DRONACHARYA
College of Engineering

LABORATORY MANUAL

B.Tech. Semester- II

PYTHON LAB
Subject code: CSE-106P

Prepared by:	**Checked by:**	**Approved by:**
Prof. Sukrati Chaturvedi	Prof. Megha Goel	Name : Prof. (Dr.) Isha Malhotra
Sign.: ……	**Sign.: ……**	**Sign.: ……**

DEPARTMENT OF APPLIED SCIENCE & HUMANITIES
DRONACHARYA COLLEGE OF ENGINEERING
KHENTAWAS, FARRUKH NAGAR, GURUGRAM (HARYANA)

TABLE OF CONTENTS

Vision and Mission of the Institute

Vision:

"Empowering human values and advanced technical education to navigate and address global challenges with excellence".

Mission:

- **M1 -** Seamlessly integrate human values with advanced technical education.
- **M2 -** Supporting the cultivation of a new generation of innovators who are not only skilled but also ethically responsible.
- **M3 -** Inspire global citizens who are equipped to create positive and sustainable impact, driving progress towards a more inclusive and harmonious world.

VISION AND MISSION OF THE DEPARTMENT

Vision

- To establish a strong foundation for first-year engineering students, aiming to equip them with the skills to innovate and devise engineering solutions.

Mission

- **M1**: To develop a solid foundation of knowledge and hands on experience in budding technocrats, empowering them to apply scientific principles to address complex engineering challenges.
- **M2**: To provide education that fosters comprehension and collaboration between engineering and other core field of Applied Sciences.
- **M3**: To inculcate values and ethics in students and make them responsible citizens of India.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOS)

- **PEO1:** PEO1: To instill the basic principles of Applied Sciences to enable students learn technical subjects effectively.
- **PEO2:** To equip students with innovative skills that improve their practical understanding enabling them to solve real-world challenges effectively.
- **PEO3:** To enhance students' team-building skills and leadership qualities continuously through social, cultural, and environmental activities.

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

UNIVERSITY SYLLABUS

1. WAP to display your biodata.
2. WAP to swap two numbers using two methods.
3. WAP to determine whether the given number is even or odd.
4. WAP to determine whether the given number is Armstrong number.
5. WAP to print Fibonacci series upto n^{th} term.
6. WAP to take an integer input from user between 0-100 and print its name spelling in English using list.
7. WAP to convert hexadecimal number to binary using dictionary.
8. WAP to find sum and average of tuple elements.
9. WAP to take an integer input from user. Calculate and display its factorial using recursion.
10. WAP to implement a python class with a method to print area of a triangle using heron's formula using inheritance.
11. WAP to create a GUI for calculator.
12. WAP to implement a person class (Name, address, phone, Id) and derive two classes class teacher(additional TID, Course, faculty) and student Roll_no, course enrolled) from person class to show the use of Inheritance.
13. WAP to plot histogram of the following data.

10-15	15-20	20-25	25-30	30-35
5	6	9	8	2

14. WAP to draw a box-and-whisker plot for the data set {3, 7, 8, 5, 12, 14, 21, 13, 18}.
15. WAP to draw the Line Plot and Bar chart for the following data.

Elapsed time (s)	0	1	2	3	4	5	6
Speed(m/s)	0	3	7	12	20	30	45.6

COURSE OUTCOMES (COs)

Upon successful completion of the course, the students will acquire:
CO1: Basic understanding of Python syntax. Learn the fundamental syntax and structure of the Python programming language, including variables, data types, loops, conditionals and functions.

CO2: Proficiency in writing Python code. Gain the ability to write Python programs to solve simple to moderately complex problems.

CO3: Understanding of Object Oriented Programming (OOP). OOP is a fundamental concept in Python, and student will learn how to create classes, objects, and use inheritance to write more modular and reusable code.

CO4: Understanding of GUI concepts: Students will learn the basic concepts and principles of GUI design and development, including user interfaces, widgets, event handling, and layout management.

CO5: Introduction to data analysis and visualisation. Students will learn techniques to analyse and visualise data by plotting different graphs including bar graph, line plot, and box plot.

CO-PO Mapping

	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1				2	3			1	2	2	2	2
CO2			1		3			1	2	2	2	2
CO3				2	3			1	2	2	2	2
CO4	1			2	3			1	2	2	2	2
CO5			2	2	3			1	2	2	2	2
CO	**0.2**		**0.6**	**1.6**	**3**			**1**	**2**	**2**	**2**	**2**

CO-PSO Mapping

	PSO1	**PSO2**	**PSO3**
CO1		2	2
CO2		2	2
CO3		2	2
CO4		2	2
CO5		2	2
CO		**2**	**2**

COURSE OVERVIEW

Python lab introduces a comprehensive overview of the Python programming language, covering its syntax, data structures, control flow, object-oriented programming, file handling, modules and libraries, data analysis and visualization. Through practical exercises and projects, students gain proficiency in Python programming, learn to write clean and readable code, work with external libraries, handle files and databases, and analyse data. The course equips students with the skills and knowledge to solve real-world problems using Python.

LIST OF EXPERIMENTS MAPPED WITH COS

<table>
<tr><th>S.no.</th><th>List of
Experiments</th><th>Course
Outcome</th><th>Page no.</th></tr>
<tr><td>1</td><td>WAP to display your biodata.</td><td>CO1</td><td>1-2</td></tr>
<tr><td>2</td><td>WAP to swap two numbers using two methods</td><td>CO1</td><td>3-5</td></tr>
<tr><td>3</td><td>WAP to determine whether the given number is even or odd</td><td>CO2</td><td>6-7</td></tr>
<tr><td>4</td><td>WAP to determine whether the given number is Armstrong number</td><td>CO2</td><td>8-10</td></tr>
<tr><td>5</td><td>WAP to print Fibonacci series upto nth term</td><td>CO3</td><td>11-12</td></tr>
<tr><td>6</td><td>WAP to take an integer input from user between 0-100 and print its name spelling in English using list</td><td>CO3</td><td>13-15</td></tr>
<tr><td>7</td><td>WAP to convert hexadecimal number to binary using dictionary</td><td>CO3</td><td>16-18</td></tr>
<tr><td>8</td><td>WAP to find sum and average of tuple elements</td><td>CO3</td><td>19-20</td></tr>
<tr><td>9</td><td>WAP to take an integer input from user. Calculate and display its factorial using recursion</td><td>CO4</td><td>21-23</td></tr>
<tr><td>10</td><td>WAP to implement a python class with a method to print area of a triangle using heron’s formula using inheritance</td><td>CO3</td><td>24-26</td></tr>
<tr><td>11</td><td>WAP to create a GUI for calculator</td><td>CO4</td><td>27-30</td></tr>
<tr><td>12</td><td>WAP to implement a person class (Name, address, phone, Id) and derive two classes class teacher(additional TID, Course, faculty) and student Roll_no, course enrolled) from person class to show the use of Inheritance.</td><td>CO5</td><td>31-33</td></tr>
<tr><td>13</td><td>WAP to plot histogram of the following data.<table><tr><td>10-15</td><td>15-20</td><td>20-25</td><td>25-30</td><td>30-35</td></tr><tr><td>5</td><td>6</td><td>9</td><td>8</td><td>2</td></tr></table></td><td>CO4</td><td>34-36</td></tr>
<tr><td>14</td><td>WAP to draw a box-and-whisker plot for the data set {3, 7, 8, 5, 12, 14, 21, 13, 18}.</td><td>CO4</td><td>37-39</td></tr>
<tr><td>15</td><td>WAP to draw the Line Plot and Bar chart for the following data.<table><tr><td>Elapsed time (s)</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr><tr><td>Speed(m/s)</td><td>0</td><td>3</td><td>7</td><td>12</td><td>20</td><td>30</td><td>45.6</td></tr></table></td><td>CO5</td><td>40-43</td></tr>
</table>

DOs and DON'Ts

DOs

1. Login-on with your username and password.
2. Log off the Computer every time when you leave the Lab.
3. Arrange your chair properly when you are leaving the lab.
4. Put your bags in the designated area.
5. Ask permission to print.

DON'Ts

1. Do not share your username and password.
2. Do not remove or disconnect cables or hardware parts.
3. Do not personalize the computer setting.
4. Do not run programs that continue to execute after you log off.
5. Do not download or install any programs, games or music on computer in Lab.
6. Personal Internet use chat room for Instant Messaging (IM) and Sites is strictly prohibited.
7. No Internet gaming activities allowed.
8. Tea, Coffee, Water & Eatables are not allowed in the Computer Lab.

GENERAL SAFETY PRECAUTIONS

Precautions (In case of Injury or Electric Shock)

1. To break the victim with live electric source, use an insulator such as fire wood or plastic to break the contact. Do not touch the victim with bare hands to avoid the risk of electrifying yourself.

2. Unplug the risk of faulty equipment. If main circuit breaker is accessible, turn the circuit off.

3. If the victim is unconscious, start resuscitation immediately, use your hands to press the chest in and out to continue breathing function. Use mouth-to-mouth resuscitation if necessary.

4. Immediately call medical emergency and security. Remember! Time is critical; be best.

Precautions (In case of Fire)

1. Turn the equipment off. If power switch is not immediately accessible, take plug off.

2. If fire continues, try to curb the fire, if possible, by using the fire extinguisher or by covering it with a heavy cloth if possible isolate the burning equipment from the other surrounding equipment.

