicate the traversal operation for the array
for element in my_array:  # Loop through each element in the array
    print(element)  # Print each element one by one

# Insertion at a given index
insert_index = 2  # Define the index to insert at
my_array.insert(insert_index, 10)  # Insert 10 at index 2
print(f"Array after inserting 10 at index {insert_index}:", my_array)  # Display the
array after insertion
```

```
# Updating an element at a given index
update_index = 1  # Define the index to update
my_array[update_index] = 20  # Update the element at index 1 to 20
print(f"Array after updating index {update_index} to 20:", my_array)  # Display the array after updating

# Deleting an element using remove() (by value)
my_array.remove(4)  # Remove the first occurrence of 4
print("Array after removing 4:", my_array)  # Display the array after removal

# Deleting an element using pop() (by index)
pop_index = 3  # Define the index to remove
my_array.pop(pop_index)  # Remove the element at index 3
print(f"Array after popping element at index {pop_index}:", my_array)  # Display the array after popping

# Length of the array
print("Length of the array:", len(my_array))  # Display the length of the array
```

Output:

Using an array:

Original Array: array('i', [1, 2, 3, 4, 5])

Array after appending 6: array('i', [1, 2, 3, 4, 5, 6])

Traversing the array:

1

2

3

4

5

6

Array after inserting 10 at index 2: array('i', [1, 2, 10, 3, 4, 5, 6])

Array after updating index 1 to 20: array('i', [1, 20, 10, 3, 4, 5, 6])

Array after removing 4: array('i', [1, 20, 10, 3, 5, 6])

Array after popping element at index 3: array('i', [1, 20, 10, 5, 6])

Length of the array: 5

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 13

Date of Performance: **Date of Completion:**

Title: Implement a program to create, import and use package and modules.

Code:

mypackage/module1.py

Define a function greet() that takes a name and returns a greeting.

```
def greet(name):
    return f"Hello, {name}!"
```

mypackage/module2.py

Define two functions add() and subtract() to perform basic arithmetic.

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

mypackage/__init__.py

This file initializes the package. You can define imports for easier access to module functions:

```
from .module1 import greet
from .module2 import add, subtract
```

main.py

```
# Import the entire package
import mypackage

# Import specific functions or modules
from mypackage import module1
from mypackage.module2 import subtract

# Using functions from the package
name = "Alice"
print(mypackage.greet(name))  # Access `greet()` from __init__.py
print(module1.greet("Bob"))  # Access `greet()` directly from module1

# Arithmetic operations
result1 = mypackage.add(10, 5)  # Access `add()` from __init__.py
result2 = subtract(10, 5)       # Access `subtract()` directly from module2

print("Addition Result:", result1)
print("Subtraction Result:", result2)
```

Output :

When you run main.py, you should see this output:

Hello, Alice!
Hello, Bob
Addition Result: 15
Subtraction Result: 5

Explaination

step-by-step explanation for creating, importing, and using Python packages and modules:

Step 1: Understand Modules and Packages

- A **module** is a single Python file (.py) that contains definitions like functions, classes, or variables.
- A **package** is a directory of modules with a special __init__.py file to indicate that it's a package. This allows the directory to be importable.

Step 2: Set Up the Directory Structure

Create a folder structure for your package. Let's name the folder mypackage. Inside this folder:

1. Create a file named __init__.py. This file tells Python that mypackage is a package. It can be empty or contain code to initialize the package.

2. Add two Python files (module1.py and module2.py) with some code inside.

Here's the structure:

```
mypackage/
   __init__.py
   module1.py
   module2.py
main.py
```

Step 3: Write Code for the Modules

Add code to your modules to define functions or classes.

mypackage/module1.py

Define a function greet() that takes a name and returns a greeting.

```
def greet(name):
   return f"Hello, {name}!"
```

mypackage/module2.py

Define two functions add() and subtract() to perform basic arithmetic.

```
def add(a, b):
   return a + b


def subtract(a, b):
   return a - b
```

mypackage/__init__.py

This file initializes the package. You can define imports for easier access to module functions:

```
from .module1 import greet
from .module2 import add, subtract
```

Step 4: Create a Main Script

Create a script main.py in the same folder as mypackage/. This script will import and use your package.

main.py

```
# Import the entire package
import mypackage

# Import specific functions or modules
from mypackage import module1
from mypackage.module2 import subtract

# Using functions from the package
name = "Alice"
print(mypackage.greet(name))  # Access `greet()` from __init__.py
print(module1.greet("Bob"))  # Access `greet()` directly from module1

