George College, Kolkata

LAB MANUAL

Bachelor of Computer Applications.

Semester : 4TH

Subject: PYTHON PROGRAMMING LAB

Course Code : BCAC493

VER 2024

DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.
2. All the students should sit according to their roll numbers starting from their left to right.
3. All the students are supposed to enter the terminal number in the log book.
4. Do not change the terminal on which you are working.
5. All the students are expected to get at least the algorithm of the program/concept to be implemented.
6. Strictly observe the instructions given by the teacher/Lab Instructor.
7. Do not disturb machine Hardware / Software Setup.

Instruction for Laboratory Teachers:

1. Submission related to whatever lab work has been completed should be done during the next lab session along with signing the index.

2. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

3. Continuous assessment in the prescribed format must be followed.

Practical: (Python Programming Lab) Skills to be developed:

Intellectual skills:

1. **Skill to understand the python environment and different data types.**
2. **Knowledge of advanced data structures and their operations in python.**
3. **Ability to implement algorithms to perform various operations on data structures in python**

List of Practical:

1. Program to display name, college name and other messages.

2. Program using type() function to display different basic data types in python.

3. Program to input two numbers the find larger / smaller number.

4. Program to input three numbers and find largest and smallest number.

5. Program to determine Armstrong number / palindrome number.

6. Program to display the terms of a Fibonacci series.

7. Program to work with string.

8. Program to find largest / smallest number in a list/tuple.

9. Program to work with dictionary.

10. Program to create class / objects in python

11. Program to work with class constructors and other elements of OOP in python.

12. Programs involving NumPy with Pandas and Matplotlib.

11. Practice package installation and other basic application usage.

Experiment 1

Q. Program to display name, college name and other messages.

```
# Prompt the user to enter their name
    name = input("Enter your name: ")
    # Prompt the user to enter their college name
    college = input("Enter your college name: ")
# Display the name, college name, and other messages
    print("Welcome, " + name + "!")
    print("You are attending " + college + ".")
```

Experiment 2

Q. program that demonstrates the use of the `type()` function to display different basic data types:

```
def main():
    # Integer
    integer_var = 10
    print("Type of integer_var:", type(integer_var))
    # Float
    float_var = 3.14
    print("Type of float_var:", type(float_var))
    # String
    string_var = "Hello, world!"
    print("Type of string_var:", type(string_var))
    # Boolean
    bool_var = True
    print("Type of bool_var:", type(bool_var))
    # List
    list_var = [1, 2, 3, 4, 5]
    print("Type of list_var:", type(list_var))
    # Tuple
    tuple_var = (1, 2, 3, 4, 5)
    print("Type of tuple_var:", type(tuple_var))
    # Dictionary
    dict_var = {"name": "John", "age": 30}
    print("Type of dict_var:", type(dict_var))

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

Experiment 2.1

Q. Python program that demonstrates the use of different operators:

```
# Arithmetic operators
x = 10
y = 3
print("Arithmetic Operators:")
print("x + y =", x + y)  # Addition
print("x - y =", x - y)  # Subtraction
print("x * y =", x * y)  # Multiplication
print("x / y =", x / y)  # Division
print("x // y =", x // y)  # Floor Division
print("x % y =", x % y)  # Modulus
print("x ** y =", x ** y)  # Exponentiation
```



```
# Comparison operators
a = 5
b = 7
print("\nComparison Operators:")
print("a < b is", a < b)  # Less than
print("a > b is", a > b)  # Greater than
print("a == b is", a == b)  # Equal to
print("a != b is", a != b)  # Not equal to
print("a <= b is", a <= b)  # Less than or equal to
print("a >= b is", a >= b)  # Greater than or equal to
```

```
# Logical operators
p = True
q = False
print("\nLogical Operators:")
print("p and q is", p and q)  # Logical AND
print("p or q is", p or q)  # Logical OR
print("not p is", not p)  # Logical NOT

# Assignment operators
num = 10
num += 5  # Equivalent to num = num + 5
print("\nAssignment Operators:")
print("num after adding 5 is", num)

# Bitwise operators
m = 60  # 60 = 0011 1100
n = 13  # 13 = 0000 1101
print("\nBitwise Operators:")
print("m & n =", m & n)  # Bitwise AND
print("m | n =", m | n)  # Bitwise OR
print("m ^ n =", m ^ n)  # Bitwise XOR
print("~m =", ~m)  # Bitwise NOT
print("m << 2 =", m << 2)  # Bitwise Left Shift
print("m >> 2 =", m >> 2)  # Bitwise Right Shift
```

Experiment 3

Q. Program to input two numbers the find larger / smaller number.