3. Sound the fire alarm by activating the nearest alarm switch located in the hallway.

4. Call security and emergency department immediately:

Emergency : Reception
Security : Main Gate

GUIDELINES TO STUDENTS FOR REPORT PREPARATION

All students are required to maintain a record of the experiments conducted by them. Guidelines for its preparation are as follows:-

1) All files must contain a title page followed by an index page. ***The files will not be signed by the faculty without an entry in the index page.***

2) Student's Name, Roll number and date of conduction of experiment must be written on all pages.

3) For each experiment, the record must contain the following

(i) Aim/Objective of the experiment

(ii) Pre-experiment work (as given by the faculty)

(iii) Lab assignment questions and their solutions

(iv) Test Cases (if applicable to the course)

(v) Results/ output

Note:

1. Students must bring their lab record along with them whenever they come for the lab.
2. Students must ensure that their lab record is regularly evaluated.

Lab Assessment Criteria

An estimated 10 lab classes are conducted in a semester for each lab course. These lab classes are assessed continuously. Each lab experiment is evaluated based on 5 assessment criteria as shown in following table. Assessed performance in each experiment is used to compute CO attainment as well as internal marks in the lab course.

Grading Criteria	Exemplary (4)	Competent (3)	Needs Improvement (2)	Poor (1)
AC1: Pre-Lab written work (this may be assessed through viva)	Complete procedure with underlined concept is properly written	Underlined concept is written but procedure is incomplete	Not able to write concept and procedure	Underlined concept is not clearly understood
AC2: Program Writing/ Modeling	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied, Program/solution written is readable	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied	Assigned problem is properly analyzed & correct solution designed	Assigned problem is properly analyzed
AC3: Identification & Removal of errors/ bugs	Able to identify errors/ bugs and remove them	Able to identify errors/ bugs and remove them with little bit of guidance	Is dependent totally on someone for identification of errors/ bugs and their removal	Unable to understand the reason for errors/ bugs even after they are explicitly pointed out
AC4:Execution & Demonstration	All variants of input /output are tested, Solution is well demonstrated and implemented concept is clearly explained	All variants of input /output are not tested, However, solution is well demonstrated and implemented concept is clearly explained	Only few variants of input /output are tested, Solution is well demonstrated but implemented concept is not clearly explained	Solution is not well demonstrated and implemented concept is not clearly explained
AC5:Lab Record Assessment	All assigned problems are well recorded with objective, design constructs and solution along with Performance analysis using all variants of input and output	More than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	Less than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	

LAB EXPERIMENTS

LAB EXPERIMENT 1

OBJECTIVE

WAP to display your biodata.

PRE-EXPERIMENT QUESTIONS

Q1. What are different data types in python?

Q2. What is the use of print function in python?

BRIEF DISCUSSION AND EXPLANATION

To create a program that displays a person's biodata, follow these steps:

1. Define variables: Begin by defining variables to store the individual's personal information, such as their name, age, address, and educational background.

```
name = "ABC"
age = 25
address = "123 Main Street, City"
education = "Bachelor's Degree in Computer Science"
```

2. Print the biodata: Utilize the `print()` function to showcase each piece of information on a separate line. Combine the information with appropriate labels using string concatenation or formatted strings.

```
print("Name: " + name)
print("Age: " + str(age))
print("Address: " + address)
print("Education: " + education)
```

3. Run the program: Save the file with a `.py` extension, such as `biodata.py`, and execute it using a Python interpreter. The program will display the person's biodata in the console.

Here's an example of how the complete program would look:

```
name = "ABC"
age = 25
address = "123 Main Street, City"
education = "Bachelor's Degree in Computer Science"

print("Name: " + name)
print("Age: " + str(age))
print("Address: " + address)
print("Education: " + education)
```

When the program is executed, it will output the person's biodata, with each piece of information on a separate line and labelled accordingly.

OUTPUT

```
Name: ABC
Age: 25
Address: 123 Main Street, City
Education: Bachelor's Degree in Computer Science
>>>
```

POST EXPERIMENT QUESTIONS

Q1. How do you define variables in python?
Q2. How do you save and compile a python file?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program displays the biodata of a person named "ChatGPT" (an AI language model) including their name, age, gender, occupation, skills, education details, and interests.
2. How is the biodata stored in this program?
    - The biodata is stored using variables. Each variable represents a different aspect of the biodata, such as name, age, gender, occupation, skills, education, and interests.
3. What data types are used to store different pieces of information in the program?
    - The data types used in the program include strings (for name, gender, occupation), integers (for age, year of education), lists (for skills and interests), and dictionaries (for education details).
4. How is the biodata displayed to the user?
    - The biodata is displayed using the print() function. Each piece of information is printed with a corresponding label to make it more readable.
5. What other information could be added to this biodata program?
    - Additional information that could be added to the program includes contact details (phone number, email address), address, work experience, projects, achievements, and references.

LAB EXPERIMENT 2

OBJECTIVE

WAP to swap two numbers using two methods.

PRE-EXPERIMENT QUESTIONS

1. What are the two methods you plan to use to swap the numbers?
2. Are the numbers to be swapped integers or floating-point numbers?

BRIEF DISCUSSION AND EXPLANATION

Python program that swaps two numbers using two different methods:

Method 1: Using a temporary variable
1. Start by defining two variables, `num1` and `num2`, to store the two numbers that need to be swapped.

2. Initialize a temporary variable, `temp`, and assign it the value of `num1`.

3. Assign the value of `num2` to `num1`.

4. Finally, assign the value of `temp` to `num2`.

5. Now, the values of `num1` and `num2` have been swapped.

Method 2: Without using a temporary variable
1. Define two variables, `num1` and `num2`, to store the two numbers that need to be swapped.

2. Perform the swapping operation using arithmetic operations.

```
num1 = 10
num2 = 20

# Method 1: Using a temporary variable
temp = num1
num1 = num2
num2 = temp

# Method 2: Without using a temporary variable
num1 = num1 + num2
num2 = num1 - num2
num1 = num1 - num2

print("After swapping:")
print("num1 =", num1)
```

```
print("num2 =", num2)
```

In both methods, the values of `num1` and `num2` will be swapped. The first method uses a temporary variable to hold the value temporarily during the swapping process. The second method performs the swapping operation without using an extra variable, using arithmetic operations to manipulate the values.

By running this program, you will see the output displaying the swapped values of `num1` and `num2`.

OUTPUT

```
========== RESTART: C:/Users/Jax
After swapping:
num1 = 10
num2 = 20
>>>
```

POST-EXPERIMENT QUESTIONS

1. What were the two methods you used to swap the numbers?

2. Did you allow user input for the numbers or were they predefined?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program swaps the values of two numbers using two different methods: one using a temporary variable and the other without using a temporary variable.
2. How are the two numbers initially stored in the program?
    - The two numbers are stored in variables **num1** and **num2**.
3. Explain Method 1: Using a temporary variable.
    - In this method, a temporary variable (**temp**) is used to store the value of **num1**. Then, the value of **num2** is assigned to **num1**, and finally, the value of **temp** (which was the original value of **num1**) is assigned to **num2**. This effectively swaps the values of the two numbers.
4. Explain Method 2: Without using a temporary variable (using arithmetic operations).
    - In this method, the values of **num1** and **num2** are swapped without using a temporary variable. The new value of **num1** is obtained by adding the original values of **num1** and **num2**. Then, the new value of **num2** is obtained by subtracting the original value of **num2** from the new value of **num1**. Finally, the new value of **num1** is obtained by subtracting the original value of **num2** from the new value of **num1**. This achieves the swapping of values.

5. Can you explain the output of the program for each method?

    - The output of the program displays the values of **num1** and **num2** before and after swapping using each method. The values are printed using the **print()** function.

6. Are there any limitations or restrictions when swapping numbers without a temporary variable?

    - Swapping numbers without a temporary variable using arithmetic operations may result in incorrect values or unexpected behavior if the numbers are large and their sum or difference exceeds the range of the data type used. It is important to consider the data type and potential overflow or underflow when using this method.

LAB EXPERIMENT 3

OBJECTIVE

WAP to determine whether the given number is even or odd.

PRE-EXPERIMENT QUESTIONS

1. Are there any additional requirements or restrictions for the program's implementation?
2. Should the program handle any specific error cases, such as non-numeric input?

BRIEF DISCUSSION AND EXPLANATION

Python program that determines whether a given number is even or odd:

1. Start by obtaining the input from the user, either by hardcoding a number or using input() to prompt the user for a number.

2. Use the modulus operator (%) to check if the remainder of dividing the number by 2 is equal to 0. If it is, the number is even. Otherwise, it is odd.

3. Use an if-else statement to display the appropriate message based on the result.

Here's an example of how the complete program would look:

```
# Obtain the input from the user
num = int(input("Enter a number: "))

# Check if the number is even or odd
if num % 2 == 0:
    print(num, "is even.")
else:
    print(num, "is odd.")
```

In this program, the user is prompted to enter a number. The program then uses the modulus operator to check if the number is divisible evenly by 2. If the remainder is 0, it is even, and the program displays an appropriate message. Otherwise, it is odd, and the program displays a different message.

By running this program and entering different numbers, you can determine whether they are even or odd.