# Arithmetic operations
result1 = mypackage.add(10, 5)  # Access `add()` from __init__.py
result2 = subtract(10, 5)       # Access `subtract()` directly from module2

print("Addition Result:", result1)
print("Subtraction Result:", result2)
```

Step 5: Run the Program

1. Save all files in the specified structure.
2. Open a terminal or command prompt and navigate to the folder containing main.py.
3. Run the program:

   python main.py

Step 6: Observe the Output

When you run main.py, you should see this output:

Hello, Alice!
Hello, Bob
Addition Result: 15
Subtraction Result: 5

Step 7: Explanation of Code

1. **mypackage/:** The folder acts as a package due to the __init__.py file.
2. **Modules:**
   - module1.py has a function greet().
   - module2.py has functions add() and subtract().
3. **Importing:**
   - mypackage.greet is imported via __init__.py.
   - module1.greet and module2.subtract are directly imported in main.py.
4. **Usage:**
   - Functions from the package can be accessed either directly from their modules or from the package itself.

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 14

Date of Performance: **Date of Completion:**

Title: Implement a program based on Writing Text Files, Appending Text to a File, Reading Text Files.

.

Code:

```
# Writing data to the file
file = open("myfile.txt", "w")
L = ["This is DSA \n", "This is Python \n", "This is Java \n"]

# Writing a line and then multiple lines from the list L
file.write("Hello There \n")
file.writelines(L)
file.close()

# Reading data from the file
file = open("myfile.txt", "r+")
print("Output of the read() function is:")
print(file.read())
print()

# Moving the file pointer back to the beginning of the file
file.seek(6)

print("Output of the readline() function is:")
print(file.readline())
print()
```

```
file.seek(0)

# Showing the difference between read() and readline()
print("Output of read(10) function is:")
print(file.read(10))  # Reads only the first 12 characters
print()

file.seek(0)

print("Output of readline(8) function is:")
print(file.readline(8))  # Reads only the first 8 characters of the first line
print()

file.seek(14)

print("Output of readlines() function is:")
print(file.readlines())  # Reads all lines into a list
print()
file.close()

# Appending data to the file
file = open("myfile.txt", "a")
file.write("This is an appended line.\n")
file.write("Another line added to the file.\n")
file.close()

# Reading the file again to verify appended content
file = open("myfile.txt", "r")
```

```
print("Content after appending new lines:")
print(file.read())
file.close()
```

Output:

```
Output of the read() function is:
Hello There
This is DSA
This is Python
This is Java

Output of the readline() function is:
There


Output of read(10) function is:
Hello Ther

Output of readline(8) function is:
Hello Th
```

Output of readlines() function is:

['This is DSA \n', 'This is Python \n', 'This is Java \n']

Content after appending new lines:

Hello There

This is DSA

This is Python

This is Java

This is an appended line.

Another line added to the file.

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 15

Date of Performance: **Date of Completion:**

Title: Implement a program based on paths and directories, file information, naming, moving, copying, and removing files.

 # Theory for Create File and Folder,File Information and copy file

The code demonstrates how to create a source folder and file, display their details,
and copy the file and folder to a destination folder.

How It Works

1. The script creates a folder (source_folder) and a text file (sample_file.txt) if they don't exist.
2. It displays metadata (path, size, last modified time) of the source folder and file.
3. It ensures the destination folder (destination_folder) exists.
4. It copies the file and the folder:
    - The file is copied to destination_folder.
    - The folder is recursively copied to destination_folder/copied_source_folder.

Code :

```
import shutil
from pathlib import Path

# Define paths

src_folder = Path("source_folder")
dst_folder = Path("destination_folder")

src_file = src_folder / "sample_file.txt"

# Ensure source folder and file exist; create them if they don't
src_folder.mkdir(parents=True, exist_ok=True)
if not src_file.exists():
    src_file.write_text("This is a sample file.")  # Create a sample file

# Display file and folder information before copying
print(f"\nSource Folder Information:")
print(f"Path: {src_folder}")
print(f"Is it a directory? {'Yes' if src_folder.is_dir() else 'No'}")
print(f"Files in Source Folder: {[file.name for file in src_folder.iterdir()]}")

if src_file.exists():
    print(f"\nSource File Information:")
    print(f"Path: {src_file}")
    print(f"Size: {src_file.stat().st_size} bytes")
    print(f"Last Modified: {src_file.stat().st_mtime}")

# Ensure destination folder exists
```

```
dst_folder.mkdir(parents=True, exist_ok=True)