```
def main():
    # Input the first number
    num1 = float(input("Enter the first number: "))
    # Input the second number
    num2 = float(input("Enter the second number: "))
    # Find the larger number
    larger_num = max(num1, num2)
    # Find the smaller number
    smaller_num = min(num1, num2)
    # Print the larger and smaller numbers
    print("Larger number:", larger_num)
    print("Smaller number:", smaller_num)

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

Experiment 4

Program to input three numbers and find largest and smallest number.

```
def main():
    # Input the first number
    num1 = float(input("Enter the first number: "))
    # Input the second number
    num2 = float(input("Enter the second number: "))
    # Input the third number
    num3 = float(input("Enter the third number: "))
    # Find the largest number
    largest_num = max(num1, num2, num3)
    # Find the smallest number
    smallest_num = min(num1, num2, num3)
    # Print the largest and smallest numbers
    print("Largest number:", largest_num)
    print("Smallest number:", smallest_num)

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

Experiment 5:

Q. Program to determine Armstrong number

```
def is_armstrong_number(number):
    # Count the number of digits in the number
    num_digits = len(str(number))

    # Initialize sum to store the result
    sum_of_digits = 0
    temp_number = number

    # Calculate sum of digits each raised to the power of the number of digits
    while temp_number > 0:
        digit = temp_number % 10
        sum_of_digits += digit ** num_digits
        temp_number //= 10
        # Check if the number is Armstrong or not
    return sum_of_digits == number

def main():
    # Input a number from the user
    number = int(input("Enter a number: "))
        # Check if the number is Armstrong or not
    if is_armstrong_number(number):
        print(number, "is an Armstrong number.")
    else:
        print(number, "is not an Armstrong number.")
```

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

Experiment 6:

Q. Program to determine palindrome number.

```
def is_palindrome_number(number):
    # Convert the number to a string
    num_str = str(number)

    # Reverse the string
    reversed_str = num_str[::-1]

    # Check if the original string is equal to its reverse
    return num_str == reversed_str

def main():
    # Input a number from the user
    number = int(input("Enter a number: "))

    # Check if the number is a palindrome or not
    if is_palindrome_number(number):
        print(number, "is a palindrome number.")
    else:
        print(number, "is not a palindrome number.")

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

:: Assignment 1 ::

1. **Write a Python program that checks whether a given number is positive, negative, or zero. Prompt the user to enter a number and then display whether it is positive, negative, or zero.**

2. **Write a Python program that checks whether a given number is even or odd. Prompt the user to enter a number and then display whether it is even or odd.**

3. **Write a Python program that checks whether a given year is a leap year or not. Prompt the user to enter a year and then display whether it is a leap year or not.**

4. **Write a Python program that classifies a person's age into different categories such as infant, child, teenager, adult, or senior citizen. Prompt the user to enter their age and then display the corresponding category.**

5. **Write a Python program that checks whether a given set of three numbers can form a valid triangle and if so, whether it is an equilateral, isosceles, or scalene triangle. Prompt the user to enter three numbers and then display the type of triangle.**

6. **Write a Python program that implements a simple calculator with operations such as addition, subtraction, multiplication, and division. Prompt the user to enter two numbers and the desired operation, and then perform the calculation.**

7. **Write a Python program that calculates the grade of a student based on their marks in a subject. Prompt the user to enter the marks scored by the student and then display the corresponding grade according to a predefined grading system.**

8. **Write a Python program that finds the largest of three given numbers. Prompt the user to enter three numbers and then display the largest among them.**

9. **Write a Python program that determines the number of days in a given month of a year. Prompt the user to enter the month and year, and then display the number of days.**

10. **Write a Python program that checks whether a given character is a vowel or a consonant. Prompt the user to enter a character and then display whether it is a vowel or a consonant.**

:: Experiment 7 ::