OUTPUT

```
Enter a number:
6
6 is even.
>>>
============== RESTART: C:/Use
Enter a number: 5
5 is odd.
>>>
```

POST-EXPERIMENT QUESTIONS

1. How did you handle cases where the input was not a valid number?

2. Did you test the program with different input values? If so, did it produce the correct results?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program determines whether a given number is even or odd.
2. How is the number to be checked stored in the program?
    - The number to be checked is stored in the variable **num**.
3. How does the program determine whether the number is even or odd?
    - The program uses the modulo operator (**%**) to check if the remainder of **num** divided by 2 is zero. If the remainder is zero, it means the number is divisible by 2 and hence even. Otherwise, it is odd.
4. Can you explain the use of the if-else statement in the program?
    - The if-else statement is used to provide different instructions based on whether the condition **num % 2 == 0** is true or false. If the condition is true, the program prints that the number is even. Otherwise, it prints that the number is odd.
5. What would be the output of the program for the given number?
    - The output of the program for the given number (**24**) would be: "24 is even."
6. Is there any limitation or restriction to using this program?
    - This program works correctly for integer numbers. However, if the input is a floating-point number, it may not provide the expected result. It is important to ensure that the input is of the correct data type for accurate evaluation.

LAB EXPERIMENT 4

OBJECTIVE

WAP to determine whether the given number is Armstrong number.

PRE-EXPERIMENT QUESTIONS

1. Should the program take user input for the number or should it be predefined?
2. How should the program handle non-numeric inputs or invalid inputs?

BRIEF DISCUSSION AND EXPLANATION

Python program that determines whether a given number is an Armstrong number:

1. Start by obtaining the input from the user, either by hardcoding a number or using input() to prompt the user for a number.

2. Calculate the sum of the cubes of each digit in the number. To do this, you need to extract each digit from the number using modulus and division operations.

3. Compare the calculated sum with the original number. If they are equal, the number is an Armstrong number. Otherwise, it is not.

4. Use an if-else statement to display the appropriate message based on the result.

Here's an example of how the complete program would look:

python

```
# Obtain the input from the user
num = int(input("Enter a number: "))

# Calculate the sum of the cubes of each digit
temp = num
sum = 0
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp //= 10

# Check if the number is an Armstrong number
if num == sum:
    print(num, "is an Armstrong number.")
else:
```

```
print(num, "is not an Armstrong number.")
```

In this program, the user is prompted to enter a number. The program then uses a while loop to extract each digit from the number and calculate the sum of the cubes of these digits. After that, it compares the calculated sum with the original number to determine if it is an Armstrong number or not. Finally, it displays an appropriate message based on the result.

By running this program and entering different numbers, you can determine whether they are Armstrong numbers.

OUTPUT

```
Enter a number: 156
156 is not an Armstrong number.
>>>
============== RESTART: C:/Users/Sukrati
Enter a number: 153
153 is an Armstrong number.
>>>
```

POST-EXPERIMENT QUESTIONS

1. Did you allow user input for the number or was it predefined?

2. How did you handle cases where the input was not a valid number?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program determines whether a given number is an Armstrong number.
2. What is an Armstrong number?
    - An Armstrong number is a number that is equal to the sum of its digits raised to the power of the number of digits in it. For example, 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153.
3. How is the number to be checked stored in the program?
    - The number to be checked is stored in the variable **num**.
4. Can you explain the logic behind checking for an Armstrong number in this program?
    - The program uses a while loop to extract each digit of the number. Inside the loop, it calculates the sum of each digit raised to the power of the number of digits. If the sum is equal to the original number, it means the number is an Armstrong number.

5. What would be the output of the program for the given number?

    - The output of the program for the given number (**153**) would be: "153 is an Armstrong number."

6. Are there any limitations or restrictions when using this program?

    - This program works correctly for positive integers. However, it does not handle negative numbers, floating-point numbers, or numbers with leading zeros. It is important to ensure that the input is of the correct data type and within the desired range for accurate evaluation.

LAB EXPERIMENT 5

OBJECTIVE

WAP to print Fibonacci series upto n^{th} term.

PRE-EXPERIMENT QUESTIONS

1. Are there any specific requirements for the implementation, such as using recursion or iteration?
2. Should the program provide any error messages or prompts for user input?

BRIEF DISCUSSION AND EXPLANATION

```
def fibonacci(n):
    series = [0, 1]  # Initialize the series with the first two terms

    if n <= 1:
        return series[:n + 1] # Return the series up to n if n is 0 or 1

    while len(series) <= n:
        next_term = series[-1] + series[-2] # Calculate the next term
        series.append(next_term)  # Add the next term to the series
    return series
# Prompt the user to enter the value of n
n = int(input("Enter the value of n: "))

# Print the Fibonacci series up to the nth term
print("Fibonacci Series up to", n, ":")
print(*fibonacci(n))
```

Discussion:

1. The program defines a function `fibonacci` that takes an integer `n` as input and returns the Fibonacci series up to the nth term.
2. The function initializes the series with the first two terms, 0 and 1.
3. If `n` is 0 or 1, the function returns the series up to `n` directly.
4. If `n` is greater than 1, the function uses a while loop to calculate the next terms of the series until it reaches the nth term.
5. The next term is calculated by adding the last two terms in the series.
6. The calculated next term is appended to the series.
7. Finally, the function returns the complete Fibonacci series up to the nth term.
8. In the main part of the program, the user is prompted to enter the value of `n`.
9. The Fibonacci series up to the nth term is then printed using the `print` function and the `*` operator to unpack the series.

OUTPUT

```
Enter the value of n: 6
Fibonacci Series up to 6 :
0 1 1 2 3 5 8
>>>
```

POST-EXPERIMENT QUESTIONS

1. Did you encounter any challenges during the implementation of the program?

2. Did you consider any optimizations or performance improvements in your implementation, especially for large values of "n"?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program prints the Fibonacci series up to the nth term.
2. What is the Fibonacci series?
    - The Fibonacci series is a sequence of numbers in which each number is the sum of the two preceding ones. The series starts with 0 and 1.
3. How is the number of terms in the series determined in this program?
    - The number of terms is stored in the variable **n**. In this program, it is set to 10, but you can change it to any desired value.
4. Can you explain the logic used to print the Fibonacci series?
    - The program initializes the first two terms (**first_term** and **second_term**) as 0 and 1, respectively. Then, it prints the first two terms using conditional statements. After that, it enters a loop that starts from 2 and goes up to **n-1**. In each iteration, it calculates the next term by adding the previous two terms and prints it. The values of **first_term** and **second_term** are updated accordingly to generate the next terms.
5. What would be the output of the program for the given number of terms?
    - The output of the program for **n = 10** would be: "Fibonacci series up to 10 terms: 0 1 1 2 3 5 8 13 21 34"
6. Are there any limitations or restrictions when using this program?
    - This program works correctly for positive integers. However, it assumes a fixed number of terms (**n**) and does not handle negative numbers, floating-point numbers, or invalid inputs. It is important to ensure that the input is of the correct data type and within the desired range for accurate evaluation.

LAB EXPERIMENT 6

OBJECTIVE

WAP to take an integer input from user between 0-100 and print its name spelling in English using list.

PRE-EXPERIMENT QUESTIONS

1. Are there any constraints or validations on the range of the input integer?
2. How should the program handle non-integer inputs or invalid inputs?

BRIEF DISCUSSION AND EXPLANATION

Python program that takes an integer input from the user between 0 and 100 and prints its name spelling in English using two lists:

1. Start by obtaining the input from the user using the input() function. Prompt the user to enter an integer between 0 and 100.

2. Create two lists, one for the names of numbers from 0 to 19 and another for the tens digits (20, 30, 40, etc.) up to 100.

3. Check if the entered number is within the range of 0 to 100.

4. If the number is less than 20, use the first list to get the corresponding name.

5. If the number is 20 or greater, use the second list to get the tens digit name and append the name from the first list for the units digit.

6. Print the English name of the entered number.

Here's an example of how the complete program would look:

```
# Obtain the input from the user
num = int(input("Enter a number between 0 and 100: "))

# Define the lists of English names
units_names = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten",
               "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen",
               "eighteen", "nineteen"]

tens_names = ["", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"]

# Check if the number is within the valid range and print its English name
if 0 <= num <= 100:
```

```
    if num < 20:
        print(units_names[num])
    else:
        tens_digit = num // 10
        units_digit = num % 10
        if units_digit == 0:
            print(tens_names[tens_digit])
        else:
            print(tens_names[tens_digit] + "-" + units_names[units_digit])
else:
    print("Invalid input. Please enter a number between 0 and 100.")
```

In this program, the user is prompted to enter a number between 0 and 100. The program checks if the number is within the valid range. If the number is less than 20, it directly retrieves the corresponding name from the `units_names` list. If the number is 20 or greater, it calculates the tens and units digits, and then combines the names from the `tens_names` and `units_names` lists to form the complete English name. Finally, it prints the English name of the entered number.

OUTPUT

```
Enter a number between 0 and 100: 19
nineteen
>>>
```

POST-EXPERIMENT QUESTIONS

1. Did you consider any optimizations or improvements in your implementation?

2. Did you test the program with different input values within the range of 0-100? If so, did it produce the correct name spelling in English?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program takes an integer input from the user between 0 and 100 and prints its name spelling in English using a list.
2. How is the name spelling of a number stored in the program?
    - The name spelling of each number from 0 to 100 is stored in lists called **unit_names and ten_names**.
3. How does the program validate the user's input?
    - After taking the input from the user, the program checks if the number is less than 0 or greater than 100. If it is, it displays an "Invalid input!" message.

4. What would be the output of the program for a valid input?
    - If the user enters a valid input, such as 42, the output would be: "Name spelling of 42 is: forty-two"

5. Are there any limitations or restrictions when using this program?
    - This program assumes that the user will enter a valid integer between 0 and 100. It does not handle non-integer inputs or numbers outside the given range. Additional error handling can be added to make the program more robust.