# Copy file and folder
shutil.copy(src_file, dst_folder)  # Copy file to destination folder
shutil.copytree(src_folder, dst_folder / "copied_source_folder",
dirs_exist_ok=True)  # Copy folder

print("\nStep 1: File and folder copied successfully.")
```

Output:

```
Source Folder Information:
Path: source_folder
Is it a directory? Yes
Files in Source Folder: ['sample_file.txt']

Source File Information:
Path: source_folder\sample_file.txt
Size: 22 bytes
Last Modified: 1732015300.9169207

Step 1: File and folder copied successfully.
```

 # Theory for renaming a file :

This block of code is responsible for renaming a file that has already been copied to the destination_folder.

How It Works

1. **Before Renaming**:
    - In the destination_folder, the file sample_file.txt exists (from Step 1 where it was copied).
2. **During Renaming**:
    - The file sample_file.txt is renamed to renamed_sample_file.txt in the same folder (destination_folder).
    - This operation only changes the file name, not its location or content.
3. **After Renaming**:
    - The file sample_file.txt no longer exists in the destination_folder.
    - Instead, there is now a file named renamed_sample_file.txt.

Example

- **Before Execution**:
    - destination_folder/sample_file.txt
- **After Execution**:
    - destination_folder/renamed_sample_file.txt

Code :

```
# Define paths for renaming
dst_folder = Path("destination_folder")
copied_file = dst_folder / "sample_file.txt"
renamed_file = dst_folder / "renamed_sample_file.txt"

# Rename file
copied_file.rename(renamed_file)

print("Step 2: File renamed successfully.")
```

Output:

Step 2: File renamed successfully.

Theory for moving a renamed file:

This block of code is responsible for **moving a renamed file** from the destination_folder to a new folder called move_destination_folder.

How It Works

1. **Before Moving**:
    - The file renamed_sample_file.txt exists in the destination_folder.
2. **During Moving**:
    - The script ensures that the target folder move_destination_folder exists (creating it if necessary).
    - The file is then moved from destination_folder to move_destination_folder.
3. **After Moving**:
    - The file no longer exists in the destination_folder.
    - It now resides in the move_destination_folder.

Example

- **Before Execution**:
    - destination_folder/renamed_sample_file.txt
    - move_destination_folder may or may not exist.
- **After Execution**:
    - The file renamed_sample_file.txt is no longer in destination_folder.
    - It has been moved to move_destination_folder/renamed_sample_file.txt.

Code :

```
import shutil
from pathlib import Path

# Define paths for moving
dst_folder = Path("destination_folder")
move_folder = Path("move_destination_folder")  # New folder for moving the file
renamed_file = dst_folder / "renamed_sample_file.txt"

# Create new folder for moving if it doesn't exist
move_folder.mkdir(parents=True, exist_ok=True)

# Move file
shutil.move(str(renamed_file), move_folder / renamed_file.name)

print("Step 3: File moved to the new folder successfully.")
```

Output:

```
Step 3: File moved to the new folder successfully.
```

Theory for remove a file (renamed_sample_file.txt):

This block of code demonstrates how to **remove a file** (renamed_sample_file.txt) from the move_destination_folder directory.

How It Works

1. **Before Removal**:
    - The file renamed_sample_file.txt exists in the move_destination_folder.
2. **During Removal**:
    - The unlink() method is called on the file's path, deleting it from the filesystem.
3. **After Removal**:
    - The file renamed_sample_file.txt no longer exists in the move_destination_folder.

Example

- **Before Execution**:
    - Folder: move_destination_folder
    - File: renamed_sample_file.txt
- **After Execution**:
    - Folder: move_destination_folder
    - The file renamed_sample_file.txt has been deleted.

Code :

```
from pathlib import Path

# Define paths for removal
move_folder = Path("move_destination_folder")
file_to_remove = move_folder / "renamed_sample_file.txt"



# Remove file
file_to_remove.unlink()  # Delete the file

print("Step 4: File removed successfully.")
```

Output:

```
Step 4: File removed successfully.
```

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 16

Date of Performance: **Date of Completion:**

Title: Implement a program Based on Regular Expression and its functions.

Code:

```
import re

# Example text
text = "The quick brown fox jumps over the lazy dog 12345. Let's test: 6789."

# 1. Using re.match() to check if the string starts with 'The'
match = re.match(r'The', text)
if match:
    print(f"Match found: {match.group()}")
else:
    print("No match at the beginning of the string.")

# 2. Using re.search() to find the first occurrence of a pattern anywhere in the string
search = re.search(r'\d+', text)  # Search for one or more digits
if search:
    print(f"First match found: {search.group()}")
else:
    print("No digits found.")