Q. Program to display the terms of a Fibonacci series.

```
def fibonacci_series(num_terms):
    # Initialize the first two terms of the Fibonacci series
    fib_series = [0, 1]

    # Generate the Fibonacci series up to the specified number of terms
    for i in range(2, num_terms):
        next_term = fib_series[i - 1] + fib_series[i - 2]
        fib_series.append(next_term)
    return fib_series
```



```
def main():
    # Input the number of terms for the Fibonacci series
    num_terms = int(input("Enter the number of terms for the Fibonacci series: "))
    # Check if the input is valid
    if num_terms <= 0:
        print("Number of terms should be a positive integer.")
        return
    # Display the Fibonacci series
    fib_series = fibonacci_series(num_terms)
    print("Fibonacci series:")
    for term in fib_series:
        print(term, end=" ")
```

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

:: Assignment 2 ::

1. **Write a program that takes a list of numbers as input and outputs the sum of all the numbers in the list. Use a loop to iterate through the list and accumulate the sum.**

2. **Create a program that prompts the user for an integer and then calculates and prints the factorial of that number. Use a loop to calculate the factorial.**

3. **Write a program that takes an integer input from the user and prints the multiplication table for that number up to 10. Use nested loops to generate the table.**

4. **Create a program that prompts the user to enter a string and then counts and prints the number of vowels (a, e, i, o, u) in that string. Use a loop to iterate through each character in the string and count the vowels.**

5. **Write a program that prompts the user for an integer and then prints all the prime numbers less than or equal to that integer. Use nested loops and the concept of prime numbers to accomplish this task.**

:: Experiment 8 ::

Q. Program to work with string.

```
def main():
    # Define a string
    my_string = "Hello, world!"

    # Print the string
    print("Original string:", my_string)

    # Access individual characters in the string
    print("First character:", my_string[0])
    print("Last character:", my_string[-1])

    # Slice the string to get a substring
    substring = my_string[7:]
    print("Substring:", substring)

    # Concatenate strings
    new_string = my_string + " Have a nice day!"
    print("Concatenated string:", new_string)

    # Convert the string to uppercase and lowercase
    print("Uppercase:", my_string.upper())
    print("Lowercase:", my_string.lower())

    # Check if a substring is present in the string
    if "world" in my_string:
```

```
    print("Substring 'world' is present in the string.")
else:
    print("Substring 'world' is not present in the string.")

# Split the string into a list of substrings
words = my_string.split(",")
print("Words in the string:", words)

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

:: Assignment 3 ::

1. **Write a Python program that takes a string input from the user and prints the reverse of that string.**

2. **Create a Python program that checks if a given string is a palindrome or not. A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward.**

3. **Develop a Python program that takes a string and a character as input and counts the number of occurrences of that character in the string.**

4. **Write a Python program that takes a sentence as input and counts the number of words in it.**

5. **Create a Python program that takes a string input and performs various manipulations, such as converting it to uppercase, lowercase, and title case, and also replacing certain characters with others based on user input.**

:: Experiment 9 ::

Q. Program to find largest / smallest number in a list

```
def find_largest_smallest(numbers):
    # Check if the list is empty
    if not numbers:
        print("List is empty.")
        return None, None

    # Initialize variables to hold the largest and smallest numbers
    largest = numbers[0]
    smallest = numbers[0]

    # Iterate through the list to find the largest and smallest numbers
    for num in numbers:
        if num > largest:
            largest = num
        elif num < smallest:
            smallest = num

    return largest, smallest

def main():
    # Input list of numbers from the user
    numbers = [float(x) for x in input("Enter a list of numbers separated by space: ").split()]

    # Find the largest and smallest numbers in the list
```

```
    largest, smallest = find_largest_smallest(numbers)

    # Print the largest and smallest numbers
    if largest is not None and smallest is not None:
        print("Largest number:", largest)
        print("Smallest number:", smallest)

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

:: Assignment 4 ::

1. **Write a program that takes a list of numbers as input and performs various operations such as finding the sum, average, maximum, minimum, and sorting the list.**

2. **Create a program that takes a list of strings as input and filters out words that contain a specific letter or have a length less than a given threshold.**

3. **Write a program that concatenates two lists and removes duplicates, then sorts the resulting list.**