LAB EXPERIMENT 7

OBJECTIVE

WAP to convert hexadecimal number to binary using dictionary.

PRE-EXPERIMENT QUESTIONS

1. What are hexadecimal numbers?
2. How do we use a dictionary in python?

BRIEF DISCUSSION AND EXPLANATION

Python program that converts a hexadecimal number to binary using a dictionary:

1. Start by obtaining the input from the user, either by hardcoding a hexadecimal number or using input() to prompt the user for a hexadecimal number.

2. Create a dictionary that maps each hexadecimal digit to its corresponding binary representation. For example, {'0': '0000', '1': '0001', '2': '0010', ..., 'F': '1111'}.

3. Initialize an empty string to store the binary representation.

4. Iterate through each digit in the hexadecimal number.

5. Use the dictionary to retrieve the binary representation of each digit and append it to the binary string.

6. Print the binary representation.

Here's an example of how the complete program would look:

```
# Obtain the input from the user
hex_num = input("Enter a hexadecimal number: ")

# Define the dictionary mapping hexadecimal digits to binary representation
hex_to_bin = {
    '0': '0000',
    '1': '0001',
    '2': '0010',
    '3': '0011',
    '4': '0100',
    '5': '0101',
    '6': '0110',
    '7': '0111',
    '8': '1000',
    '9': '1001',
```

```
    'A': '1010',
    'B': '1011',
    'C': '1100',
    'D': '1101',
    'E': '1110',
    'F': '1111'
}

# Initialize the binary representation
binary = ""

# Convert the hexadecimal number to binary
for digit in hex_num:
    binary += hex_to_bin[digit]

# Print the binary representation
print("Binary:", binary)
```

In this program, the user is prompted to enter a hexadecimal number. The program uses a dictionary `hex_to_bin` to map each hexadecimal digit to its corresponding binary representation. It iterates through each digit in the hexadecimal number and retrieves its binary representation from the dictionary. The binary representations are then appended together to form the complete binary representation. Finally, the program prints the binary representation.
By running this program and entering a hexadecimal number, you can convert it to binary using the dictionary mapping.

OUTPUT

```
Enter a hexadecimal number: A2
Binary: 10100010
>>>
```

POST-EXPERIMENT QUESTIONS

1. How did you use dictionary for conversion?
2. Did you test program for multi digit hexadecimal number?

QUIZ WITH ANSWERS

1. What does this program do?

    - This program converts a hexadecimal number to binary using a dictionary.

2. How is the hexadecimal-to-binary conversion implemented in the program?

    - The program uses a dictionary called **hex_to_bin** to map each hexadecimal digit to its binary equivalent. It then iterates through each digit of the input hexadecimal number, looks up its binary representation in the dictionary, and appends it to a string variable called **binary_num**.

3. How is the input obtained from the user?

    - The input hexadecimal number is obtained from the user using the **input()** function.

4. What is the purpose of the **if** statement in the program?

    - The **if** statement checks if each digit of the hexadecimal number exists in the **hex_to_bin** dictionary. If it does, the corresponding binary representation is appended to **binary_num**. If a digit is not found in the dictionary, it prints an error message and breaks out of the loop.

5. Can you explain the output of the program for a valid input?

    - If the user enters a valid hexadecimal number, such as "1A4", the output would be: "Binary: 000110100100"

6. What happens if the user enters an invalid hexadecimal digit?

    - If the user enters an invalid hexadecimal digit, such as "G" or "z", the program prints an error message stating that it is an invalid digit.

7. Are there any limitations or restrictions when using this program?

    - This program assumes that the user will enter a valid hexadecimal number. It does not handle non-hexadecimal characters or provide error correction. Additional error handling can be added to make the program more robust.

LAB EXPERIMENT 8

OBJECTIVE

WAP to find sum and average of tuple elements.

PRE-EXPERIMENT QUESTIONS

1. What is tuple in python?
2. How to work with for loop in python?

BRIEF DISCUSSION AND EXPLANATION

Python program that finds the sum and average of tuple elements:

1. Start by defining a tuple containing the elements for which you want to calculate the sum and average.

2. Initialize variables for the sum and count. Set the sum variable to 0 and the count variable to the length of the tuple.

3. Iterate through each element in the tuple using a for loop.

4. Add each element to the sum variable.

5. Calculate the average by dividing the sum by the count.

6. Print the sum and average.

Here's an example of how the complete program would look:

```
# Define the tuple
numbers = (10, 20, 30, 40, 50)

# Calculate the sum
sum = 0
for num in numbers:
    sum += num

# Calculate the average
count = len(numbers)
average = sum / count

# Print the sum and average
print("Sum:", sum)
print("Average:", average)
```

In this program, the tuple `numbers` contains the elements for which we want to calculate the sum and average. The program iterates through each element in the tuple, adding them to the sum variable. Then, it calculates the average by dividing the sum by the count, which is the length of the tuple. Finally, it prints the sum and average.

By running this program with different tuples, you can find the sum and average of their elements.

OUTPUT

```
Sum: 150
Average: 30.0
>>>
```

POST-EXPERIMENT QUESTIONS

1. Are tuple elements pre-defined or user input?
2. How should the program handle non-integer inputs or invalid inputs?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program finds the sum and average of elements in a tuple.
2. How are the tuple elements stored in the program?
    - The tuple elements are stored in the variable **numbers**.
3. Can you explain how the sum of the tuple elements is calculated?
    - The **sum()** function is used to calculate the sum of the elements in the **numbers** tuple. It takes the tuple as an argument and returns the sum of all the elements.
4. How is the average of the tuple elements calculated?
    - The average is calculated by dividing the sum of the tuple elements by the number of elements in the tuple. The length of the tuple is obtained using the **len()** function.
5. What would be the output of the program for the given tuple?
    - For the given tuple **(10, 20, 30, 40, 50)**, the output would be:

      Sum of elements: 150 Average of elements: 30.0
6. Are there any limitations or restrictions when using this program?
    - This program assumes that the elements in the tuple are numeric. It does not handle non-numeric elements or check for empty tuples. Additional error handling can be added to make the program more robust.

LAB EXPERIMENT 9

OBJECTIVE

WAP to take an integer input from user. Calculate and display its factorial using recursion.

PRE-EXPERIMENT QUESTIONS

1. How to define a function in python?
2. What is the use of a recursive function?

BRIEF DISCUSSION AND EXPLANATION

Python program that takes an integer input from the user, calculates its factorial using recursion, and displays the result:

1. Start by obtaining the input from the user using the input() function. Prompt the user to enter an integer for which you want to calculate the factorial.

2. Define a recursive function called `factorial` that takes an integer parameter. This function will be responsible for calculating the factorial.

3. In the `factorial` function, add a base case to handle the termination condition. If the input number is 0 or 1, return 1, as the factorial of 0 and 1 is 1.

4. If the input number is greater than 1, call the `factorial` function recursively with the argument as `n-1` and multiply the result with `n`.

5. In the main part of the program, convert the user input to an integer and call the `factorial` function with the input number as the argument.

6. Print the factorial result.

Here's an example of how the complete program would look:

```
# Define the recursive function to calculate factorial
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Obtain the input from the user
num = int(input("Enter an integer: "))
```

```
# Calculate the factorial using recursion
result = factorial(num)

# Print the factorial
print("Factorial of", num, "is", result)
```

In this program, the user is prompted to enter an integer for which the factorial will be calculated. The `factorial` function is defined to calculate the factorial using recursion. It checks for the base case where the input number is 0 or 1 and returns 1. For numbers greater than 1, it calls itself recursively with `n-1` as the argument and multiplies the result with `n`. Finally, the factorial result is printed.

By running this program and entering different integers, you can calculate and display their factorial using recursion.

OUTPUT

```
Enter an integer: 5
Factorial of 5 is 120
>>>
```

POST-EXPERIMENT QUESTIONS

1. How did you implement the program to calculate the factorial using recursion?
2. Did you consider any optimizations or improvements in your recursive implementation?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program takes an integer input from the user and calculates its factorial using recursion.
2. What is factorial?
    - Factorial of a non-negative integer **n** is the product of all positive integers from 1 to **n**. It is denoted by **n!**. For example, factorial of 5 (**5!**) is equal to 5 x 4 x 3 x 2 x 1 = 120.
3. How is recursion used to calculate the factorial in this program?
    - The **factorial()** function is defined to calculate the factorial of a number. It uses recursion by calling itself with a smaller value (**n - 1**) until it reaches the base case, which is when **n** becomes 0. In the base case, it returns 1. For other values of **n**, it returns **n** multiplied by the factorial of **n - 1**.
4. Can you explain the output of the program for a valid input?
    - If the user enters a valid positive integer, such as 5, the output would be: "Factorial of 5 is: 120"

5. What happens if the user enters a negative number?
    - If the user enters a negative number, the program displays an error message stating that the factorial is not defined for negative numbers.
6. Are there any limitations or restrictions when using this program?
    - This program assumes that the user will enter a valid integer. It does not handle non-integer inputs or numbers outside the valid range. Additional error handling can be added to make the program more robust.

LAB EXPERIMENT 10

OBJECTIVE

WAP to implement a python class with a method to print area of a triangle using heron's formula using inheritance.

PRE-EXPERIMENT QUESTIONS

1. What do you understand by OOPs concept?
2. What are different OOPs properties?

BRIEF DISCUSSION AND EXPLANATION

Python program that implements a class with a method to print the area of a triangle using Heron's formula using inheritance:

1. Start by creating a parent class called `Shape` that will serve as the base class for other shapes. In this case, we will focus on triangles.