# 3. Using re.findall() to find all occurrences of a pattern
findall = re.findall(r'\b\w+\b', text)  # Find all words in the string
```

```
print(f"All words found: {findall}")

# 4. Using re.sub() to replace all digits with an asterisk
replaced_text = re.sub(r'\d+', '*', text)
print(f"Text after replacing digits: {replaced_text}")

# 5. Using re.split() to split the text by spaces
split_text = re.split(r'\s+', text)
print(f"Text split by spaces: {split_text}")

# 6. Using re.finditer() to find all occurrences of a pattern and return match objects
finditer = re.finditer(r'\d+', text)
print("Matches using finditer():")
for match in finditer:
    print(f"Found {match.group()} at position {match.start()}")

# 7. Using re.fullmatch() to check if the entire string matches a pattern
fullmatch = re.fullmatch(r'The.*dog.*', text)
if fullmatch:
    print(f"Full match found: {fullmatch.group()}")
else:
    print("No full match for the pattern.")

# 8. Using re.subn() to replace all digits with an asterisk and return the number of replacements
subn_result = re.subn(r'\d+', '*', text)
print(f"Text after replacing digits with subn: {subn_result[0]}")
print(f"Number of replacements: {subn_result[1]}")
```

```
# 9. Using re.escape() to escape special characters in a string
escaped_string = re.escape("Hello. How are you? $%^&*()")
print(f"Escaped string: {escaped_string}")

# 10. Using re.compile() to compile a regular expression pattern for reuse
pattern = re.compile(r'\b\w+\b')  # Compile a pattern to match words
compiled_findall = pattern.findall(text)
print(f"All words found using compiled pattern: {compiled_findall}")
```

Output:

Match found: The

First match found: 12345

All words found: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '12345', 'Let', 's', 'test', '6789']

Text after replacing digits: The quick brown fox jumps over the lazy dog *. Let's test: *.

Text split by spaces: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '12345.', "Let's", 'test:', '6789.']

Matches using finditer():

Found 12345 at position 44

Found 6789 at position 63

Full match found: The quick brown fox jumps over the lazy dog 12345. Let's test: 6789.

Text after replacing digits with subn: The quick brown fox jumps over the lazy dog *. Let's test: *.

Number of replacements: 2

Escaped string: Hello\.\ How\ are\ you\?\ \$%\^\&*\(\)

All words found using compiled pattern: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '12345', 'Let', 's', 'test', '6789']

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 17

Date of Performance: **Date of Completion:**

Title: Implement a program creating Types of GUI Widgets, Resizing, and Configuring Options.

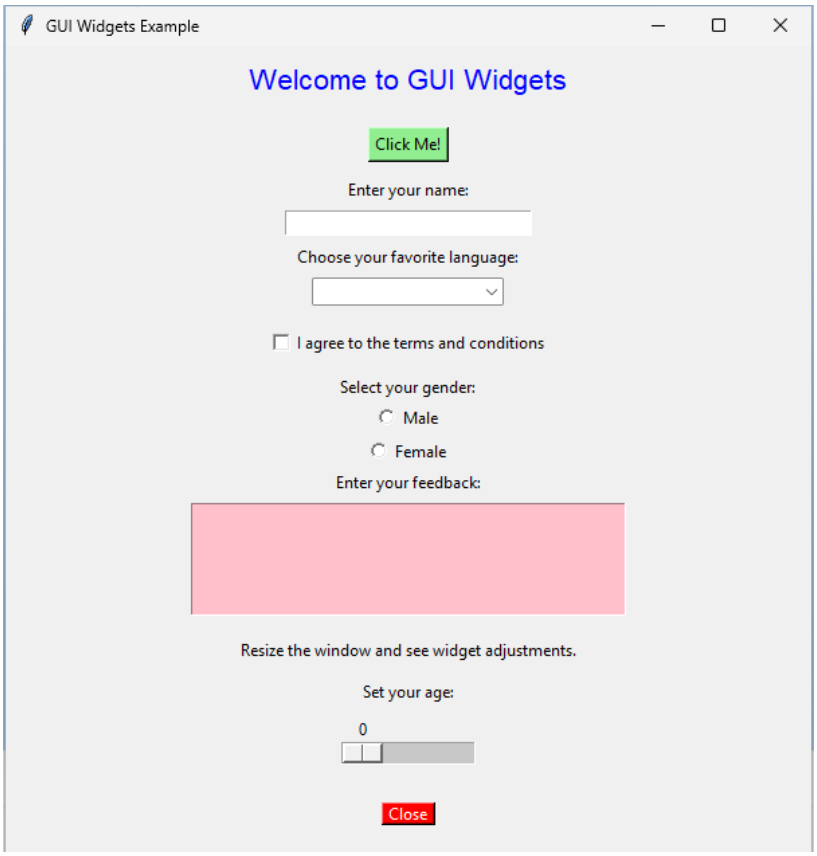
Code:

```
import tkinter as tk
from tkinter import ttk

# Create the main window
root = tk.Tk()
root.title("GUI Widgets Example")
root.geometry("600x600")  # Set the window size

# Label Widget
label = tk.Label(root, text="Welcome to GUI Widgets", font=("Arial", 16),
fg="blue")
label.pack(pady=10)  # Add padding for better layout

# Button Widget
def on_button_click():
    label.config(text="Button Clicked!")

button = tk.Button(root, text="Click Me!", command=on_button_click,
bg="lightgreen")
button.pack(pady=10)
```

```
# Entry Widget
entry_label = tk.Label(root, text="Enter your name:")
entry_label.pack()

entry = tk.Entry(root, width=30)
entry.pack(pady=5)

# Dropdown Menu (Combobox)
dropdown_label = tk.Label(root, text="Choose your favorite language:")
dropdown_label.pack()

languages = ["Python", "Java", "C++", "JavaScript"]
dropdown = ttk.Combobox(root, values=languages)
dropdown.pack(pady=5)

# Checkbox Widget
checkbox_var = tk.BooleanVar()
checkbox = tk.Checkbutton(root, text="I agree to the terms and conditions",
variable=checkbox_var)
checkbox.pack(pady=10)

# Radio Button Widget
radio_label = tk.Label(root, text="Select your gender:")
radio_label.pack()

gender_var = tk.StringVar(value="None")
radio1 = tk.Radiobutton(root, text="Male", variable=gender_var, value="Male")
radio2 = tk.Radiobutton(root, text="Female", variable=gender_var,
value="Female")
radio1.pack()
```

```
radio2.pack()

# Text Widget
text_label = tk.Label(root, text="Enter your feedback:")
text_label.pack()

text = tk.Text(root, height=5, width=40, bg="pink")
text.pack(pady=5)

# Resizing Widgets
resize_label = tk.Label(root, text="Resize the window and see widget
adjustments.")
resize_label.pack(pady=10)

# Slider Widget (Scale)
slider_label = tk.Label(root, text="Set your age:")
slider_label.pack()

slider = tk.Scale(root, from_=0, to=100, orient="horizontal")
slider.pack(pady=5)

# Close Button
close_button = tk.Button(root, text="Close", command=root.destroy, bg="red",
fg="white")
close_button.pack(pady=20)