4. **Implement a program that searches for a given element in a list and returns the index if found, or a message if not found.**

5. **Create a program that generates a list of squares or cubes of numbers from 1 to n using list comprehension.**

Experiment 10

Q. Program to find largest / smallest number in a tuple.

```
def find_largest_smallest_tuple(numbers):
    # Check if the tuple is empty
    if not numbers:
        print("Tuple is empty.")
        return None, None

    # Initialize variables to hold the largest and smallest numbers
    largest = numbers[0]
    smallest = numbers[0]

    # Iterate through the tuple to find the largest and smallest numbers
    for num in numbers:
        if num > largest:
            largest = num
        elif num < smallest:
            smallest = num

    return largest, smallest

def main():
    # Input tuple of numbers from the user
    numbers = tuple(float(x) for x in input("Enter a tuple of numbers separated by space: ").split())

    # Find the largest and smallest numbers in the tuple
```

```
    largest, smallest = find_largest_smallest_tuple(numbers)

    # Print the largest and smallest numbers
    if largest is not None and smallest is not None:
        print("Largest number:", largest)
        print("Smallest number:", smallest)

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

:: Assignment 5 ::

1. **Write a program that takes two tuples of integers as input and returns a tuple containing the element-wise sum of the corresponding elements of the input tuples.**
2. **Write a program that takes a tuple containing a person's name and age, and unpacks it to print out a message stating the person's name and age.**
3. **Write a program that swaps the values of two variables using tuples.**
4. **Write a program that takes a list of tuples, each containing a student's name and their score, and sorts the list by score in descending order.**
5. **Write a program that concatenates two tuples into one.**

Experiment 11

Q. Program to work with dictionary.

```
def main():
    # Create a dictionary
    student = {"name": "John", "age": 20, "major": "Computer Science"}

    # Print the dictionary
    print("Original dictionary:", student)

    # Access value by key
    print("Name:", student["name"])
    print("Age:", student["age"])



    # Add a new key-value pair
    student["grade"] = "A"
    print("Dictionary after adding 'grade' key:", student)

    # Modify the value of an existing key
    student["age"] = 21
    print("Dictionary after modifying 'age' value:", student)

    # Remove a key-value pair
    del student["major"]
    print("Dictionary after removing 'major' key:", student)
```

```
# Check if a key exists in the dictionary
    if "name" in student:
        print("'name' key exists in the dictionary.")
    else:
        print("'name' key does not exist in the dictionary.")

    # Iterate through the keys of the dictionary
    print("Keys in the dictionary:")
    for key in student.keys():
        print(key)

    # Iterate through the values of the dictionary
    print("Values in the dictionary:")
    for value in student.values():
        print(value)

    # Iterate through key-value pairs of the dictionary
    print("Key-value pairs in the dictionary:")
    for key, value in student.items():
        print(key, ":", value)

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

:: Assignment 6 ::

1. **Write a program that takes a sentence as input and counts the occurrences of each word. Use a dictionary to store the word counts and then print out each word along with its count.**

2. **Create a program that acts as a simple contact manager. Allow users to add contacts with their names and phone numbers. Use a dictionary with names as keys and phone numbers as values. Implement functionalities to add, delete, and search for contacts.**

3. **Build a program that keeps track of stock prices. Use a dictionary to store stock symbols as keys and their corresponding prices as values. Implement functionalities to add new stocks, update prices, and display the current prices for all stocks.**

4. **Develop a quiz game where questions are stored in a dictionary. Each question should have its corresponding answer. Allow the user to select a category, then randomly select a question from that category. Prompt the user for an answer and check if it's correct.**

5. **Create a simple language translator using dictionaries. Store pairs of words or phrases in dictionaries for different languages. Prompt the user to enter a word or phrase in one language and translate it to another language by looking it up in the dictionaries.**

:: Experiment 12 ::

Q. Program to create class / objects in python

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print("Car Info:")
        print("Make:", self.make)
        print("Model:", self.model)
        print("Year:", self.year)

def main():
    # Create objects of the Car class
    car1 = Car("Toyota", "Camry", 2020)
    car2 = Car("Honda", "Accord", 2019)

    # Display information about the cars
    car1.display_info()
    print()
    car2.display_info()

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

In this program:

We define a class named Car with attributes make, model, and year.