2. Inside the `Shape` class, define an `__init__` method to initialize the necessary attributes of a triangle, such as its three sides.

3. Implement a method called `calculate_area` within the `Shape` class. This method will use Heron's formula to calculate the area of the triangle. Heron's formula states that the area of a triangle with sides `a`, `b`, and `c` is given by the formula: `sqrt(s * (s - a) * (s - b) * (s - c))`, where `s` is the semi-perimeter of the triangle.

4. Create a child class called `Triangle` that inherits from the `Shape` class.

5. In the `Triangle` class, override the `calculate_area` method to implement the specific logic for calculating the area of a triangle.

6. Create an instance of the `Triangle` class and provide the necessary side lengths as arguments to the constructor.

7. Call the `calculate_area` method on the instance to print the area of the triangle.

Here's an example of how the complete program would look:

```
import math

class Shape:
    def__init_(self, side1, side2, side3):
        self.side1 = side1
        self.side2 = side2
```

```
        self.side3 = side3

    def calculate_area(self):
        raise NotImplementedError("Subclass must implement calculate_area method")

class Triangle(Shape):
    def calculate_area(self):
        # Calculate the semi-perimeter
        s = (self.side1 + self.side2 + self.side3) / 2

        # Calculate the area using Heron's formula
        area = math.sqrt(s * (s - self.side1) * (s - self.side2) * (s - self.side3))

        return area

# Create an instance of the Triangle class
triangle = Triangle(5, 12, 13)

# Calculate and print the area of the triangle
area = triangle.calculate_area()
print("Area of the triangle:", area)
```

In this program, we define a parent class `Shape` that contains an `__init__` method to initialize the sides ofa triangle and a `calculate_area` method. The `calculate_area` method is marked as `NotImplementedError` since it will be overridden in the child class.

We then create a child class `Triangle` that inherits from the `Shape` class. Inside the `Triangle` class, we implement the `calculate_area` method to calculate the area of a triangle using Heron's formula.

Finally, we create an instance of the `Triangle` class, passing the side lengths as arguments. We call the `calculate_area` method on the instance to calculate the area of the triangle, which is then printed.

By running this program, you can implement a class with a method to print the area of a triangle using Heron's formula using inheritance.

OUTPUT

```
Area of the triangle: 30.0
>>>
```

POST-EXPERIMENT QUESTIONS

1. How did you implement the Python class with the method to calculate and print the area of a triangle using Heron's formula?

2. How did you handle cases where the triangle's side lengths were not valid or where they formed an invalid triangle?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program implements a Python class with a method to calculate and print the area of a triangle using Heron's formula. It also demonstrates inheritance by creating a subclass that inherits the calculation method and adds a printing method.
2. What is Heron's formula?
    - Heron's formula is used to calculate the area of a triangle given the lengths of its three sides. It states that the area is equal to the square root of the semi-perimeter multiplied by the difference between the semi-perimeter and each side length.
3. How is the Triangle class defined?
    - The Triangle class is defined with an **__init__**method to initialize the side lengths, and a **calculate_area** method to calculate the area using Heron's formula.
4. What is the purpose of the PrintableTriangle class?
    - The PrintableTriangle class is a subclass of Triangle. It inherits the calculation method from the Triangle class and adds a **print_area** method to print the calculated area.
5. How does the **print_area** method calculate and print the area?
    - The **print_area** method calls the **calculate_area** method to calculate the area of the triangle. It then prints the calculated area using the **print()** function.
6. What would be the output of the program for the given side lengths?
    - For the given side lengths (5, 6, 7), the output would be: "Area of the Triangle: 14.696938456699069"
7. Can you explain the concept of inheritance demonstrated in this program?
    - Inheritance is demonstrated by creating the PrintableTriangle class as a subclass of Triangle. PrintableTriangle inherits the **calculate_area** method from the Triangle class, which means it can use the inherited method to perform the calculation without redefining it.

LAB EXPERIMENT 11

OBJECTIVE

WAP to create a GUI for calculator.

PRE-EXPERIMENT QUESTIONS

1. What are the necessary modules to create a GUI?
2. What is the use of Tkinter module in python?

BRIEF DISCUSSION AND EXPLANATION

Creating a GUI for a calculator involves using a GUI library such as Tkinter in Python. Here's a brief discussion on how you can create a GUI for a calculator:

1. Import the necessary modules, including the Tkinter module for GUI functionality.

2. Create an instance of the Tkinter class to represent the main window of the calculator.

3. Design the layout of the calculator by placing buttons and display widgets appropriately. You can use Tkinter's grid or pack layout managers to arrange the elements.

4. Define functions for the calculator operations, such as addition, subtraction, multiplication, and division.

5. Connect the functions to the buttons so that when a button is clicked, the corresponding operation is performed.

6. Implement a display widget to show the numbers and results.

7. Run the Tkinter event loop, which waits for user input and handles events such as button clicks.

Here's an example of how the complete program might look:

```
import tkinter as tk

def calculate():
    """Performs the calculation and updates the display"""
    expression = display.get()
    try:
        result = eval(expression)
        display.delete(0, tk.END)
        display.insert(tk.END, str(result))
```

```
    except:
        display.delete(0, tk.END)
        display.insert(tk.END, "Error")

def append_to_display(value):
    """Appends the clicked value to the display"""
    display.insert(tk.END, value)

def clear_display():
    """Clears the display"""
    display.delete(0, tk.END)

# Create the main window
window = tk.Tk()
window.title("Calculator")

# Create the display widget
display = tk.Entry(window, width=30, font=("Arial", 16), justify="right")
display.grid(row=0, column=0, columnspan=4, padx=10, pady=10)

# Create number buttons
for i in range(1, 10):
    button = tk.Button(window, text=str(i), width=7, height=3, font=("Arial", 14),
                       command=lambda i=i: append_to_display(str(i)))
    button.grid(row=(9-i)//3 + 1, column=(i-1) % 3, padx=5, pady=5)

# Create operation buttons
operations = ['+', '-', '*', '/']
row = 1
col = 3
for operation in operations:
    button = tk.Button(window, text=operation, width=7, height=3, font=("Arial", 14),
                       command=lambda operation=operation: append_to_display(operation))
    button.grid(row=row, column=col, padx=5, pady=5)
    row += 1

# Create other buttons
button_clear = tk.Button(window, text="C", width=7, height=3, font=("Arial", 14), command=clear_display)
button_clear.grid(row=4, column=0, padx=5, pady=5)

button_zero = tk.Button(window, text="0", width=7, height=3, font=("Arial", 14),
                        command=lambda: append_to_display("0"))
button_zero.grid(row=4, column=1, padx=5, pady=5)

button_equal = tk.Button(window, text="=", width=7, height=3, font=("Arial", 14), command=calculate)
button_equal.grid(row=4, column=2, padx=5, pady=5)
```

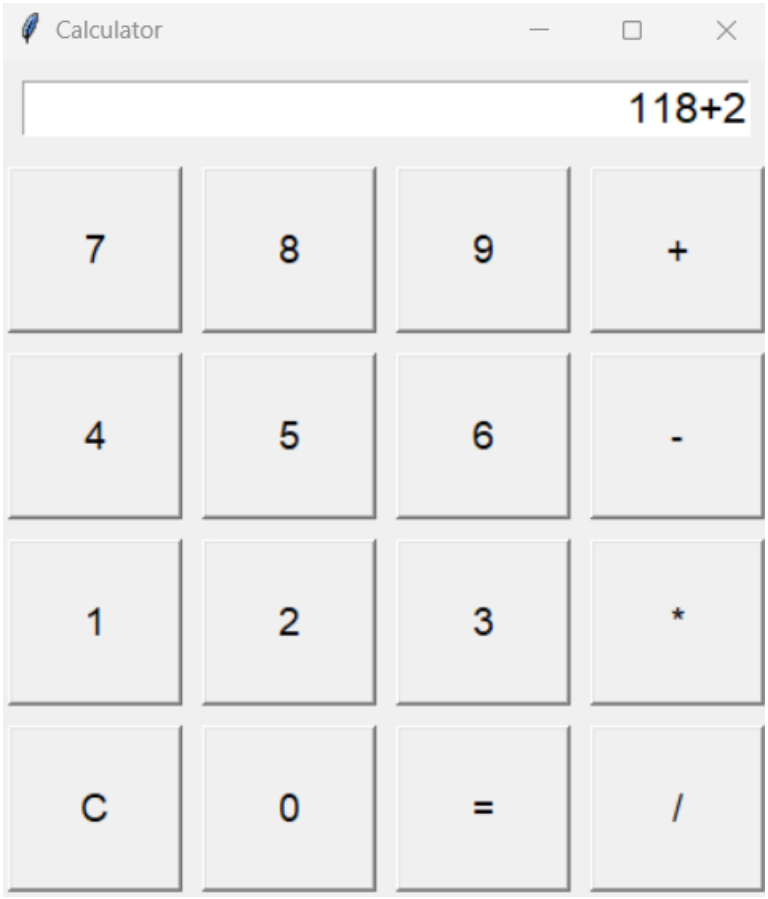
```
# Run the main event loop
window.mainloop()
```

In this program, we create a simple calculator GUI using Tkinter. The calculator consists of number buttons, operation buttons, a clear button, a zero button, and an equal button. The display is implemented using a Tkinter

entry widget. Each button is connected to a corresponding function that performs the desired operation. The `calculate()` function evaluates the expression and updates the display accordingly. The `append_to_display()` function appends the clicked value to the display, and the `clear_display()` function clears the display.