# Run the application
root.mainloop()
```

Output:



Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 18

Date of Performance: **Date of Completion:**

Title: Implement a program Creating Layouts, Packing Order, Controlling Widget Appearances.

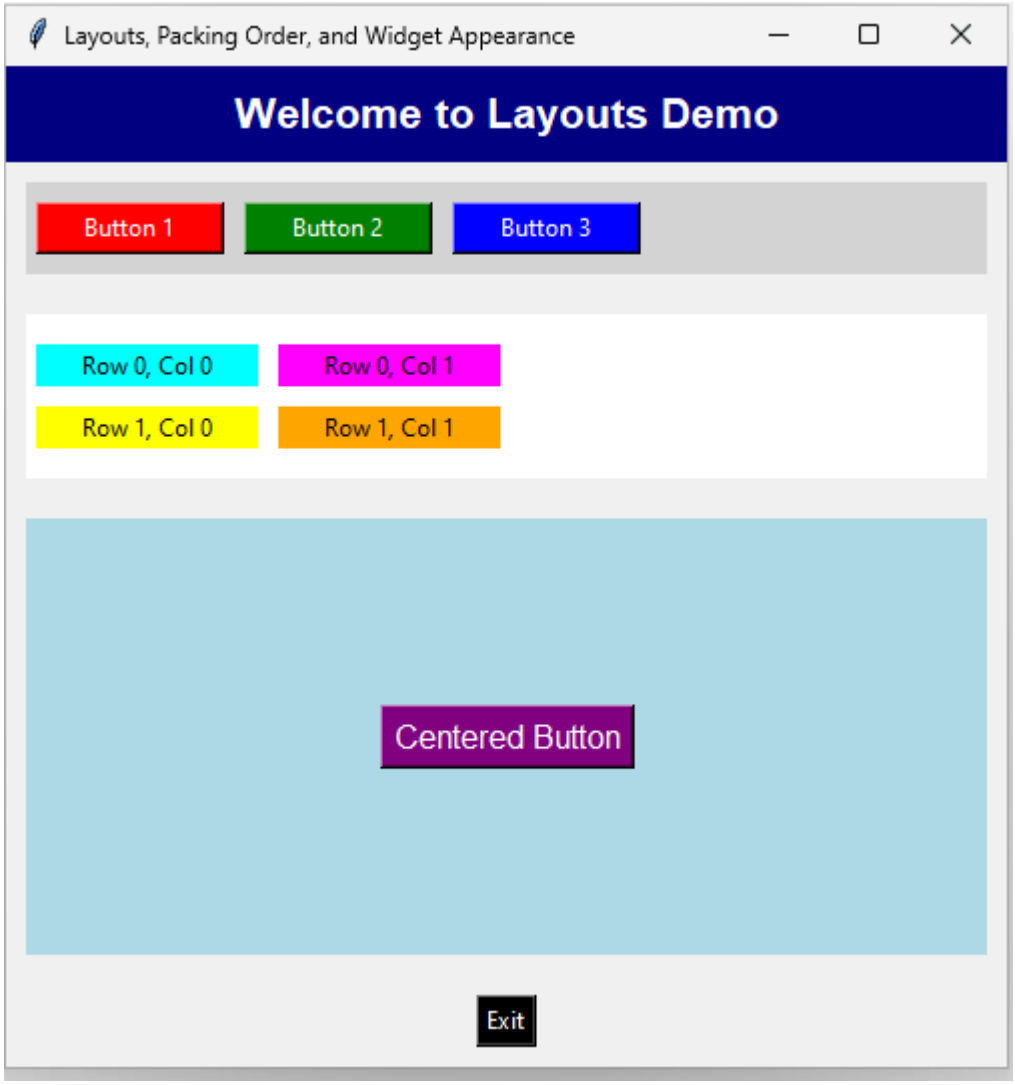
Code:

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Layouts, Packing Order, and Widget Appearance")
root.geometry("500x500")  # Set window size

# Header Label - Appearance customization
header = tk.Label(
    root,
    text="Welcome to Layouts Demo",
    font=("Helvetica", 16, "bold"),
    bg="navy",
    fg="white",
    padx=10,
    pady=10
)
header.pack(fill="x")  # Pack at the top, fill horizontally

# Frame 1: Packing Order Demonstration
frame1 = tk.Frame(root, bg="lightgray", pady=10)
frame1.pack(fill="x", padx=10, pady=10)
```

```
btn1 = tk.Button(frame1, text="Button 1", bg="red", fg="white", width=12)
btn2 = tk.Button(frame1, text="Button 2", bg="green", fg="white", width=12)
btn3 = tk.Button(frame1, text="Button 3", bg="blue", fg="white", width=12)

btn1.pack(side="left", padx=5)
btn2.pack(side="left", padx=5)
btn3.pack(side="left", padx=5)

# Frame 2: Grid Layout Demonstration
frame2 = tk.Frame(root, bg="white", pady=10)
frame2.pack(fill="x", padx=10, pady=10)

label1 = tk.Label(frame2, text="Row 0, Col 0", bg="cyan", width=15)
label2 = tk.Label(frame2, text="Row 0, Col 1", bg="magenta", width=15)
label3 = tk.Label(frame2, text="Row 1, Col 0", bg="yellow", width=15)
label4 = tk.Label(frame2, text="Row 1, Col 1", bg="orange", width=15)

label1.grid(row=0, column=0, padx=5, pady=5)
label2.grid(row=0, column=1, padx=5, pady=5)
label3.grid(row=1, column=0, padx=5, pady=5)
label4.grid(row=1, column=1, padx=5, pady=5)

# Frame 3: Place Layout Demonstration
frame3 = tk.Frame(root, bg="lightblue", pady=20)
frame3.pack(fill="both", expand=True, padx=10, pady=10)

resizable_button = tk.Button(frame3, text="Centered Button", bg="purple",
fg="white", font=("Arial", 12))
resizable_button.place(relx=0.5, rely=0.5, anchor="center")
```

```
# Footer Button - Close Application
footer_button = tk.Button(root, text="Exit", command=root.destroy, bg="black",
fg="white")
footer_button.pack(pady=10)

# Run the application
root.mainloop()
```

Output:



Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 19

Date of Performance: **Date of Completion:**

Title: Implement a program based on Text Processing, Searching for Files.