The __init__ method is the constructor, which initializes the object's attributes.

The display_info method prints information about the car.

In the main function, we create two objects (car1 and car2) of the Car class using the constructor.

We then call the display_info method on each object to display information about the cars.

:: Experiment 13 ::

Program to work with class constructors

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print("Name:", self.name)
        print("Age:", self.age)

def main():
    # Create objects of the Person class using the constructor
    person1 = Person("Alice", 25)
    person2 = Person("Bob", 30)

    # Display information about the persons
    print("Person 1:")
    person1.display_info()
    print()
    print("Person 2:")
    person2.display_info()

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

In this program:

- We define a class named `Person` with attributes `name` and `age`.

- The `__init__` method is the constructor, which initializes the object's attributes `name` and `age`.

- The `display_info` method prints information about the person.

- In the `main` function, we create two objects (`person1` and `person2`) of the `Person` class using the constructor with different parameters.

- We then call the `display_info` method on each object to display information about the persons.

Experiment 14.

Q. program that demonstrates class inheritance:

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

    def speak(self):
        pass  # Abstract method, to be overridden by subclasses

class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"



class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"

def main():
    dog = Dog("Buddy")
    cat = Cat("Whiskers")

    print(dog.speak())  # Output: Buddy says Woof!
    print(cat.speak())  # Output: Whiskers says Meow!

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

In this program:

- We define a base class `Animal` with an `__init__` method to initialize the `name` attribute and a `speak` method, which is an abstract method (implemented with `pass`), meant to be overridden by subclasses.

- We define two subclasses `Dog` and `Cat` which inherit from the `Animal` class.

- Each subclass implements its own version of the `speak` method to provide a specific sound.

- In the `main` function, we create objects of the `Dog` and `Cat` classes and call the `speak` method on each object.

This demonstrates how inheritance allows us to create specialized classes that inherit properties and behavior from a base class.

:: Assignment 7 ::

1. **Create classes for a bank account, including functionalities like deposit, withdraw, and balance inquiry.**

2. **Design classes for a library catalog, including books, authors, and patrons. Implement methods for checking out, returning, and searching for books.**

3. **Develop classes for different types of vehicles (car, bike, truck) and a rental management system. Include functionalities like renting, returning, and calculating rental fees.**

4. **Build classes for products, customers, and a shopping cart. Implement methods for adding/removing items, calculating total cost, and processing orders.**

5. **Design classes for students, courses, and instructors. Implement functionalities for enrolling students in courses, assigning grades, and generating transcripts.**

:: Experiment 15 ::

Q. program that demonstrates basic usage of NumPy in Python:

```
import numpy as np

def main():
    # Create a 1D array
    arr1d = np.array([1, 2, 3, 4, 5])
    print("1D array:", arr1d)

    # Create a 2D array
    arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
    print("\n2D array:")
    print(arr2d)

    # Perform arithmetic operations on arrays
    print("\nArray addition:")
    print(arr1d + 10)
    print("\nArray multiplication:")
    print(arr1d * 2)

    # Perform element-wise operations on arrays
    print("\nElement-wise square:")
    print(np.square(arr1d))
```

```
# Array indexing and slicing
    print("\nArray indexing:")
    print("First element:", arr1d[0])
    print("Slice of array:", arr1d[1:4])

    # Reshaping arrays
    arr2d_reshaped = arr2d.reshape(1, 9)
    print("\nReshaped 2D array:")
    print(arr2d_reshaped)

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


In this program:

- We import NumPy as `np`.

- We create 1D and 2D NumPy arrays using `np.array`.

- We perform arithmetic operations (`+`, `*`) and element-wise operations (`np.square`) on arrays.

- We demonstrate array indexing and slicing.

- We reshape a 2D array using the `reshape` method.

You can run this program to see the output. Make sure you have NumPy installed (`pip install numpy`).

Experiment 16:

Q. programs demonstrating basic usage of Pandas in Python:

```
# Example 1: Creating a DataFrame and Performing Basic Operation
import pandas as pd
def main():
    # Create a DataFrame from a dictionary
    data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
            'Age': [25, 30, 35, 40],
            'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}
    df = pd.DataFrame(data)
```