By running this program, you can create a GUI calculator that allows users to perform calculations interactively.

OUTPUT



POST-EXPERIMENT QUESTIONS

1. How did you implement the GUI for the calculator? Which library or framework did you use?

2. What functionalities did you include in the calculator GUI? Did it support basic arithmetic operations?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program creates a GUI for a simple calculator using the Tkinter framework in Python.
2. What is Tkinter?
    - Tkinter is a standard GUI framework for Python. It provides classes and functions to create and manipulate GUI components like windows, buttons, labels, and entry fields.
3. How does the program handle button clicks?
    - The program defines a function button_click that is called when a button is clicked. It retrieves the text of the clicked button, evaluates the expression if it is the "=" button, clears the entry field if it is the "C" button, or appends the clicked button's text to the entry field.
4. How is the GUI layout structured?
    - The GUI layout is organized using the grid layout manager. The entry field occupies the top row, and the buttons are arranged in a grid below it.
5. What happens when the "=" button is clicked?
    - When the "=" button is clicked, the program attempts to evaluate the expression entered in the entry field using the eval function. If the evaluation is successful, the result is displayed in the entry field. Otherwise, an error message is displayed.
6. Can you explain the purpose of the bind method in button creation?
    - The bind method is used to bind the click event ("<Button-1>") to the button_click function. This ensures that the function is called whenever the button is clicked.

LAB EXPERIMENT 12

OBJECTIVE

WAP to implement a person class (Name, address, phone, Id) and derive two classes class teacher(additional TID, Course, faculty) and student Roll_no, course enrolled) from person class to show the use of Inheritance.

PRE-EXPERIMENT QUESTIONS

1. What do you understand by OOPs concept?
2. What are different OOPs properties?

BRIEF DISCUSSION AND EXPLANATION

Python program that implements a `Person` class with attributes for name, address, phone, and ID, and then derives two classes `Teacher` and `Student` from the `Person` class to demonstrate inheritance:

1. Start by creating the `Person` class with the attributes `name`, `address`, `phone`, and `ID`. Provide an `__init__` method to initialize these attributes.

2. Create the `Teacher` class as a subclass of `Person`. Inherit the attributes and methods of the `Person` class using the `super()` function.

3. Add additional attributes specific to the `Teacher` class, such as `TID` (Teacher ID), `course`, and `faculty`. Define an `__init__` method in the `Teacher` class to initialize these attributes and call the `super()` method to initialize the attributes inherited from the `Person` class.

4. Create the `Student` class as another subclass of `Person`. Inherit the attributes and methods of the `Person` class using the `super()` function.

5. Add additional attributes specific to the `Student` class, such as `roll_no` and `course_enrolled`. Define an `_init_` method in the `Student` class to initialize these attributes and call the `super()` method to initialize the attributes inherited from the `Person` class.

Here's an example of how the complete program would look:

```
class Person:
   def__init__(self, name, address, phone, ID):
      self.name = name
      self.address = address
      self.phone = phone
      self.ID = ID
```

```
class Teacher(Person):
    def__init_(self, name, address, phone, ID, TID, course, faculty):
        super()._init_(name, address, phone, ID)
        self.TID = TID
        self.course = course
        self.faculty = faculty

class Student(Person):
    def__init_(self, name, address, phone, ID, roll_no, course_enrolled):
        super()._init_(name, address, phone, ID)
        self.roll_no = roll_no
        self.course_enrolled = course_enrolled

# Create instances of Teacher and Student
teacher = Teacher("John Doe", "123 Main St", "1234567890", "P001", "T001", "Math", "Science")
student = Student("Jane Smith", "456 Elm St", "9876543210", "S001", "R001", "Computer Science")

# Access attributes of Teacher
print("Teacher Name:", teacher.name)
print("Teacher Address:", teacher.address)
print("Teacher Phone:", teacher.phone)
print("Teacher ID:", teacher.ID)
print("Teacher TID:", teacher.TID)
print("Teacher Course:", teacher.course)
print("Teacher Faculty:", teacher.faculty)

# Access attributes of Student
print("Student Name:", student.name)
print("Student Address:", student.address)
print("Student Phone:", student.phone)
print("Student ID:", student.ID)
print("Student Roll No:", student.roll_no)
print("Student Course Enrolled:", student.course_enrolled)
```

In this program, we define a `Person` class that represents a person with attributes for name, address, phone, and ID. The `Teacher` class is created as a subclass of `Person` and adds additional attributes for TID, course, and faculty. The `Student` class is also a subclass of `Person` and includes additional attributes for roll_no and course_enrolled.

We create instances of the `Teacher` and `Student` classes and access their attributes to demonstrate how the inherited attributes and subclass-specific attributes work.

By running this program, you can implement a person class and derive two classes, `Teacher` and `Student`, from the `Person` class to showcase the use of inheritance.

POST-EXPERIMENT QUESTIONS

1. How did you implement the person class with the attributes Name, address, phone, and Id?

2. How did you use inheritance to derive the teacher and student classes from the person class?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program demonstrates the use of inheritance by implementing a Person class and deriving two classes, Teacher and Student, from it.
2. What is inheritance?
    - Inheritance is a mechanism in object-oriented programming where a class inherits the properties and methods of another class. The derived class (subclass) can extend or modify the behavior of the base class (superclass).
3. How is inheritance used in this program?
    - The Teacher and Student classes are derived from the Person class using inheritance. They inherit the attributes (name, address, phone, ID) and methods of the Person class, allowing them to reuse and extend its functionality.
4. What is the purpose of the **super().__init__()** call in the derived classes' **__init__**methods?
    - The **super()._init_()** call is used to invoke the constructor of the base class (Person) in the derived classes (Teacher and Student). It ensures that the attributes inherited from the base class are initialized correctly.
5. How are objects created for Teacher and Student classes?
    - Objects for the Teacher and Student classes are created by calling their respective constructors and providing the required parameters for the base class (Person) as well as the additional attributes specific to each class.
6. What is the output of the program?
    - The program outputs the details of a Teacher object and a Student object, including their name, address, phone, ID, and additional attributes specific to each class.

LAB EXPERIMENT 13

OBJECTIVE

WAP to plot histogram of the following data.

10-15	15-20	20-25	25-30	30-35
5	6	9	8	2

PRE-EXPERIMENT QUESTIONS

1. What is matplotlib module?
2. What function is used to plot histogram?

BRIEF DISCUSSION AND EXPLANATION

To plot a histogram of the given data, you can use a Python library such as Matplotlib. Here's a brief discussion on how you can create a program to plot the histogram:

1. Import the necessary modules, including Matplotlib's pyplot module.

2. Create two lists: one for the data range labels (`labels`) and another for the corresponding frequencies (`data`).

3. Use the `bar` function from Matplotlib to create the histogram. Provide the `labels` and `data` lists as the x and y values respectively.