Code:

```
import os

# User inputs directory path and the search word
directory_path = input("Enter the directory path to search for text files: ")
search_word = input("Enter the word/phrase to search for in the text files: ")

# Check if the provided directory exists
if os.path.isdir(directory_path):
    print(f"\nSearching for the word '{search_word}' in the text files within directory: {directory_path}\n")

    # Loop through all the files in the specified directory
    for filename in os.listdir(directory_path):
        # Check if the file is a text file (i.e., ends with .txt)
        if filename.endswith(".txt"):
            file_path = os.path.join(directory_path, filename)

            try:
                # Open the text file for reading
                with open(file_path, 'r', encoding='utf-8') as file:
                    # Read the content of the file
```

```
                file_content = file.read()

                # Perform a case-insensitive search for the search word
                if search_word.lower() in file_content.lower():  # Case-insensitive
search
                    print(f"Found '{search_word}' in file: {filename}")
                else:
                    print(f"'{search_word}' not found in file: {filename}")

            except Exception as e:
                # Handle errors such as file access issues
                print(f"Error reading file {filename}: {e}")
else:
    print("The provided directory path does not exist.")
```

Output:

Enter the directory path to search for text files: D:\Users\LENOVO\Desktop\MCA
Enter the word/phrase to search for in the text files: Python

Searching for the word 'Python' in the text files within directory:
D:\Users\LENOVO\Desktop\MCA

Found 'Python' in file: File1.txt
'Python' not found in file: File2.txt
Found 'Python' in file: File3.txt

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 20

Date of Performance: **Date of Completion:**

Title: Implement a program to demonstrate HTML Parsing.

Code:

```
from bs4 import BeautifulSoup
def parse_html(html_content):
    """
    Demonstrates HTML parsing by extracting the title, links, and text content.
    """
    soup = BeautifulSoup(html_content, 'html.parser')

    # Extract and display the title
    title = soup.title.string if soup.title else "No title found"
    print(f"Title: {title}")

    # Extract and display all links
    print("\nLinks found:")
    for link in soup.find_all('a', href=True):
        print(f"  {link.text.strip()} -> {link['href']}")

    # Extract and display paragraph texts
    print("\nParagraphs:")
    for para in soup.find_all('p'):
        print(f"  {para.text.strip()}")

    # Extract and display all headings (h1, h2, h3, etc.)
```

```
    print("\nHeadings:")
    for i in range(1, 7):  # h1 to h6
        for heading in soup.find_all(f"h{i}"):
            print(f"  {heading.name}: {heading.text.strip()}")

def main():
    # Input: HTML content (could also be read from a file)
    html_content = """
    <html>
      <head>
        <title>HTML Parsing Example</title>
      </head>
      <body>
        <h1>Main Heading</h1>
        <p>This is a paragraph of text.</p>
        <a href="https://example.com">Visit Example</a>
        <h2>Subheading</h2>
        <p>Another paragraph with <a href="https://another.com">another
link</a>.</p>
      </body>
    </html>
    """
    # Parse the HTML content
    parse_html(html_content)

if __name__ == "__main__":
    main()
```

Output:

Title: HTML Parsing Example

Links found:

Visit Example -> https://example.com

another link -> https://another.com

Paragraphs:

This is a paragraph of text.

Another paragraph with another link.

Headings:

h1: Main Heading

h2: Subheading

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 21

Date of Performance: **Date of Completion:**

Title: Implement a program based on using DBM - Creating and Accessing Persistent Dictionaries.

Code:

```
import dbm

# Function to create and add key-value pairs to a DBM database
def create_dbm(db_name):
    with dbm.open(db_name, 'c') as db:
        # Adding key-value pairs
        db['name'] = 'Alice'
        db['age'] = '30'
        db['city'] = 'New York'
        print("Database created and entries added.")

# Function to read and access values from a DBM database
def access_dbm(db_name):
    with dbm.open(db_name, 'r') as db:
        # Access and display all key-value pairs
        print("Contents of the DBM database:")
        for key in db.keys():
            print(f"{key.decode()}: {db[key].decode()}")

        # Access specific values
        print("\nAccessing specific values:")
        name = db.get('name'.encode()).decode()
```

```
        print(f"Name: {name}")

# Function to update a value in the DBM database
def update_dbm(db_name, key, value):
    with dbm.open(db_name, 'w') as db:
        db[key] = value
        print(f"Updated {key.decode()} to {value.decode()}")

# Function to delete an entry in the DBM database
def delete_entry_dbm(db_name, key):
    with dbm.open(db_name, 'w') as db:
        del db[key]
        print(f"Deleted entry with key: {key.decode()}")

# Main program
db_name = "sample_db"
create_dbm(db_name)  # Create and populate DBM
access_dbm(db_name)  # Access and display contents

# Update a value
update_dbm(db_name, b'city', b'Los Angeles')
access_dbm(db_name)  # Verify update

# Delete an entry
delete_entry_dbm(db_name, b'age')
access_dbm(db_name)  # Verify deletion
```

Output:

Database created and entries added.