```
    # Display the DataFrame
    print("DataFrame:")
    print(df)
    # Display basic information about the DataFrame
    print("\nDataFrame info:")
    print(df.info())
    # Display statistical summary of the DataFrame
    print("\nStatistical summary:")
    print(df.describe())

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

Experiment 17:

Example 2: Reading Data from a CSV File

```
import pandas as pd

def main():
    # Read data from a CSV file into a DataFrame
    df = pd.read_csv('data.csv')

    # Display the first few rows of the DataFrame
    print("First few rows of the DataFrame:")
    print(df.head())

    # Display basic information about the DataFrame
    print("\nDataFrame info:")
    print(df.info())

    # Display statistical summary of the DataFrame
    print("\nStatistical summary:")
    print(df.describe())

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

Experiment 18:

Example 3: Data Manipulation with Pandas

```
import pandas as pd
def main():
    # Read data from a CSV file into a DataFrame
    df = pd.read_csv('data.csv')
    # Filter the DataFrame based on a condition
    filtered_df = df[df['Age'] > 30]
    # Display the filtered DataFrame
    print("Filtered DataFrame:")
    print(filtered_df)
    # Sort the DataFrame by a column
    sorted_df = df.sort_values(by='Age', ascending=False)
    # Display the sorted DataFrame
    print("\nSorted DataFrame:")
    print(sorted_df)
    # Group the DataFrame by a column and perform aggregation
    grouped_df = df.groupby('City').agg({'Age': 'mean'})
    # Display the grouped DataFrame
    print("\nGrouped DataFrame:")
    print(grouped_df)

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

These examples demonstrate basic operations such as creating DataFrames, reading data from CSV files, and performing data manipulation tasks like filtering, sorting, and grouping using Pandas. You can run these programs with your data or modify them according to your requirements.

:: Experiment 19 ::

Q. programs demonstrating basic usage of Matplotlib in Python.

```
### Example 1: Line Plot
import matplotlib.pyplot as plt

def main():
    # Data
    x = [1, 2, 3, 4, 5]
    y = [2, 3, 5, 7, 11]

    # Plot
    plt.plot(x, y)
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Line Plot')
    plt.grid(True)
    plt.show()

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

Experiment 20:

```
### Example 2: Scatter Plot
import matplotlib.pyplot as plt

def main():
    # Data
    x = [1, 2, 3, 4, 5]
    y = [2, 3, 5, 7, 11]

    # Plot
    plt.scatter(x, y, color='red', marker='o')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Scatter Plot')
    plt.grid(True)
    plt.show()

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

Experiment 21:

```
### Example 3: Bar Plot
import matplotlib.pyplot as plt

def main():
    # Data
    x = ['A', 'B', 'C', 'D', 'E']
    y = [10, 20, 15, 25, 30]

    # Plot
    plt.bar(x, y, color='blue')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Bar Plot')
    plt.grid(True)
    plt.show()

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


These examples demonstrate basic plots such as line plot, scatter plot, and bar plot using Matplotlib in Python. You can run these programs to see the plots or modify them according to your requirements.

:: Assignment 8 ::

Data Analysis:

Load a dataset using pandas.

Perform basic data exploration (e.g., checking for missing values, data types).

Calculate descriptive statistics (mean, median, mode, standard deviation, etc.).

Visualize data using pandas built-in plotting capabilities.

Data Cleaning:

Load a dataset with missing or inconsistent data.

Clean the data by filling missing values or removing rows/columns with missing values.

Standardize or normalize data if necessary.

Handle outliers.

Data Manipulation:

Load multiple datasets and merge or join them using pandas.

Perform groupby operations to aggregate data.

Create new columns based on existing data (e.g., calculations, transformations).

Time Series Analysis:

Load a time series dataset.

Resample data to different time frequencies (e.g., daily to monthly).

Calculate rolling statistics (e.g., rolling mean, rolling standard deviation).

Plot time series data and analyze trends, seasonality, and anomalies.

Advanced Analysis:

Perform advanced statistical analysis using pandas (e.g., hypothesis testing, regression analysis).

Apply machine learning algorithms (e.g., linear regression, decision trees) using pandas for data preprocessing and feature engineering.

Implement custom functions or methods to solve specific analytical problems.