4. Customize the histogram by adding labels to the x-axis, y-axis, and title.

5. Show the plot using the `show` function.

Here's an example of how the complete program might look:

```
import matplotlib.pyplot as plt

# Define the data
labels = ['10-15', '15-20', '20-25', '25-30', '30-35']
data = [5, 6, 9, 8, 2]

# Create the histogram
plt.bar(labels, data)

# Add labels and title
plt.xlabel('Range')
plt.ylabel('Frequency')
```

```
plt.title('Histogram of Data')

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

In this program, we define the `labels` list to represent the data ranges and the `data` list to represent the corresponding frequencies.

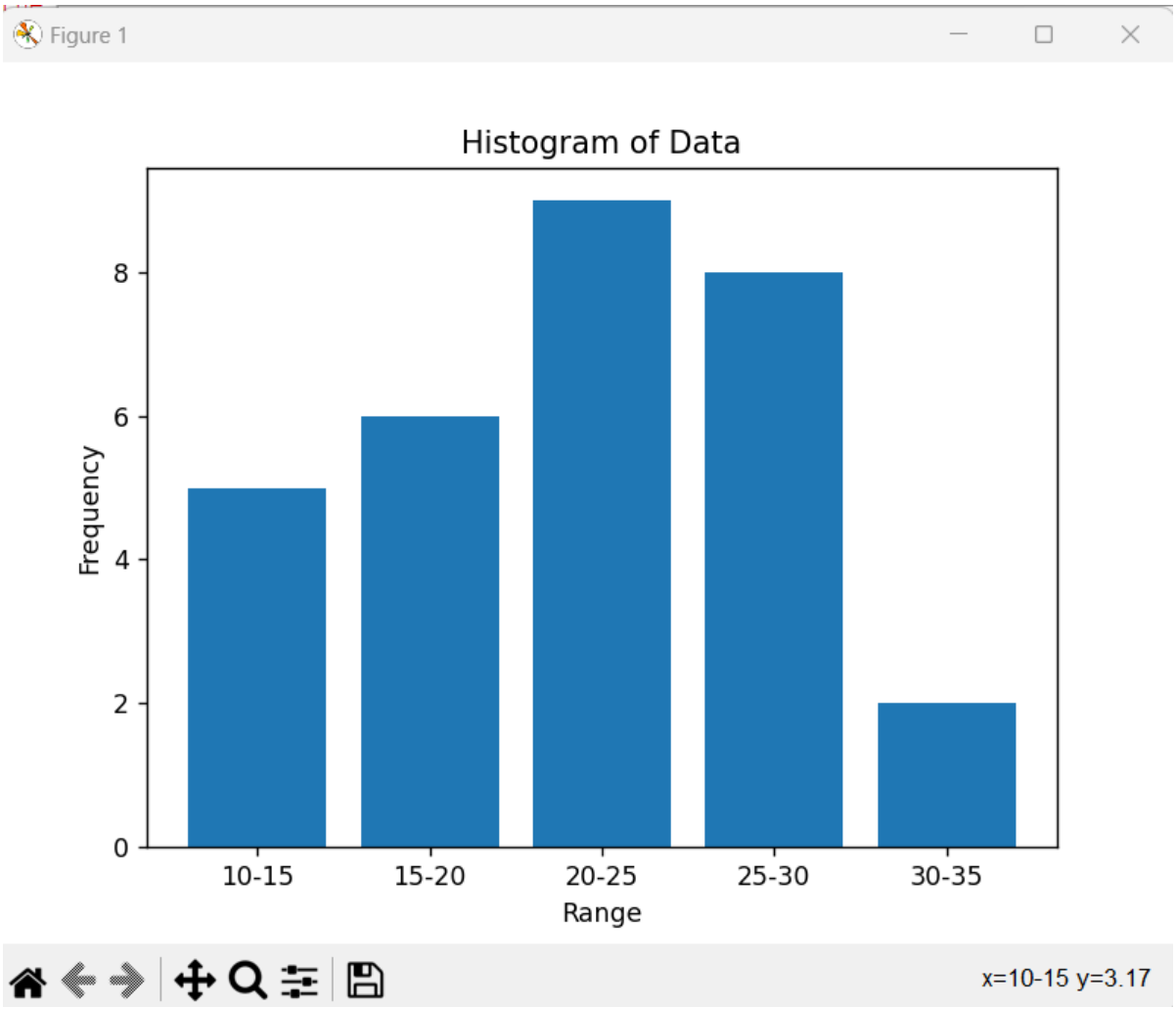
We use the `bar` function from Matplotlib to create the histogram. The `labels` and `data` lists are provided as the x and y values, respectively.

We then customize the histogram by adding labels to the x-axis (`xlabel`), y-axis (`ylabel`), and title (`title`).

Finally, we use the `show` function to display the plot.

By running this program, you can plot a histogram of the given data using Matplotlib.

OUTPUT



POST-EXPERIMENT QUESTIONS

1. How did you implement the program to plot a histogram of the given data?

2. Which library or tool did you use to plot the histogram?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program plots a histogram of the given data using the matplotlib library in Python.
2. What is a histogram?
    - A histogram is a graphical representation of the distribution of numerical data. It consists of a set of bars, where the width of each bar represents a range of values and the height represents the frequency or count of data points within that range.
3. How is the data stored in the program?
    - The data is stored in two lists: which contains the range labels, and the corresponding counts or frequencies.
4. How is the histogram plotted using matplotlib?
    - The **plt.bar()** function from the matplotlib library is used to create a bar plot. It takes the data range as the x-axis values and the data counts as the corresponding y-axis values.
5. What labels and title are added to the plot?
    - The **plt.xlabel()**, **plt.ylabel()**, and **plt.title()** functions are used to add labels to the x-axis, y-axis, and title of the plot, respectively.
6. What would be the output of the program for the given data?
    - The program would display a histogram with bars representing the data ranges (10-15, 15-20, 20-25, 25-30, 30-35) on the x-axis and the corresponding counts (5, 6, 9, 8, 2) on the y-axis.

LAB EXPERIMENT 14

OBJECTIVE

WAP to draw a box-and-whisker plot for the data set {3, 7, 8, 5, 12, 14, 21, 13, 18}.

PRE-EXPERIMENT QUESTIONS

1. What is matplotlib module?
2. What function is used to draw box-and-whisker plot?

BRIEF DISCUSSION AND EXPLANATION

To draw a box-and-whisker plot for the given data set {3, 7, 8, 5, 12, 14, 21, 13, 18}, you can use a Python library such as Matplotlib. Here's a brief discussion on how you can create a program to draw the box-and-whisker plot:

1. Import the necessary modules, including Matplotlib's pyplot module.

2. Create a list to represent the data set.

3. Use the `boxplot` function from Matplotlib to create the box-and-whisker plot. Provide the data set as the input.

4. Customize the plot if desired by adding labels to the x-axis, y-axis, and title.

5. Show the plot using the `show` function.

Here's an example of how the complete program might look:

```
import matplotlib.pyplot as plt

# Define the data set
data = [3, 7, 8, 5, 12, 14, 21, 13, 18]

# Create the box-and-whisker plot
plt.boxplot(data)

# Add labels and title
plt.xlabel('Data Set')
plt.ylabel('Values')
plt.title('Box-and-Whisker Plot')

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

In this program, we define the `data` list to represent the data set.

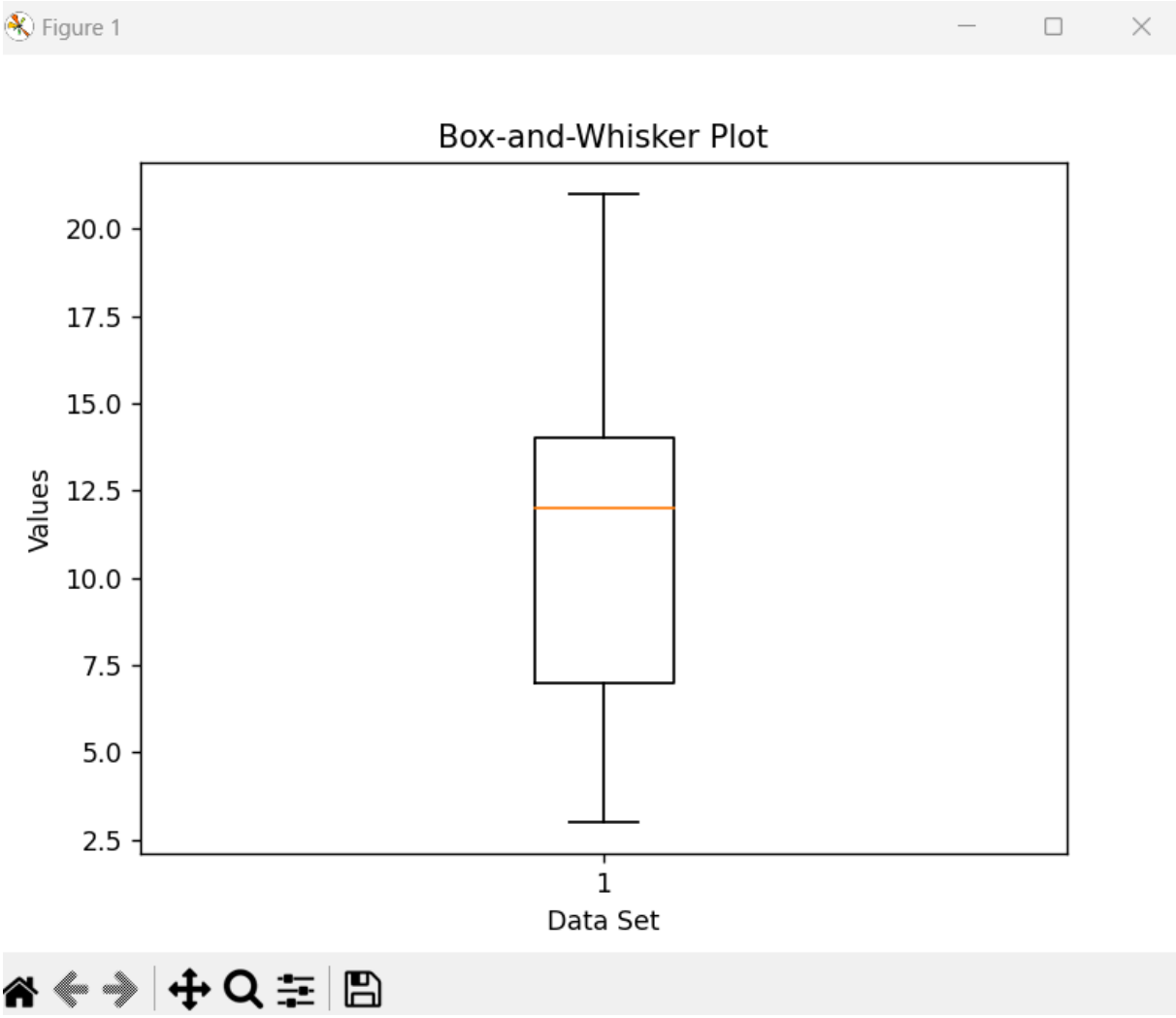
We use the `boxplot` function from Matplotlib to create the box-and-whisker plot. The `data` list is provided as the input.

We then customize the plot by adding labels to the x-axis (`xlabel`), y-axis (`ylabel`), and title (`title`).

Finally, we use the `show` function to display the plot.

By running this program, you can draw a box-and-whisker plot for the given data set using Matplotlib.