Contents of the DBM database:

name: Alice

city: New York

age: 30

Accessing specific values:

Name: Alice

Updated city to Los Angeles

Contents of the DBM database:

name: Alice

city: Los Angeles

age: 30

Accessing specific values:

Name: Alice

Deleted entry with key: age

Contents of the DBM database:

name: Alice

city: Los Angeles

Accessing specific values:

Name: Alice

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 22

Date of Performance: **Date of Completion:**

Title: Implement a program based on using Relational Database - Writing SQL Statements, Defining Tables, Setting Up a Database.(Transactions and Committing the Results.)

Code:

```
import sqlite3

# Step 1: Connect to SQLite database (or create one if it doesn't exist)
connection = sqlite3.connect("bca_students.db")
cursor = connection.cursor()

# Step 2: Define tables
def create_tables():
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Students (
            student_id INTEGER PRIMARY KEY,
            name TEXT NOT NULL,
            age INTEGER,
            department TEXT
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Subjects (
            subject_id INTEGER PRIMARY KEY,
            subject_name TEXT NOT NULL,
            credits INTEGER
```

```
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Registrations (
            registration_id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id INTEGER,
            subject_id INTEGER,
            registration_date TEXT,
            FOREIGN KEY(student_id) REFERENCES Students(student_id),
            FOREIGN KEY(subject_id) REFERENCES Subjects(subject_id)
        )
    """)
    print("Tables created successfully.")

# Step 3: Insert data
def insert_data():
    try:
        # Insert into Students table
        cursor.execute("INSERT INTO Students (student_id, name, age, department)
VALUES (?, ?, ?, ?)",
                       (1, "John Doe", 20, "Computer Applications"))
        cursor.execute("INSERT INTO Students (student_id, name, age, department)
VALUES (?, ?, ?, ?)",
                       (2, "Jane Smith", 21, "Computer Applications"))

        # Insert into Subjects table
        cursor.execute("INSERT INTO Subjects (subject_id, subject_name, credits)
VALUES (?, ?, ?)",
                       (101, "Database Management Systems", 4))
        cursor.execute("INSERT INTO Subjects (subject_id, subject_name, credits)
```

```
VALUES (?, ?, ?)",
                (102, "Operating Systems", 3))

    # Insert into Registrations table
    cursor.execute("INSERT INTO Registrations (student_id, subject_id,
registration_date) VALUES (?, ?, ?)",
                (1, 101, "2024-11-21"))
    cursor.execute("INSERT INTO Registrations (student_id, subject_id,
registration_date) VALUES (?, ?, ?)",
                (2, 102, "2024-11-21"))

    # Commit transaction
    connection.commit()
    print("Data inserted successfully and committed.")
  except Exception as e:
    # Rollback transaction in case of error
    connection.rollback()
    print("Error occurred during insertion:", e)

# Step 4: Display data
def display_data():
  print("\nStudents Table:")
  cursor.execute("SELECT * FROM Students")
  for row in cursor.fetchall():
    print(row)

  print("\nSubjects Table:")
  cursor.execute("SELECT * FROM Subjects")
  for row in cursor.fetchall():
    print(row)
```

```
    print("\nRegistrations Table:")
    cursor.execute("""
        SELECT r.registration_id, s.name AS student_name, sub.subject_name,
r.registration_date
        FROM Registrations r
        JOIN Students s ON r.student_id = s.student_id
        JOIN Subjects sub ON r.subject_id = sub.subject_id
    """)
    for row in cursor.fetchall():
        print(row)

# Main Execution
create_tables()
insert_data()
display_data()

# Close the database connection
connection.close()
```

Output:

Tables created successfully.

Error occurred during insertion: UNIQUE constraint failed: Students.student_id

Students Table:

(1, 'John Doe', 20, 'Computer Applications')

(2, 'Jane Smith', 21, 'Computer Applications')

Subjects Table:

(101, 'Database Management Systems', 4)

(102, 'Operating Systems', 3)

Registrations Table:

(1, 'John Doe', 'Database Management Systems', '2024-11-21')

(2, 'Jane Smith', 'Operating Systems', '2024-11-21')

Submitted By : **Checked By :**

Sign :

Name : **Asst. Prof. Sapana A. Fegade**

Roll No :