OUTPUT



POST-EXPERIMENT QUESTIONS

1. How did you implement the program to draw a box-and-whisker plot of the given data?

2. Which library or tool did you use to plot the box-and-whisker plot?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program draws a box-and-whisker plot for the given data set using the matplotlib library in Python.
2. What is a box-and-whisker plot?
    - A box-and-whisker plot is a graphical representation of the distribution of numerical data. It displays a summary of the data's minimum, maximum, median, and quartiles in a visually concise way.
3. How is the data stored in the program?
    - The data is stored in a list called **data**.
4. How is the box-and-whisker plot created using matplotlib?
    - The **plt.boxplot()** function from the matplotlib library is used to create a box-and-whisker plot. It takes the data as an argument and automatically calculates and displays the minimum, maximum, median, and quartiles.
5. What title is added to the plot?
    - The **plt.title()** function is used to add a title to the plot. In this case, the title is "Box-and-Whisker Plot".
6. What would be the output of the program for the given data set?
    - The program would display a box-and-whisker plot showing the minimum, maximum, median, and quartiles of the given data set: {3, 7, 8, 5, 12, 14, 21, 13, 18}.

LAB EXPERIMENT 15

OBJECTIVE

WAP to draw the Line Plot and Bar chart for the following data.

Elapsed time (s)	0	1	2	3	4	5	6
Speed(m/s)	0	3	7	12	20	30	45.6

PRE-EXPERIMENT QUESTIONS

1. What is matplotlib module?
2. What function is used to draw line plot and bar chart?

BRIEF DISCUSSION AND EXPLANATION

To draw a line plot and a bar chart for the given data, Python library such as Matplotlib can be used. Here's a brief discussion on how one can create a program to plot both the line plot and the bar chart:

1. Import the necessary modules, including Matplotlib's pyplot module.

2. Create two lists: one for the elapsed time values (`time`) and another for the corresponding speed values (`speed`).

3. Use the `plot` function from Matplotlib to create the line plot. Provide the `time` and `speed` lists as the x and y values, respectively.

4. Use the `bar` function from Matplotlib to create the bar chart. Provide the `time` and `speed` lists as the x and y values, respectively.

5. Customize the plots by adding labels to the x-axis, y-axis, and title.

6. Show the plots using the `show` function.

Here's an example of how the complete program might look:

```
import matplotlib.pyplot as plt

# Define the data
time = [0, 1, 2, 3, 4, 5, 6]
speed = [0, 3, 7, 12, 20, 30, 45.6]
```

```
# Create the line plot
plt.plot(time, speed)

# Add labels and title to the line plot
plt.xlabel('Elapsed time (s)')
plt.ylabel('Speed (m/s)')
plt.title('Line Plot')

# Show the line plot
plt.show()

# Create the bar chart
plt.bar(time, speed)

# Add labels and title to the bar chart
plt.xlabel('Elapsed time (s)')
plt.ylabel('Speed (m/s)')
plt.title('Bar Chart')

# Show the bar chart
plt.show()
```

In this program, we define the `time` list to represent the elapsed time values and the `speed` list to represent the corresponding speed values.

We use the `plot` function from Matplotlib to create the line plot. The `time` and `speed` lists are provided as the x and y values, respectively.

We customize the line plot by adding labels to the x-axis (`xlabel`), y-axis (`ylabel`), and title (`title`).

We then use the `bar` function from Matplotlib to create the bar chart. Again, the `time` and `speed` lists are provided as the x and y values, respectively.

We customize the bar chart by adding labels to the x-axis, y-axis, and title.

Finally, we use the `show` function to display both the line plot and the bar chart.

By running this program, you can draw a line plot and a bar chart for the given data using Matplotlib.

OUTPUT

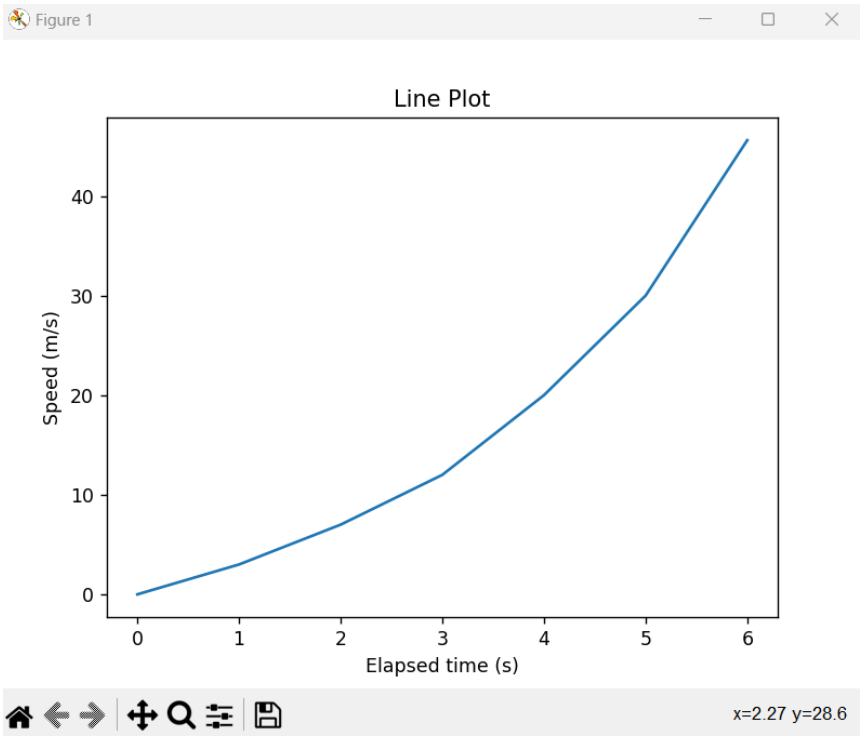


Figure 1
Line Plot
Speed (m/s)
Elapsed time (s)
0
10
20
30
40
0
1
2
3
4
5
6
x=2.27 y=28.6

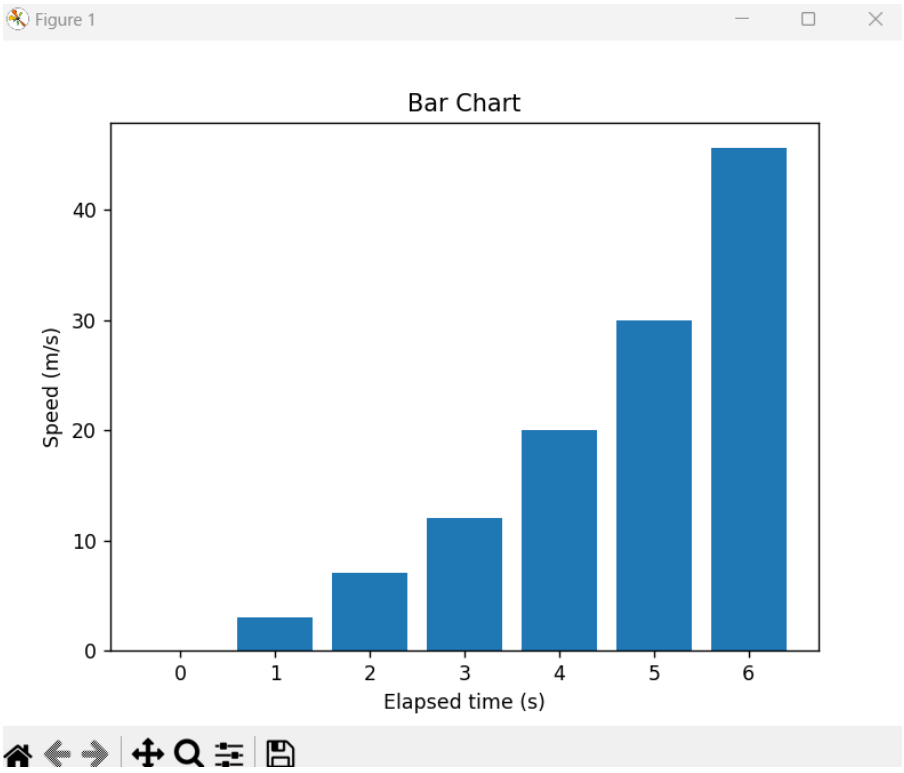


Figure 1
Bar Chart
Speed (m/s)
Elapsed time (s)
0
10
20
30
40
0
1
2
3
4
5
6

POST-EXPERIMENT QUESTIONS

1. How did you implement the program to draw line plot and bar chart of the given data?

2. Which library or tool did you use to draw line plot and bar chart?

QUIZ WITH ANSWERS

1. What does this program do?
    - This program draws a line plot and a bar chart for the given data using the matplotlib library in Python.
2. What is a line plot?
    - A line plot is a graphical representation of data points connected by straight line segments. It shows the trend or progression of a variable over time or other ordered categories.
3. How is the line plot created using matplotlib?
    - The **plt.plot()** function from the matplotlib library is used to create a line plot. It takes the x-axis values (years) and the corresponding y-axis values (sales) as arguments. Additional parameters like marker style, linestyle, and color can be specified to customize the plot.
4. What is a bar chart?
    - A bar chart is a graphical representation of categorical data using rectangular bars of varying heights. It shows the distribution or comparison of different categories.
5. How is the bar chart created using matplotlib?
    - The **plt.bar()** function from the matplotlib library is used to create a bar chart. It takes the x-axis values (years) and the corresponding y-axis values (sales) as arguments. Additional parameters like color can be specified to customize the chart.
6. What labels and titles are added to the plots?
    - The **plt.xlabel()**, **plt.ylabel()**, and **plt.title()** functions are used to add labels to the x-axis, y-axis, and title of each plot, respectively.
7. What would be the output of the program for the given data?
    - The program would display a line plot showing the sales trend over the years and a bar chart showing the distribution of sales for each year based on the given data.

This lab manual has been updated by

Prof. Sukrati Chaturvedi
(Sukrati.chaturvedi@ggnindia.dronacharya.info)

Crosschecked By
HOD Applied Sciences and Humanities

Please spare some time to provide your valuable feedback.